МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 36.00.00.000 ПЗ

Група ІПм-24-2

Мельник Дмитро

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Мельник Дмитро Васильович

(прізвище, ім'я, по батькові)

УДК 004.9

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі та методи запобігання інсайдерським атакам у розподілених обчислювальних середовищах

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Мельник Д.В.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Процюк Василь Романович, к.т.н., доцент

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

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2025 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Мельнику Дмитру Васильовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі та методи запобігання інсайдерським атакам у розподілених обчислювальних середовищах ”

керівник проекту (роботи) Процюк Василь Романович, к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 05 ” листопада 2025 р. № 695/7

2. Строк подання студентом проекту (роботи) 15 грудня 2025 р.

3. Вихідні дані до проекту (роботи) Концепції та формальні моделі і методи побудови інформаційних технологій розподілених обчислювальних середовищ

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Дослідження предметної області запобігання атакам у обчислювальних середовищах
2. Дослідження та опис методів виявлення інсайдерських атак у розподілених середовищах
3. Інтеграція методів глибоко навчання для аналізу шаблонів використання пам'яті
4. Імплементація моделей та методології у фреймворк запобігання інсайдерським атакам

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Частка компаній, що використовують великі дані та ШІ у 2024 році, % (рис. 1.1)
2. Галузі, що застосовують великі дані та хмарні обчислення (рис. 1.2)
3. Концепція 3V для великих даних (рис. 1.3)
4. Розподіл вразливостей за мовами програмування (рис. 1.4)
5. Розподіл вразливостей по веб-серверах (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2025 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2025	виконано
2	Дослідження предметної області запобігання атакам у обчислювальних середовищах	01.10.2025	виконано
3	Дослідження та опис методів виявлення інсайдерських атак у розподілених середовищах	17.10.2025	виконано
4	Інтеграція методів глибоко навчання для аналізу шаблонів використання пам'яті	02.11.2025	виконано
5	4. Імплементація моделей та методології у фреймворк запобігання інсайдерським атакам	19.11.2025	виконано
6	Деталізація методів забезпечення безпеки на основі потоку завдань і LSTM	02.12.2025	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2025	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 86 с., 16 рис., 4 табл., 37 джерел.

Тема: Моделі та методи запобігання інсайдерським атакам у розподілених обчислювальних середовищах

Мета магістерської роботи: розроблення моделей і методів запобігання інсайдерським атакам у розподілених обчислювальних середовищах, заснованих на динамічному аналізі поведінки користувачів та використанні технологій глибокого навчання.

Об'єкт дослідження: процеси забезпечення інформаційної безпеки у розподілених обчислювальних середовищах, орієнтованих на обробку великих даних.

Предмет дослідження: моделі, методи та алгоритми запобігання інсайдерським атакам у розподілених обчислювальних середовищах із використанням технологій глибокого навчання та поведінкового аналізу.

Результати дослідження

В роботі розроблено архітектуру фреймворку безпеки, який забезпечує багаторівневий моніторинг поведінки користувачів і безпечну комунікацію між вузлами системи.

Висновок

Впровадження запропонованого підходу дозволяє своєчасно виявляти підозрілі дії користувачів, мінімізувати наслідки внутрішніх інцидентів безпеки та підвищити довіру до обчислювальних сервісів у середовищах великих даних

РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ, ВЕЛИКІ ДАНІ, ІНФОРМАЦІЙНА БЕЗПЕКА, ІНСАЙДЕРСЬКІ АТАКИ, ПОВЕДІНКОВИЙ АНАЛІЗ, ВИЯВЛЕННЯ ВТОРГНЕНЬ, ГЛИБОКЕ НАВЧАННЯ, ФРЕЙМВОРК БЕЗПЕКИ.

ABSTRACT

Master Thesis: 86 pp., 16 fig., 4 tab., 37 sources.

Topic: Models and methods for preventing insider attacks in distributed computing environments

The purpose of the master's thesis: to develop models and methods for preventing insider attacks in distributed computing environments based on dynamic analysis of user behavior and the use of deep learning technologies.

Object of research: processes of ensuring information security in distributed computing environments focused on processing big data.

Subject of research: models, methods and algorithms for preventing insider attacks in distributed computing environments using deep learning and behavioral analysis technologies.

Research results

The paper developed the architecture of a security framework that provides multi-level monitoring of user behavior and secure communication between system nodes.

Conclusion

The implementation of the proposed approach allows for timely detection of suspicious user actions, minimize the consequences of internal security incidents, and increase trust in computing services in big data environments

DISTRIBUTED COMPUTING, BIG DATA, INFORMATION SECURITY, INSIDER ATTACKS, BEHAVIORAL ANALYSIS, INTRUSION DETECTION, DEEP LEARNING, SECURITY FRAMEWORK.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

РОС - Розподілене обчислювальне середовище

CFG Control Flow Graph - граф потоку управління

MST - Minimum Spanning Tree

MSA - Minimum Spanning Arborescence

RNN - Recurrent Neural Network

LSTM - Long Short-Term Memory

IDS - Intrusion Detection System

CFI - Control Flow Integrity

LUT - Look-Up Table

TPM - Trusted Platform Module

TXT - Trusted Execution Technology

SECaaS - Security as a Service

ВСТУП

Актуальність теми.

Стрімкий розвиток інформаційних технологій, зростання обсягів даних та активне впровадження хмарних і розподілених обчислювальних систем зумовлюють підвищення вимог до забезпечення інформаційної безпеки. Сучасні системи великих даних характеризуються складною архітектурою, великою кількістю взаємодіючих вузлів і користувачів, що створює сприятливі умови для виникнення внутрішніх (інсайдерських) загроз. Інсайдерські атаки, здійснювані користувачами з легітимними правами доступу, становлять особливу небезпеку, оскільки часто залишаються непоміченими стандартними системами захисту, орієнтованими переважно на зовнішні вторгнення.

В умовах розподілених середовищ обробки даних особливої актуальності набуває побудова гнучких і адаптивних систем моніторингу, здатних виявляти аномалії поведінки користувачів, прогнозувати інциденти безпеки та автоматично реагувати на них. Сучасні підходи до безпеки вже не можуть обмежуватися класичними методами автентифікації та контролю доступу — необхідним стає використання методів машинного та глибокого навчання, здатних виявляти приховані закономірності у великих обсягах даних і підвищувати ефективність систем захисту.

Магістерська робота присвячена дослідженню моделей і методів запобігання інсайдерським атакам у розподілених обчислювальних середовищах шляхом інтеграції інтелектуальних технологій аналізу поведінки користувачів, виявлення аномалій та динамічного моніторингу потоків управління.

Проблема захисту інформаційних систем від інсайдерських атак набуває дедалі більшої ваги у зв’язку з поширенням хмарних технологій, мультиорендних архітектур і сервісних платформ, де контроль над усіма компонентами системи є розподіленим. Згідно з аналітичними звітами

провідних центрів кібербезпеки, до 30–40% інцидентів у корпоративних середовищах спричиняються саме інсайдерами — особами, які мають або мали внутрішній доступ до систем.

Традиційні механізми безпеки, що ґрунтуються на статичному контролі доступу та сигнатурному виявленні атак, виявляються неефективними у випадку внутрішніх загроз, оскільки не враховують динамічний контекст поведінки користувача, його рольову активність та закономірності взаємодії з ресурсами системи.

Зростання складності архітектур великих даних, збільшення кількості вузлів та потоків обміну інформацією вимагає нових підходів до моніторингу, заснованих на інтелектуальних моделях, здатних до самонавчання та адаптації. Використання глибоких нейронних мереж, зокрема моделей з довгостроковою пам’яттю (LSTM), дозволяє здійснювати аналіз часових залежностей і шаблонів поведінки в розподілених системах, забезпечуючи більш точне прогнозування потенційних атак.

Таким чином, актуальність роботи зумовлена потребою у створенні науково обґрунтованих моделей і методів запобігання інсайдерським атакам, які б поєднували принципи поведінкової аналітики, машинного навчання та архітектурної безпеки для забезпечення цілісності, конфіденційності та надійності розподілених обчислювальних середовищ.

Метою магістерської роботи є розроблення моделей і методів запобігання інсайдерським атакам у розподілених обчислювальних середовищах, заснованих на динамічному аналізі поведінки користувачів та використанні технологій глибокого навчання.

Об’єктом дослідження є процеси забезпечення інформаційної безпеки у розподілених обчислювальних середовищах, орієнтованих на обробку великих даних.

Предметом дослідження є моделі, методи та алгоритми запобігання інсайдерським атакам у розподілених обчислювальних середовищах із використанням технологій глибокого навчання та поведінкового аналізу.

Для досягнення поставленої мети у роботі вирішено такі основні завдання:

1. Провести аналіз сучасних підходів до забезпечення безпеки в розподілених системах і виявлення інсайдерських атак.

2. Визначити основні типи внутрішніх загроз і розробити класифікацію інсайдерських атак у контексті екосистем великих даних.

3. Дослідити вектори внутрішніх атак, пов’язані з поведінковими та архітектурними вразливостями систем.

4. Інтегрувати методи глибокого навчання (PCA, LSTM) у процеси виявлення аномалій для підвищення точності прогнозування атак.

5. Запропонувати архітектуру фреймворку безпеки для динамічного моніторингу та запобігання інсайдерським загрозам.

Методи дослідження

У роботі використано такі методи:

- методи системного та порівняльного аналізу для вивчення архітектур безпеки розподілених систем;

- методи математичного моделювання та машинного навчання для побудови моделей поведінкової аналітики;

- алгоритми глибокого навчання (LSTM, PCA) для аналізу часових залежностей та шаблонів доступу до пам’яті;

- методи динамічного аналізу потоків управління для виявлення аномалій у поведінці користувачів.

Наукова новизна отриманих результатів

Розроблено комплексну модель запобігання інсайдерським атакам у розподілених обчислювальних середовищах, що поєднує динамічний поведінковий аналіз і методи глибокого навчання. Запропоновано метод виявлення вторгнень, заснований на аналізі шаблонів доступу до пам’яті та системних викликів із використанням моделей LSTM.

Практичне значення отриманих результатів

Розроблені моделі та фреймворк можуть бути використані:

- у хмарних і гібридних середовищах для підвищення стійкості до внутрішніх атак;

- у системах моніторингу корпоративних IT-інфраструктур;

- як основа для подальших досліджень у галузі поведінкового аналізу користувачів та кіберзахисту розподілених систем.

Структура магістерської роботи. Представлена робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 86 сторінок, і містить 16 рисунків, 4 таблиці, перелік використаних джерел із 37 позицій.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАПОБІГАННЯ АТАКАМ У РОЗПОДІЛЕНИХ ОБЧИСЛЮВАЛЬНИХ СЕРЕДОВИЩАХ АНАЛІЗУ ВЕЛИКИХ ДАНИХ

1.1. Хмарні обчислення та середовища для аналізу великих даних

Хмарні обчислення визначаються як модель, що забезпечує всеперевершний, мережевий, on-demand доступ до спільного пулу конфіґурованих обчислювальних ресурсів — мереж, серверів, сховищ, програмного забезпечення та сервісів — з оперативним наданням і звільненням за принципом самообслуговування та з мінімальною взаємодією з постачальником. Еластичність інфраструктури стала каталізатором появи концепції «великі дані» (Big Data), що описує ексабайтні множини структурованої та неструктурованої інформації, які традиційні засоби обробити нездатні.

Аналітичний попит з боку фінансового, телекомунікаційного, ритейл-, медичного та програмного секторів спричинив конвергенцію статистичного аналізу й машинного навчання в нову міждисциплінарну парадигму — науку про дані (Data Science). Її інструментарій імплементує алгоритми, що мають видавати результати майже в режимі реального часу, що, у свою чергу, потребує високопродуктивного середовища для зберігання, керування та обробки петабайтних масивів.

Розподілене обчислювальне середовище (РОС), сформоване ще у 1990-х, постало як домінантна технологія для оркестрації обчислень і комунікацій у кластерах commodity-серверів [5]. Його ключові складові — служба імен, аутентифікації, синхронізації часу, комунікаційний протокол і розподілена файлова система — у поєднанні з горизонтальним масштабуванням забезпечують відмовостійке сховище ексабайтного рівня та обробку терабайтів даних за хвилини. Саме тому сучасні стеки великих даних

(Hadoop, Spark, Google File System + MapReduce) реалізуються поверх POC і активно експлуатуються як у державному, так і в корпоративному секторах.

Провідні сценарії використання включають інтелектуальний аналіз неструктурованих даних і аналітику в реальному часі. Гіпермасштабні оператори (Google, Amazon, Microsoft) пропонують публічні та приватні хмари, що слугують інфраструктурною основою для сторонніх big-data сервісів. За прогнозами, глобальний ринок big-data і бізнес-аналітики зросте з ≈ 300 млрд USD у 2026 р. до > 500 млрд USD у 2030 р., що підтверджує тренд консолідації даних у хмарних середовищах.

Рис. 1.1. Частка компаній, що використовують великі дані та ШІ у 2024 році, %

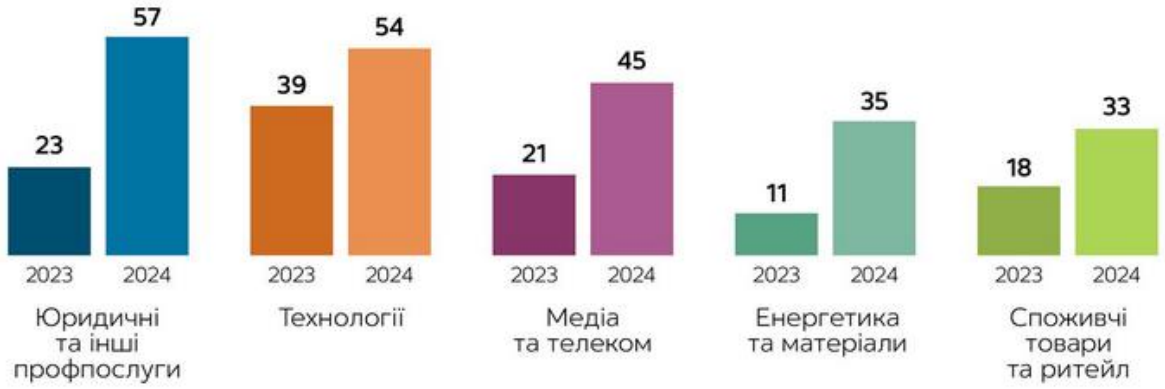


Рис. 1.2. Галузі, що застосовують великі дані та хмарні обчислення

1.2. Виклики та архітектурні проблеми безпеки у розподілених середовищах великих даних

Попри значну популярність та експоненційне зростання ринку, тренд великих даних (Big Data) нерозривно пов'язаний із низкою викликів та ризиків. На початковому етапі архітектори та розробники рішень для великих даних зосереджувалися переважно на параметрах продуктивності. Проте, притаманні характеристики великих даних, зокрема швидкість (Velocity), обсяг (Volume) та різноманітність (Variety), суттєво акцентують проблеми безпеки та конфіденційності.

Рис. 1.3. Концепція 3V для великих даних

Концепція "3V big data" описує три ключові характеристики, які відрізняють великі дані від традиційних наборів даних. Ці три "V" були вперше визначені у 2001 році:

- Обсяг (Volume) - величезна кількість даних, які генеруються та зберігаються. Традиційні бази даних не можуть ефективно обробляти такі масиви інформації.

- Швидкість (Velocity) - швидкість, з якою дані створюються, збираються та обробляються. Потоки даних є масивними та безперервними,

часто вимагаючи обробки в режимі реального часу (наприклад, дані з соціальних мереж, датчиків або онлайн-транзакцій).

- Різноманітність (Variety) - велика кількість різних типів і форматів даних. Це включає структуровані дані (наприклад, реляційні бази даних), напівструктуровані (наприклад, XML або JSON файли) та неструктуровані дані (наприклад, текстові документи, зображення, аудіо- та відеофайли).

Пізніше до цих трьох характеристик часто додають ще: достовірність (Veracity) - якість і точність даних, що важливо для прийняття правильних рішень і цінність (Value) - можливість перетворення великих обсягів даних на корисну інформацію та бізнес-цінність.

1.2.1. Проблематика довіри та вразливості систем

У хмарній моделі «Infrastructure-as-a-Service» та у фреймворках великих даних кінцевий споживач формально втрачає прямий контроль над фізичним рівнем зберігання, що змушує його оперувати категорією довіри (trust-by-default) до постачальника. Дана довіра припускає відсутність як злочинного інсайдера, так і скомпрометованих компонентів у ланцюгу постачання програмного забезпечення. Проте емпіричні дослідження показують, що складні socio-технічні системи неминуче містять атакові поверхні: від привілейованих користувачів (insider threats) до архітектурних вразливостей, що експлуатуються нульовими днями.

У контексті повсюдної комодитизації аналітичних інструментів екстракція цінності з даних стала тривіальною операцією, що підвищує чутливість користувачів до ризику втрати конфіденційності. Віддалене резидування інформації в поєднанні з експоненціальним зростанням кількості кібератак (річний приріст > 15 % за даними ENISA) трансформує безпеку в критичний фактор конвергенції хмарних і big-data екосистем.

На рис. 1.4 наведено емпіричний розподіл уразливостей за мовами програмування (ASP.NET, Java, PHP) з ранжуванням Low/Medium/High, що ілюструє heterogeneity атакової поверхні в типовому стеку великих даних.

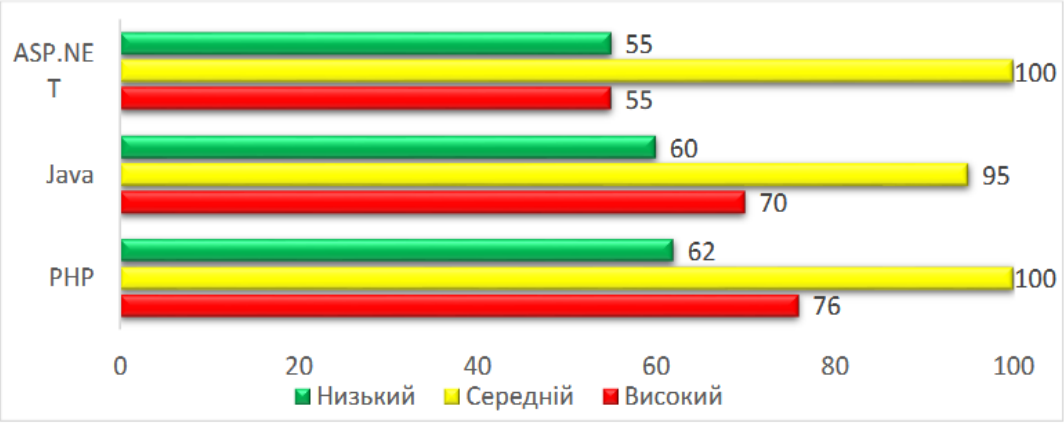


Рис. 1.4. Розподіл вразливостей за мовами програмування

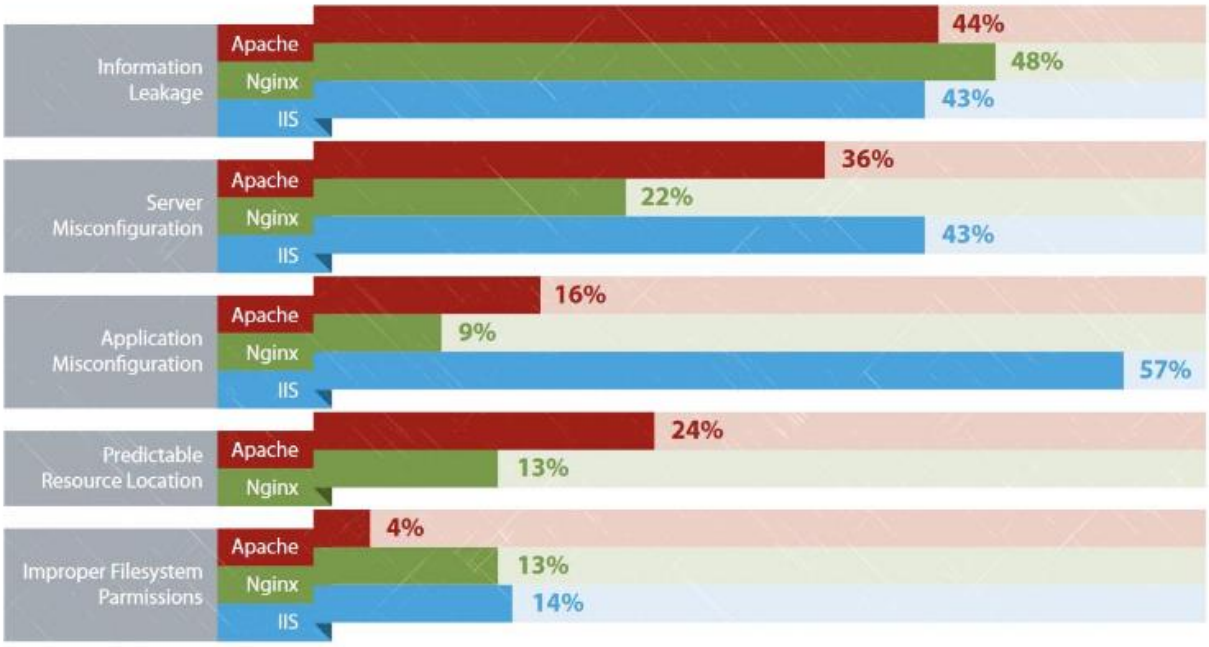


Рис. 1.5. Розподіл вразливостей по веб-серверах

1.2.2. Недосконалість інвестицій та архітектурні недоліки

Провідні вендори big-data екосистем (Hortonworks, Cloudera, IBM, SAS) формально декларують стратегічний пріоритет безпеки й конфіденційності, однак емпіричні оцінки витрат показують, що збитки від реалізованих уразливостей систематично перевищують обсяги інвестицій у превентивні заходи. Вказаний дисбаланс індикативний для ембріональної зрілості галузевої практики безпеки великих даних, де домінантною залишається

реактивна модель «виявлення інцидентів» (intrusion-detection) замість архітектурного усунення кореневих атакових поверхонь.

У privacy-first парадигмі централізована модель збору й резидування даних породжує інформаційну асиметрію: постачальник зберігає повне домінування над користувацьким атрибутивним простором, що створює можливість для функціонального розповсюдження даних (secondary use) та їхньої прихованої монетизації без зовнішнього аудиту. Отже, фундаментальна архітектурна вада полягає не лише в технічній уразливості, а й у економічному конфлікті інтересів, що підриває довіру до хмарно-базованих big-data сервісів.

1.2.3. Шляхи покращення безпеки

Для підвищення рівня безпеки в екосистемах великих даних необхідне впровадження комплексних підходів, що охоплюють наступні аспекти:

- гомоморфне шифрування (Homomorphic Encryption) дозволяє виконувати обчислення безпосередньо над зашифрованими даними, що забезпечує конфіденційність навіть під час обробки на хмарних серверах.

- диференційна конфіденційність (Differential Privacy) - забезпечення можливості отримання точних агрегованих результатів запитів шляхом додавання дозованого "шуму" до наборів даних, що унеможливлює ідентифікацію окремих осіб.

- децентралізоване управління доступом, використання технологій, таких як блокчейн (Blockchain), для створення незмінного та прозорого реєстру політик доступу та аудиту даних, зменшуючи залежність від єдиного центру довіри (постачальника хмари).

- безпека на рівні мікросервісів, впровадження суворих механізмів автентифікації та авторизації між окремими компонентами стеку Великих Даних, використовуючи принцип найменших привілеїв (Principle of Least Privilege).

1.3. Комплексний аналіз викликів безпеки та інсайдерських атак в екосистемах великих даних

Статистичні вибірки фіксують експоненціальне зростання кількості шкідливих програм (malware specimens) у глобальному кіберпросторі, що індикативно для ескалації атак-ас-сервіс (AaaS) екосистеми. Жертвами комплексних кампаній стають організації всьєї шкали розмірів: від транснаціональних корпорацій (LinkedIn, Yahoo, Target, Sony Pictures) до малого й середнього бізнесу (SMB).

Розподіл основних типів шкідливих програм у світі станом на 2024 рік, на основі останніх статистичних даних подано на рис. 1.6.

Рис. 1.6. Розподіл основних типів шкідливих програм у світі (на 2024 рік)

Ця кругова діаграма відображає:

- Троянські програми (Trojans): 58%

- Програми-вимагачі (Ransomware): 22%

- Інші шкідливі програми (Other Malware) включаючи рекламні програми, шпигунське ПЗ, криптомайнери, тощо: 20%

Щодо розподілу по країнах програм-вимагачів у 2024 році, то він показаних на рис. 1.7.

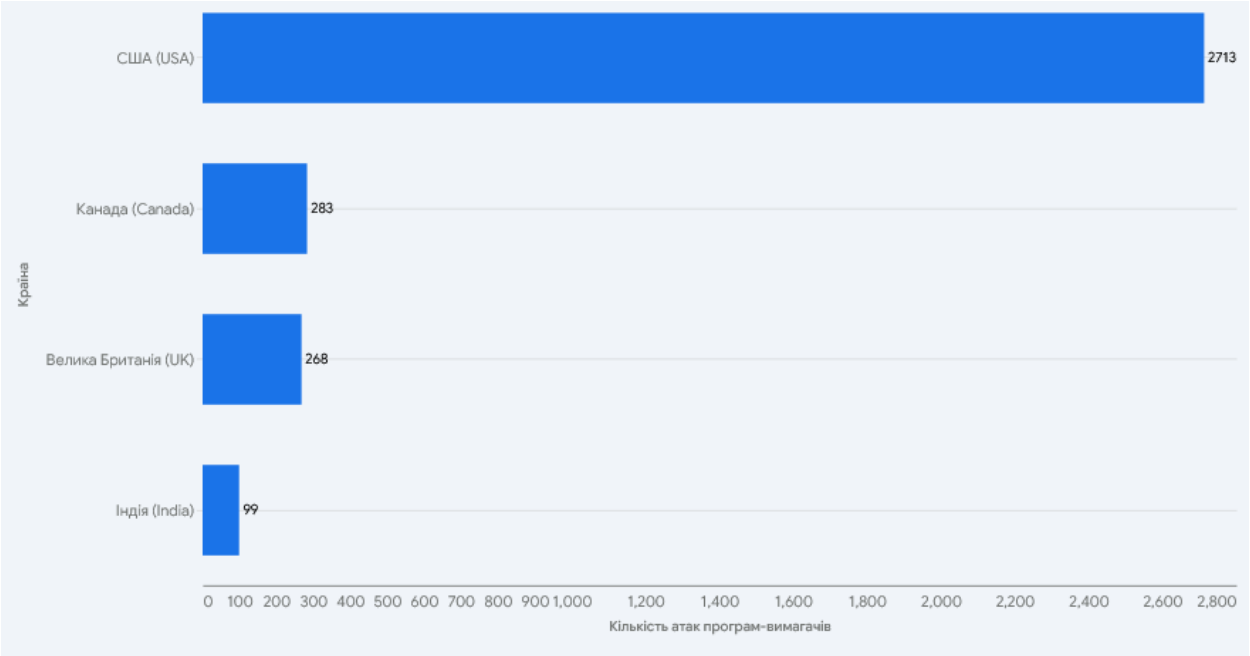


Рис. 1.7. Найбільш атаковані країни (програми-вимагачі) за 2024 рік

Цей та численні інші приклади свідчать, що традиційні превентивні методи, такі як управління ідентифікацією, списки контролю доступу (ACL) та шифрування даних, хоч і необхідні, є недостатніми для ефективного виявлення або запобігання складним кібератакам. Згідно з даними OpenSOC [11], у 60% випадків порушення безпеки дані викрадаються протягом годин, а 54% порушень залишаються невиявленими протягом місяців. Крім того, недавнє опитування [12] показало, що 87% компаній висловили занепокоєння або дуже сильне занепокоєння щодо конфіденційності даних у хмарі.

Незважаючи на наявність численних систем безпеки, кібератакуючі щодня успішно компрометують мережі організацій різного розміру. За даними іншого опитування [13], основними загрозами безпеки є:

1) Несанкціонований доступ (63%),

2) Викрадення облікових записів (61%),

3) Шкідливі внутрішні атаки (43%).

Це доводить, що програмні сервіси потребують ефективних методів виявлення атак у поєднанні з надійними превентивними механізмами для забезпечення всеосяжної безпеки. Це формує основну мотивацію для даного дисертаційного дослідження, зосередженого на пом'якшенні наслідків шкідливих внутрішніх атак.

1.3.1. Фокус на внутрішніх атаках у розподілених середовищах

Дана робота акцентує увагу на пом'якшенні інсайдерських атак, спрямованих на розголошення даних, що зберігаються у розподілених обчислювальних середовищах. Внутрішні атаки набули значного поширення і вважаються одними з найскладніших для виявлення [16]. У науковій літературі існує обмежена кількість загальних рішень для боротьби з ними [17], що є фактором, який стримує клієнтські компанії від широкого використання хмарних рішень.

Незважаючи на проголошену важливість конфіденційності та безпеки у світі великих даних, більшість розробок зосереджується лише на використанні сервісів великих даних для підвищення безпеки в інших галузях, а не на захисті самої інфраструктури. Наскільки відомо, відсутнє надійне рішення для виявлення або запобігання внутрішнім загрозам усередині інфраструктур великих даних. Наприклад, механізми безпеки популярних систем, таких як Hadoop та Spark, включають сторонні програми (наприклад, Kerberos), ACL, моніторинг журналів та певною мірою шифрування даних. Однак для внутрішнього користувача, особливо зрадника, обійти ці механізми є відносно простим завданням.

Вирішення проблеми інсайдерських атак у системах великих даних є критично важливим з чотирьох ключових причин:

а) Зловмисник, що діє зсередини організації-постачальника, має можливість обійти більшість існуючих протоколів безпеки.

б) Чутливість клієнтських даних, що зберігаються в системі, постійно зростає.

в) Успішна вразливість або атака компрометує гарантії конфіденційності, цілісності та доступності (CIA triad), надані користувачеві.

г) У спільноті великих даних відсутній консенсус щодо чітко визначених стандартів безпеки.

1.3.2. Типологія інсайдерських атак та необхідність негайного виявлення

Інсайдерські атаки є складовою ширшої категорії загроз, що включають мережеві вторгнення, переповнення буфера, атаки, специфічні для протоколів (наприклад, "людина посередині"), трояни тощо. Термін інсайдерські (внутрішні) атаки (insider threats) описує атаки, ініційовані легітимними співробітниками організації, які мають доступ до системи та знання про її архітектуру та функціонування. Зазвичай вони пов'язані з несанкціонованим викраденням даних (наприклад, за допомогою USB-накопичувачів) або маскуванням під іншого співробітника.

Іншою формою внутрішньої атаки у великих даних є деградація даних, коли внутрішній користувач модифікує дані клієнта. Побічним ефектом таких атак також може бути погіршення інфраструктури з часом. Ці атаки є складними для виявлення та ще складнішими для запобігання. В умовах зростання популярності таких концепцій, як диференційна конфіденційність, постійна втрата даних стає найбільшою проблемою, оскільки скомпрометовані дані часто можуть бути незворотними. Отже, рішення для великих даних повинні бути здатними до негайного виявлення атаки в момент її здійснення.

У цьому контексті, ми вважаємо, що платформам великих даних, ймовірно, не потрібні абсолютно нові алгоритми безпеки, але необхідне застосування існуючих методів безпеки в нових комбінаціях із перефокусуванням на захист даних користувачів та протидію внутрішнім атакам. Основний фокус даної дисертації полягає у ідентифікації,

модифікації та ретельному тестуванні таких методів відповідно до потреб платформи.

1.3.3. Статичний та динамічний аналіз як методологія дослідження

Мета рішень безпеки — ідентифікувати зловмисні атаки до того, як вони скомпрометують цілісність даних. Рішення для внутрішніх атак можуть бути реалізовані статично (на етапі компіляції) та/або динамічно (під час виконання).

Наявність механізмів безпеки на етапі компіляції мінімізує простір можливих внутрішніх атак під час виконання, що є критично важливим для продуктивно-орієнтованих систем, таких як додатки Великих Даних, оскільки вони не можуть собі дозволити значні витрати ресурсів під час виконання. Аналіз програм на етапі компіляції є статичним методом отримання інформації, корисної для різних цілей (наприклад, оптимізація паралелізму, управління ресурсами). У галузі безпеки прикладами є символічне виконання та контроль цілісності потоку управління.

Проте, аналіз на етапі компіляції з метою підвищення безпеки у великих даних раніше не проводився. Відповідно, у цій роботі пропонується кілька методів аналізу на етапі компіляції для виявлення атак. Ці методи адаптовані для перевірки потоку управління завдання, запланованого для виконання в системі великих даних.

У типовому кластері великих даних, коли користувач надсилає запит, головний вузол (master) створює завдання і планує його виконання на робочих вузлах (slave), які зберігають необхідні дані. Під час планування завдання на робочому вузлі, методи статичного аналізу можуть бути застосовані до пов'язаного скомпільованого бінарного файлу або байт-коду для виявлення вразливостей та помилок.

Запропоновані методи компіляції, аналізуючи скомпільовані програми на рівні мови асемблера, вирішують проблеми безпеки даних. Ці методи статичного аналізу для виявлення вторгнень можуть допомогти пом'якшити

наслідки вразливостей, спричинених неправильно розміщеними переходами, неініціалізованими аргументами, нульовими чи висячими вказівниками тощо.

Методи статичного аналізу мають обмеження. Вразливості, пов'язані з переповненням буфера, спільними бібліотеками та динамічним зв'язуванням, можуть зберігатися. Атаки, пов'язані з порушенням пам'яті, можуть бути виявлені та запобігнені під час виконання за допомогою адресних санітайзерів [10]. Тим не менш, обробка неправильних викликів, спричинених помилками програмістів, залишається складною. Крім того, внутрішні користувачі та маскувальники у розподілених середовищах не можуть бути повністю усунені лише статичними методами [11]. Статичний аналіз також не спрацює, якщо конфігурація робочого вузла змінена внутрішнім користувачем або якщо недобросовісний робочий вузол навмисно маскує інформацію.

Враховуючи розподілену природу платформ великих даних та їхню вимогу до узгодженості даних (через реплікацію), динамічний аналіз процесів може використовуватися для виявлення атак під час виконання та запобігання негативним наслідкам (наприклад, втраті даних). Відповідно, пропонуються більш суворі методи виявлення та запобігання вторгненням, які аналізують шаблони доступу до пам'яті та інші системні властивості процесів.

1.3.4. Аналіз шаблонів доступу до пам'яті та прогнозування атак за допомогою глибокого навчання

Шаблон доступу до пам'яті процесу корелює з адресами, на які йдуть посилання, та кількістю використовуваних сторінок. Ці шаблони, як правило, залишаються незмінними для певних даних, що використовуються тим самим додатком на стандартизованому апаратному забезпеченні. Отже, такі шаблони можуть контролюватися та керуватися системними модулями пам'яті. Шаблон доступу до пам'яті має кілька характеристик, включаючи спільні сторінки, приватні сторінки, резидентний набір тощо.

Оскільки сирі значення характеристик пам'яті можуть бути високодисперсними, у цій роботі використовується метод головних компонент (PCA) для встановлення регресії між характеристиками пам'яті. За допомогою PCA сирі шаблони доступу до пам'яті можуть бути представлені як T2-розподіл шляхом обчислення відстані від центру перекладеного простору. Далі можна застосовувати методи статистичного аналізу, такі як ANOVA та Tukey, для аналізу та порівняння дисперсії цих розподілів.

Сліпа довіра до постачальників платформ великих даних є необхідною умовою для кінцевого користувача, який зберігає дані у хмарі. Ця довіра ґрунтується на припущенні, що платформи ніколи не будуть скомпрометовані. У випадку несподіваних інцидентів, система покладається на методи виявлення вторгнень [22].

У цій роботі ми також прагнемо розробити методику для прогнозування можливості атаки. Мотивація прогнозування полягає в обмеженні впливу атак та скороченні часу їхнього виявлення. Прогнозування атак передбачає аналіз поведінкових та системних симптомів, таких як навмисні маркери, значущі помилки, підготовчі дії, корельовані шаблони використання, вербальна поведінка та риси особистості [15].

Глибоке навчання — це галузь машинного навчання, що моделює абстракції високого рівня. Нейронні мережі на основі глибокого навчання є ефективними інструментами у прогностичній аналітиці, оскільки вони можуть моделювати дані з нелінійними характеристиками, які часто проявляються у додатках великих даних. Отже, використання глибоких нейронних мереж для прогнозування атак у системах великих даних є перспективним напрямком.

Довгострокова пам'ять (LSTM) є популярною рекурентною нейронною мережею, яка доводить свою ефективність для прогнозування часових рядів та послідовностей. У даній роботі мережі LSTM використовуються для прогнозування атак усередині кластера великих даних.

У сфері великих даних принцип переміщення обчислень до місця розташування даних вважається більш ефективним, ніж традиційний підхід переміщення даних для обчислень. Основні характеристики інфраструктур великих даних включають швидку обробку, високу масштабованість, високу доступність та стійкість до збоїв.

Доступність та стійкість систем великих даних базуються на розумній реплікації даних. Це забезпечує паралельне виконання однієї програми на декількох локаціях у стилі SPMD (Single Program, Multiple Data). Коли програма планується для виконання на кластері, вона функціонує як окремий процес на кожному вузлі даних, що містить копію даних програми. Реплікація даних також може бути використана як механізм забезпечення безпеки.

Безпека для обчислювальної системи може бути реалізована на апаратному та програмному рівнях. Безпека, забезпечена на апаратному рівні, надає ізоляцію та стійкість до втручань для чутливих даних (наприклад, криптографічних ключів). Виходячи з переваг ізоляції на апаратному рівні, ми пропонуємо делегувати безпеку спеціалізованому апаратному забезпеченню, такому як Trusted Platform Module (TPM) [19] та чіпи Intel's Trusted Execution (TXT) [20], розташовані на вузлах кластера.

Така інфраструктура надасть наступні переваги:

а) виконання аналізу безпеки віддалено;

б) зменшення навантаження на основний процесор шляхом делегування завдань безпеки;

в) зменшення вартості передачі даних при забезпеченні безпеки.

Висновки до розділу

У першому розділі здійснено аналіз предметної області безпеки розподілених обчислювальних систем і середовищ обробки великих даних. Розглянуто сучасні архітектури хмарних платформ, парадигму розподіленої

обробки, а також основні напрями еволюції систем зберігання та аналізу даних. Виявлено, що існуючі виклики безпеки мають системний характер та обумовлені як технічними недоліками архітектури, так і людським фактором. Зокрема, доведено, що інсайдерські атаки становлять одну з найскладніших для виявлення загроз, оскільки здійснюються з легітимними правами доступу. Проаналізовано типологію інсайдерських атак у контексті великих даних, виділено їх поведінкові характеристики, а також сформульовано основні напрями підвищення рівня довіри та контролю в розподілених системах. Показано, що статичні методи контролю доступу не здатні забезпечити ефективний захист у динамічних обчислювальних середовищах, а тому потребують доповнення методами поведінкового моніторингу та динамічного аналізу.

Обґрунтовано доцільність використання глибокого навчання для аналізу шаблонів доступу до пам’яті, виявлення аномалій у потоках операцій та прогнозування потенційних внутрішніх вторгнень.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ТА ОПИС МЕТОДІВ ВИЯВЛЕННЯ ІНСАЙДЕРСЬКИХ АТАК У РОЗПОДІЛЕНИХ СЕРЕДОВИЩАХ

2.1. Огляд дослідницьких напрямків та фреймворків великих даних

Дане магістерське дослідження інтегроване в низку ключових наукових напрямків, які охоплюють як фундаментальні аспекти комп'ютерної безпеки, так і специфічні виклики розподілених обчислювальних середовищ. Ці напрямки включають:

а) Обробка вразливостей, індукованих помилками програмування (programmer errors).

б) Розробка методів захисту від інсайдерських атак (insider threats).

в) Методи виявлення вторгнень (Intrusion Detection) у розподілених системах.

г) Відстеження поведінки додатків для забезпечення безпеки в розподілених середовищах.

д) Безпека на платформах великих даних, зокрема в екосистемі Hadoop.

е) Аналіз графів потоку управління (Control Flow Graphs) для верифікації процесів.

є) Моніторинг шаблонів доступу до пам'яті для виявлення аномалій у поведінці процесів.

Перед поглибленим розглядом цих дослідницьких тем, необхідним є вступний огляд фреймворків великих даних для забезпечення контекстуального розуміння необхідності та потенційного впливу вищезазначених досліджень.

Сучасна IT-індустрія експлуатує широку екосистему фреймворків для обробки великих даних, переважно представлену програмним забезпеченням з відкритим вихідним кодом під ліцензіями Apache Software Foundation. До цієї категорії належать: Hadoop, Spark, Flume, Kafka, HBase, Storm, Solr, Tez, Sqoop, ZooKeeper тощо. Зазначені фреймворки функціонують поверх

розподілених обчислювальних середовищ (РОС), що забезпечують горизонтальне масштабування та відмовостійкість.

У межах даного дослідження термін «Hadoop» використовується як узагальнений синонім для позначення архітектурної платформи великих даних, яка базується на РОС. Відповідно, критично важливим є розуміння її структурної організації.

Hadoop реалізує архітектурну парадигму «майстер-робочий» (master-slave), що оптимізує витрати на обробку даних, управління винятками (fault tolerance) та відновлення після збоїв або атак. Її ключовими компонентами є:

1. HDFS (Hadoop Distributed File System) — розподілена файлова система, яка забезпечує відмовостійке сховище для масштабних наборів даних з реплікацією блоків і горизонтальним масштабуванням.

2. MapReduce — модель програмування та виконавчий рушій для паралельної обробки даних, розміщених у HDFS, що реалізує два фазові етапи: map та reduce, з прозорою підтримкою перерозподілу завдань у разі вузлових збоїв.

Таблиця 2.1.

Компоненти Hadoop

Компонент	**Роль**	**Функція**
Namenode	Майстер зберігання (HDFS)	Керує всіма метаданими файлової системи (інформація про структуру, розташування блоків).
Datanodes	Робочі вузли Namenode	Використовуються для зберігання фактичних даних у блоках.
JobTracker	Майстер обробки (MapReduce)	Керує завданнями (jobs), їхнім плануванням та правами доступу на рівні кластера і користувачів.
TaskTrackers	Робочі вузли JobTracker	Відповідальні за виконання призначених підзадач ("map" або "reduce").

Новіші версії екосистеми Hadoop також включають YARN (Yet Another Resource Negotiator) — додатковий ключовий компонент, призначений для

управління ресурсами кластера. Усі перелічені компоненти (Namenode, Datanode, JobTracker, TaskTracker та компоненти YARN) є демонами, які працюють у кластері машин. Сукупність цих машин та наданих ними сервісів становить кластер Hadoop.

Основний цикл роботи Hadoop складається з чотирьох послідовних кроків:

1. Запис HDFS (HDFS write) - завантаження вихідних даних у кластер.

2. MapReduce - паралельний аналіз даних.

3. Запис HDFS (HDFS write) - зберігання результатів обробки в кластері.

4. Читання HDFS (HDFS read) - вилучення кінцевих результатів з кластера.

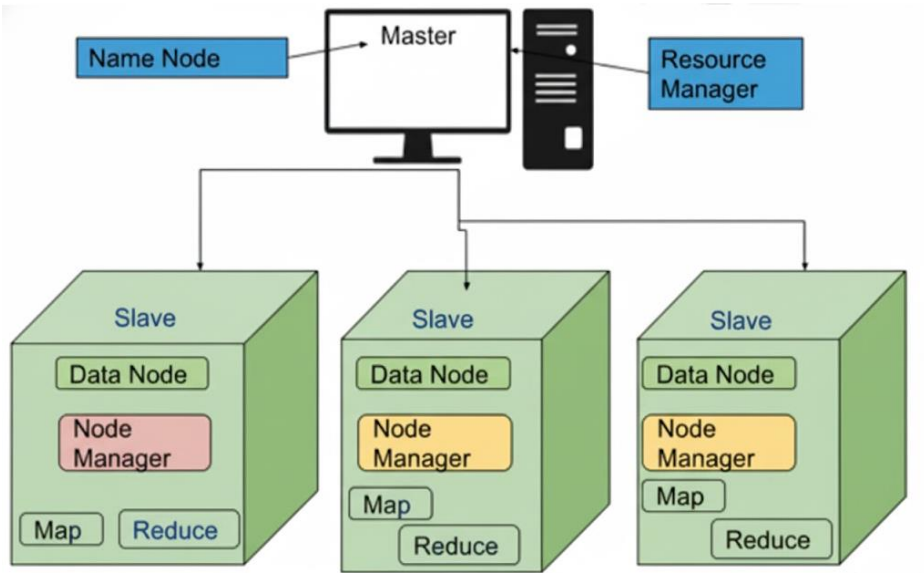


Рис. 2.1. Верхній рівень архітектури Hadoop

На цій схемі (рис. 2.1) відображені три основні шари Hadoop, які спільно забезпечують обробку великих даних:

1. Рівень зберігання (HDFS) - представлений Namenode (Master) та Datanodes (Slaves).

2. Рівень управління ресурсами (YARN) - представлений ResourceManager (Master) та NodeManagers (Slaves).

3. Рівень обробки (MapReduce) - програми, які використовують ресурси, надані YARN, для виконання обчислень на даних, що зберігаються в HDFS.

Ця архітектура дозволяє розподіляти великі обчислювальні завдання на кластері звичайних комп'ютерів.

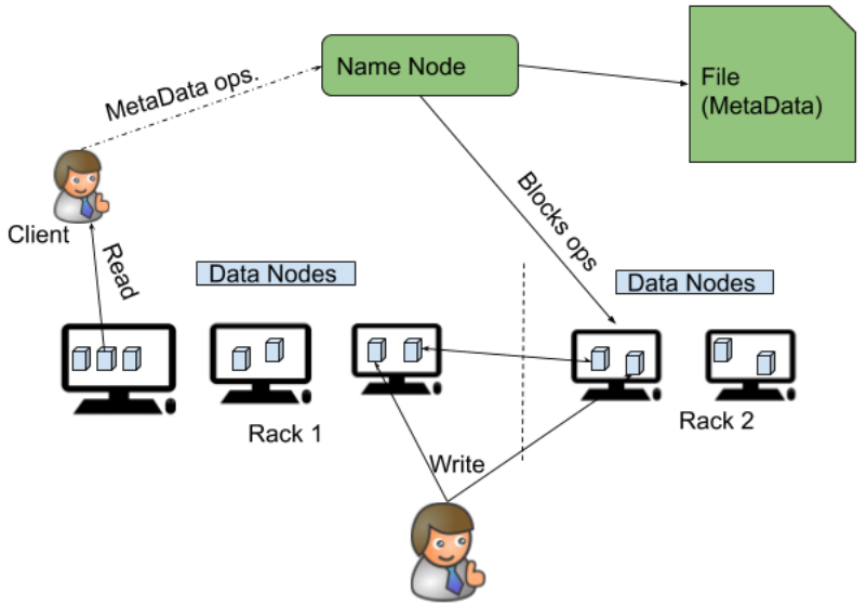


Рис. 2.2. Спрощена архітектура Hadoop Distributed File System (HDFS)

Рис. 2.2 ілюструє спрощену архітектуру Hadoop Distributed File System (HDFS), яка є основним компонентом Hadoop для зберігання великих даних. Ця архітектура заснована на моделі "майстер-робочий" (master-slave) і демонструє механізми читання та запису даних клієнтами.

Ключові компоненти архітектури:

- Client (Клієнт) є точкою входу для взаємодії з HDFS, ініціюючи операції читання та запису файлів.

- Name Node (Namenode) - головний вузол (master) у HDFS. Він зберігає та керує метаданими файлової системи. Метадані включають імена файлів та директорій, права доступу, розташування кожного блоку даних на Datanode (де фізично зберігається кожна частина файлу).

- Data Nodes (Datanodes) - робочі вузли (slaves) у HDFS. Фізично зберігають блоки даних файлів. Кожен файл у HDFS розбивається на блоки (зазвичай 128 МБ або 256 МБ), і ці блоки розподіляються між Datanodes. Namenode керує Datanodes, вказуючи, які блоки вони повинні зберігати (операції з блоками - "Blocks ops.").

- Racks. На рис. 2.2 показано дві групи Datanodes, позначені як "Rack 1" та "Rack 2". Це ілюструє фізичне розташування вузлів у центрі обробки даних. HDFS враховує топологію стійок для стратегії реплікації даних. Зазвичай, блоки файлів реплікуються на різні Datanodes, розташовані в різних стійках. Це забезпечує відмовостійкість у разі відмови цілої стійки або мережевого комутатора, пов'язаного з нею.

Процес читання даних клієнтом:

1. Клієнт відправляє запит на читання файлу до Namenode.

2. Namenode, використовуючи свої метадані, визначає, з яких Datanodes складається файл і де розташовані його блоки. Він повертає цю інформацію клієнту.

3. Клієнт напряму звертається до відповідних Datanodes і починає читати блоки даних (позначено стрілкою "Read" від Client до Datanodes).

Ця архітектура дозволяє HDFS ефективно зберігати величезні обсяги даних, забезпечуючи при цьому високу пропускну здатність (завдяки прямому доступу до Datanodes) та відмовостійкість (завдяки реплікації даних та врахуванню топології стійок). Namenode є критично важливою точкою відмови в цій спрощеній моделі, тому в реальних розгортаннях HDFS використовуються механізми високої доступності для Namenode.

2.2. Аналіз векторів атаки: інсайдерські загрози та програмні вразливості

У цій роботі аналізується обмежена підмножина загроз, що безпосередньо знижують конфіденційність і цілісність даних у розподілених

обчислювальних середовищах класу big-data. Основний вектор дослідження сформовано внутрішніми (інсайдерськими) атаками, а також уразливостями, експлуатованими через архітектурні недоліки (loopholes) та помилки програмування інфраструктурного коду.

2.2.1. Інсайдерські атаки

Незважаючи на тривалу еволюцію захисних механізмів, сучасні обчислювальні системи зберігають залишкову вразливість до кібератак, частота й економічний вплив яких демонструють стійку позитивну тенденцію. Будь-яка програмна атака, за визначенням, реалізується через зловмисне виконання інструкцій на цільовій обчислювальній платформі. Захист може бути локалізований на двох комплементарних рівнях:

a) Програмний рівень – забезпечує тонке розуміння семантики програми, але сам є потенційним об’єктом компрометації;

b) Апаратний рівень – гарантує сильнішу ізоляцію аналізаторів і цільових процесів, проте обмежує доступ до високорівневого контексту виконання.

Найвищу складність для виявлення та подолання мають атаки з інтенційним походженням, ініційовані суб’єктами з глибокою експертизою щодо топології, політик безпеки й внутрішніх інтерфейсів системи — т.зв. інсайдерські загрози.

На підставі систематичного огляду літератури сформульовано чотири ключові наукові питання, вирішення яких є необхідною умовою ефективної протидії інсайдерським атакам:

a) Хто є потенційним інсайдером (модель зловмисника)?

b) Що саме піддається впливу (активи та критичні процеси)?

c) Як виявляти аномальну поведінку в режимі реального часу?

d) Як запобігати експлуатації на етапах проєктування та виконання?

На рис. 2.3 представлено онтологію інсайдерської загрози у вигляді трійки ⟨суб’єкт, об’єкт, метод⟩ і показано, як чотири ключові запитання

(«Хто?», «Що?», «Як виявити?», «Як запобігти?») задають взаємозалежності між сутностями.

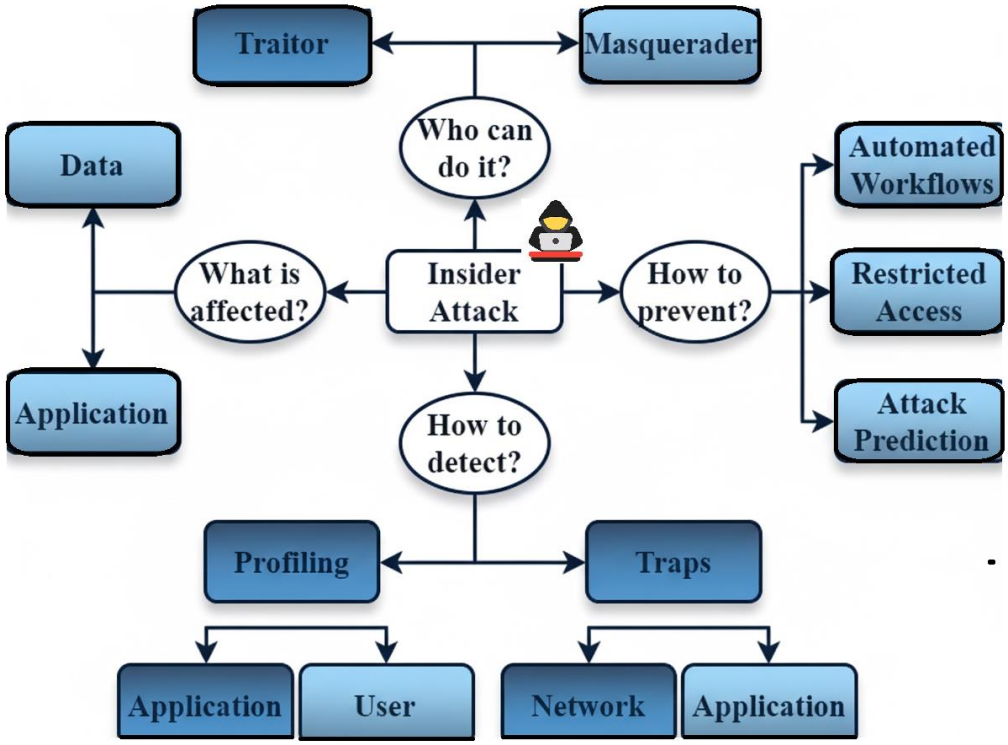


Рис. 2.3. Представлення сутностей та взаємозв'язків в інсайдерських атаках

Суб’єкти атаки

a) Malicious Insiders (зрадники) – легітимні учасники системи, що зловживають наданими правами доступу.

b) Masqueraders – зовнішні зловмисники, які імперсонізують легітимних користувачів, експлуатуючи викрадені облікові дані.

Об’єкти впливу – виконувані програми та зберігаємі дані: порушення цілісності коду призводить до некоректного функціонування сервісів, тоді як корупція даних руйнує інформаційну складову додатків.

Методи виявлення:

- Profiling – статистичне або машинно-навчальне моделювання нормальної поведінки на рівні користувача (споживання ресурсів, психометричні характеристики) та на рівні програми (послідовності системних викликів, графи потоку керування).

- Honeypots – програмні або мережеві пастки, що провокують інсайдера до активних дій, фіксуючи при цьому атрибутивні сліди.

Критичним обмеженням згаданих методів є ймовірна втрата чутливих даних під час фази збору доказів. Тому виявлення має доповнюватися превентивними механізмами: багатофакторне управління ідентифікацією, дискреційні та ролеві списки контролю доступу, а також криптографічне шифрування даних у стані спокою та в пересуванні.

2.2.2. Складність виявлення атак в екосистемах хмари та великих даних

Внутрішні атаки є значною загрозою в будь-якій галузі через їхню складність прогнозування та виявлення. Відповідно, організації повинні докладати зусиль для захисту своїх систем. Прогностичні моделі для поведінки користувача/програми/мережі, засновані на безперервному моніторингу, є загальноприйнятим рішенням. Однак їхня надійність не є абсолютною, і складність виявлення зростає пропорційно складності базової системи.

Сучасні досягнення призвели до широкого впровадження хмарних обчислень та великих даних, які є надзвичайно складними за своєю архітектурою. У хмарних обчисленнях внутрішні атаки можуть бути здійснені через маніпуляцію клієнтськими службами, а скомпрометовані дані можуть створити можливості для соціальної інженерії та каскадних атак. Запропоновані ідеї для захисту хмарних інфраструктур включають безпеку як сервіс (Security-as-a-Service) та створення закритих (приватних) хмар.

На рисунку 2.4 представлена діаграма, що відображає ключові переваги впровадження моделі безпеки як послуги (SECaaS):

1. Agility and Faster Provisioning (гнучкість та швидше надання).

SECaaS дозволяє організаціям швидко розгортати та масштабувати рішення безпеки у відповідь на мінливі потреби бізнесу, усуваючи необхідність у тривалому придбанні та налаштуванні фізичного обладнання.

Рис. 2.4. Концепти моделі Security-as-a-Service

2. Enhanced Visibility (покращена видимість).

Надання послуг із централізованої хмарної платформи часто забезпечує єдину, консолідовану панель моніторингу для всіх аспектів безпеки, покращуючи огляд загроз і стану захисту.

3. Less Vulnerability (менша вразливість).

Постачальники SECaaS, як правило, забезпечують автоматичні та своєчасні оновлення, виправлення (patches) та конфігурації своїх систем, що допомагає мінімізувати вікно вразливості, пов'язане з застарілим ПЗ.

4. Free Up Resources (звільнення ресурсів)

Передача функцій безпеки зовнішньому постачальнику дозволяє внутрішнім IT-командам перенаправити свої ресурси на основні бізнес-завдання, а не на управління безпекою.

5. Cost Savings (економія витрат).

Перехід від капітальних витрат (CAPEX) на купівлю обладнання та ліцензій до операційних витрат (OPEX) на підписку. Це часто призводить до зниження загальної вартості володіння (TCO), оскільки усуваються витрати на обслуговування, енергію та оновлення.

6. Access to Security Professionals (доступ до професіоналів з безпеки).

Клієнти отримують миттєвий доступ до висококваліфікованих експертів з кібербезпеки, моніторингу 24/7 та передових методів захисту, що зазвичай недоступні або надто дорогі для внутрішньої підтримки.

На відміну від хмарних обчислень, які акцентують увагу на обчисленнях "на льоту", великі дані зосереджені на організації та управлінні масивними наборами даних.

Виявлення та запобігання внутрішнім атакам у фреймворках великих даних залишається недостатньо дослідженою областю.

Ідеальним, хоча й нереалістичним, рішенням для протидії внутрішнім загрозам є повна автоматизація кожного аспекту системи для усунення будь-якого людського втручання. Особливо у системах великих даних, які мають сервісний стек постачальника та клієнта, постачальники хмарних послуг (наприклад, Amazon, Google) мінімізують потенціал для внутрішніх атак через:

а) автоматизацію більшості системних аспектів,

б) вимогу до клієнтів дотримуватися аналогічних процедур.

2.2.3. Вразливості, спричинені помилками програмування

Помилки програмування (programmer errors) є домінантним джерелом невизначених станів у системі, що трансформуються у критичні атакові поверхні. Вони породжують непередбачувані взаємодії між компонентами й унаслідок цього дозволяють зловмиснику ескалювати привілеї до рівня суперкористувача. Найчастіші канали експлуатації включають:

- переповнення буфера (buffer overflow);

- умови гонки (race condition);

- неконтрольоване звернення до системних файлів (unauthorized file access).

Пом’якшення цих вразливостей традиційно зводиться до поєднання статичного/динамічного аудиту коду та принципу найменших привілеїв (least privilege); при цьому для аналізу аудиторських трейсів активно залучаються big-data платформи.

Окремий клас помилок спричиняє порушення семантики пам’яті (memory-corruption vulnerabilities). Для запобігання їх експлуатації

застосовують механізми Control Flow Integrity (CFI) і компіляційну санітизацію інструкцій звернення до пам’яті (memory sanitization). Вказані підходи демонструють високу ефективність у монолітних застосунках, однак їхнє застосування до розподілених систем ускладнене:

- багатовузловий контекст інвалідує припущення одного адресного простору;

- overhead санітайзерів зростає лінійно з кількістю вузлів, що знижує загальну пропускну здатність кластера.

Ця робота концентрується виключно на підмножині memory-corruption помилок, які:

а) не розкриваються на етапі компіляції/статичного аналізу;

б) проявляються виключно під час рантайму в розподіленому середовищі.

2.3. Виявлення вторгнень та відстеження поведінки у розподілених системах

Безпека розподілених обчислювальних систем залишається предметом інтенсивних досліджень останні два десятиліття. Наукова увага зосереджена на двох взаємопов’язаних завданнях:

1) виявленні вторгнень (Intrusion Detection, ID),

2) відстеженні системної поведінки з метою ідентифікації або прогнозування атак.

2.3.1. Виявлення вторгнень

Системи виявлення вторгнень (IDS) класифікуються за стратегією виявлення:

а) Knowledge-based (сигнатурні) – пошук відомих сигнатур у предефінованій базі загроз. Через експоненційне зростання zero-day атак

монолітна сигнатурна модель втрачає життєздатність: навіть гіпотетично повна база потребує надлінійних витрат на оновлення та он-лайн-запити.

б) Behavior-based (аномальні) – побудова статистичної або машинно-навчальної моделі нормальної поведінки користувача/застосунку/мережі та виявлення відхилень. Хоча ресурсна складність вища, метод залишається єдиним масштабованим засобом протидії невідомим загрозам.

У кластерах petabyte-шкали IDS трансформується у розподілену D-IDS із централізованим управлінням політиками та локальними сенсорами на рівнях «хост», «мережа» і «дані». Ефективність D-IDS залежить від (i) агрегації телеметрії, (ii) горизонтальної комунікації вузлів, (iii) співпраці приймачів рішень. Типова практика – використання big-data платформ (Apache Metron, Apache Spot, IBM Guardium) як інфраструктури для D-IDS. Ця робота інвертує схему: метою є вбудована IDS, що захищає саму платформу big-data, використовуючи її ж розподілені обчислювальні ресурси для побудови behavior-based D-IDS.

2.3.2. Відстеження поведінки

Існує висока потреба в інструментах, здатних діагностувати складні розподілені системи, оскільки першопричина проблеми може бути пов'язана з множинними подіями або системними компонентами.

Останні розробки у сфері розподіленого відстеження зосереджені на наданні незалежного сервісу. Magpie [11] захоплює події в розподіленій системі та використовує модельну систему для зберігання трас. Magpie — це експериментальна архітектура виявлення вторгнень (Intrusion Detection Architecture), розроблена з метою поведінкового аналізу власне самої платформи великих даних, на відміну від традиційних рішень, які використовують Big Data інструменти лише як інфраструктуру для аналізу зовнішніх подій. Magpie інтегрує розподілені сенсори безпосередньо у компоненти кластера (HDFS, YARN, MapReduce, Spark тощо) і будує онлайн-модель нормальної поведінки кожного вузла на основі:

- локальних виконавчих трейсів (системні виклики, контрольні точки JVM, інструкції пам’яті);

- потокових метаданих (блокові реплікації, розміщення контейнерів, мережеві пакети між DataNode ↔ NameNode, TaskTracker ↔ JobTracker);

- семантичних подій (ACL-перевірки, зміни політик Ranger/Knox, журнали Oozie).

Xtrace надає комплексний огляд шляхом реконструкції поведінки сервісу через поширення метаданих. Хоча цей підхід схожий на запропонований у даній роботі причинно-наслідковий аналіз на рівні завдань, Xtrace акцентує увагу на аналізі на мережевому рівні.

На рисунку 2.5 зображено відправника S, який встановлює сервер звітів R як пункт призначення для звітів. Домени адміністрування (ADs) A та B надсилають покажчики звітів (pointer reports) до R, і пізніше або клієнт, або сам R отримує ці звіти.

Особливий випадок виникає, коли користувач X-Trace знаходиться в тому ж AD, що й пристрої, які генерують звіти, наприклад, мережеві оператори, що проводять внутрішнє усунення несправностей. Метадані X-Trace додаються на точках входу (ingresspoints) AD. Мережеві оператори звертаються безпосередньо до локальних баз даних звітів, і немає потреби використовувати поле призначення (destination field) у метаданих.

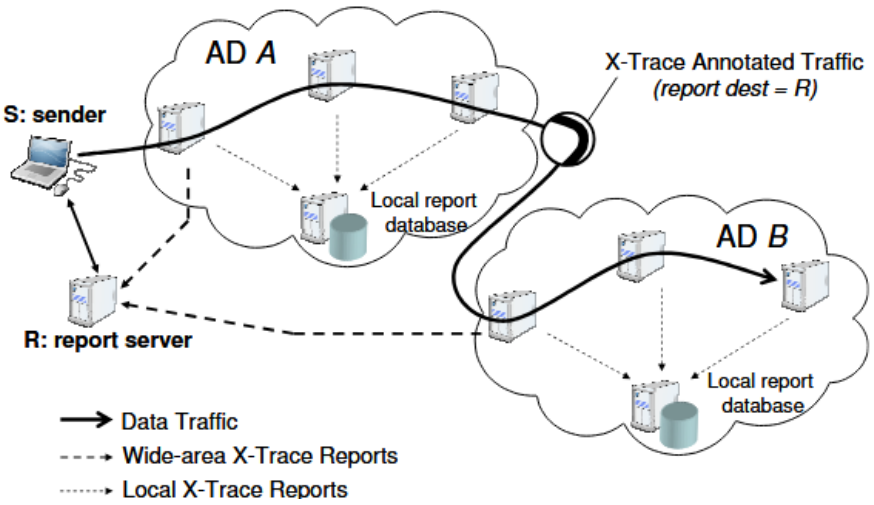


Рис. 2.5. Приклад звітування в широкій мережі

Клієнт вбудовує метадані X-Trace у повідомлення, встановлюючи пункт призначення звітів на R. Різні Інтернет-провайдери (ISP) збирають звіти локально та надсилають покажчики до R, щоб клієнт міг пізніше запитати детальні звіти.

Недолік Xtrace та Retro, що вони тісно інтегровані в систему і вимагають від користувача модифікації вихідного коду.

HTrace - проект Apache Incubator для розподіленого відстеження, що вимагає додавання інструментації до програми.

Pivot Trace - динамічна служба моніторингу причинно-наслідкових зв'язків, що надає відношення "сталося раніше" (happened-before) між дискретними подіями.

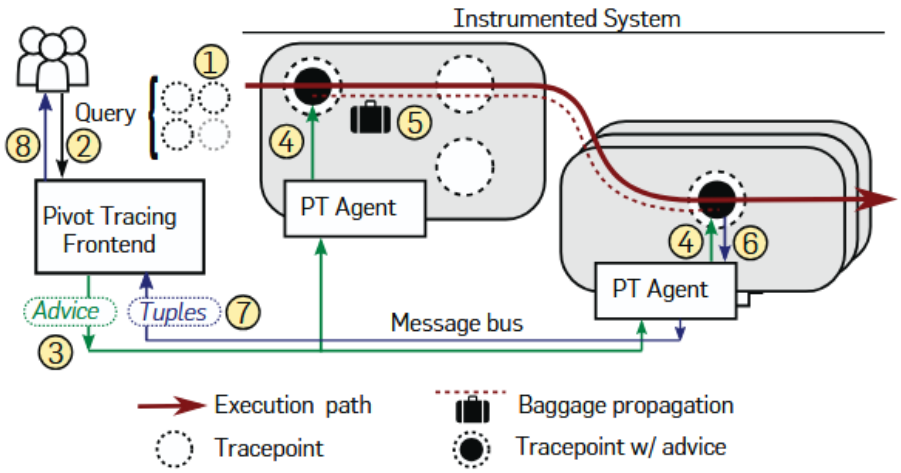


Рис. 2.6. Огляд Pivot Tracing

На рисунку 2.6 представлено високорівневий огляд того, як Pivot Tracing уможливлює такі запити, як Q2. Повна підтримка Pivot Tracing у системі вимагає двох основних механізмів:

- Динамічне впровадження коду (dynamic code injection).

- Поширення причинно-наслідкових метаданих (causal metadata propagation).

Запити у Pivot Tracing посилаються на змінні, які експонуються однією або декількома точками трасування (tracepoints) — це місця в системі, куди

Pivot Tracing може вставити інструментарій. Визначення точок трасування не є частиною системного коду, а є, скоріше, інструкціями щодо того, де і як змінити систему для отримання експортованих ідентифікаторів. Точки трасування в Pivot Tracing схожі на зрізи (pointcuts) в аспектно-орієнтованому програмуванні[14] і можуть посилатися на довільні комбінації інтерфейс/сигнатура методу.

Точки трасування визначаються особою, яка має знання про систему — можливо, розробником або експертом-оператором, і вони визначають словник для запитів. Вони можуть бути визначені та встановлені в будь-який момент часу, а також можуть поширюватися та передаватися.

У даній роботі пропонується відстежувати стек системних викликів (system calls) та бібліотечних викликів (library calls). Дослідження [22] підтвердили, що короткі послідовності системних викликів є ефективним дискримінатором між нормальною та аномальною роботою. Раніше для аналізу послідовностей системних викликів використовувалися такі моделі, як баєсівська класифікація, приховані марковські моделі (HMM) та алгебра процесів [23]. У цій роботі метадані системних викликів використовуються для побудови профілю поведінки процесу. Це досягається шляхом екстракції інформації про системні виклики, здійснені під час виконання, безпосередньо зі стеку викликів. Інформація, пов'язана з бібліотечними викликами, також включається в профілі поведінки, оскільки фреймворки великих даних інтенсивно використовують бібліотечні виклики, які можуть бути повністю враховані.

2.4. Архітектурні недоліки та внутрішні вектори атаки для платформ великих даних

Проблематика безпеки у екосистемах великих даних залишається предметом активних наукових дискусій та промислових ініціатив. Проте сучасні розробки переважно спрямовані на експлуатацію потенціалу big-data

для захисту зовнішніх інформаційних систем, тоді як внутрішня безпека самих платформ великих даних залишається недостатньо структурованою. Ідеальна архітектура передбачає інтеграцію безпеки як first-class citizen у базовий дизайн кластера, однак критичні вимоги big-data-додатків — обробка в реальному часі, стійкість до збоїв (fault-tolerance) та 24/7 доступність — накладають жорсткі обмеження на складність і обчислювальний overhead захисних механізмів.

Усі наявні рішення, інтегровані у фреймворки на кшталт Hadoop, є програмно-центричними та орієнтовані переважно на зовнішні загрози. Типова модель безпеки Hadoop базується на трьох елементах:

- Багаторівнева автентифікація (Kerberos — обов'язковий компонент).
- Журналювання та аналіз логів (log4j, Solr/Elasticsearch).
- Опційне шифрування (TLS для RPC/data-at-rest, AES-NI).

Переваги таких механізмів — висока продуктивність і простота експлуатації, проте вони не розраховані на протидію інсайдерським атакам. Навіть «secure mode» кластера зберігає централізовану природу сервісів (Kerberos KDC, Ranger Admin, ZooKeeper), що призводить до:

- Затримок оцінки ризиків через синхронізацію станів між централізованими службами.
- Проблеми «довіреної основи» (TCB): усі сервіси безпеки виконуються у тій самій JVM/OS, що й користувацький код.
- Високого CPU-overhead: ≥ 2 ядра під AES-NI для 10 Gbps-потоку.
- Нереалістичних припущень: користувачеві заборонено фізичний доступ до DataNode, що неможливо у хмарних IaaS.

Офіційний SLA Hadoop допускає лише 3 % продуктивного регресу від усіх безпекових модифікацій; саме тому шифрування data-in-use всередині кластера залишається опціональним і частковим, а запропоновані підходи «випадкового шифрування» чи «надлишкової фрагментації» не мають експериментального підтвердження прийнятного overhead.

Крім того, властивості великих даних — горизонтальна розподіленість, багаторівнева реплікація, агресивне кешування — руйнують презумпцію «дорогої точності» традиційних IDS і ускладнюють виявлення внутрішніх атак. У цій роботі демонструється неефективність існуючих механізмів шляхом реалізації двох реалістичних інсайдерських атак на повномасштабному кластері Hadoop 3.3. Нижче наведено чотири сценарії, експериментально реалізовані та проаналізовані в роботі:

1. Корупція блокової контрольної суми з подальшою реплікацією підроблених даних.

2. Ескалація привілеїв через rogue YARN-container з подальшим доступом до ключів шифрування.

3. Підміна журналів HDFS з використанням race-condition у NN-Edits.

2.5. Експериментальна демонстрація інсайдерських атак

Метадані файлової системи HDFS зберігаються на Namenode у вигляді двох компонентів: fsImage (використовується під час запуску) та EditLog (постійно оновлюється). Хоча використання вторинного Namenode (Secondary Namenode) вирішує проблему єдиної точки збою, він також допомагає асинхронно оновлювати файлову систему. Це відбувається через періодичне оновлення fsImage на вторинному Namenode за допомогою локального EditLog, після чого оновлений fsImage зливається з основним Namenode (контрольна точка).

Сценарій атаки наступний. Якщо зловмисник (внутрішній користувач з команди Hadoop ops) модифікує EditLog на вторинному Namenode, наступна контрольна точка відобразить цю зміну на основному fsImage Namenode, що може призвести до втрати даних.

Цей сценарій був реалізований на кластері Hadoop. На рисунку 2.8 наведено модель атаки.

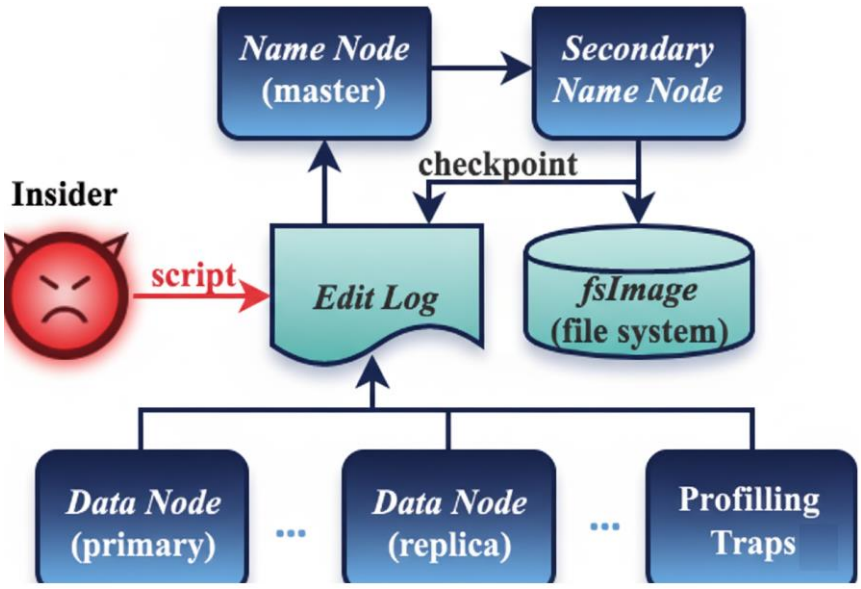


Рис. 2.7. Модель атаки для створення втрати даних

Спочатку файлова система містила 300 блоків даних. Після модифікації зловмисним скриптом, існуюча файлова система (fsImage) була повністю стерта. Ця зміна відобразилася на Namenode через 10 хвилин (на наступній контрольній точці). Це демонструє архітектурний недолік Hadoop, який легко експлуатується внутрішніми користувачами з правами системного адміністратора.

Серверні журнали Hadoop є критично важливими для управління та виявлення потенційних атак системними адміністраторами. Дані журналів часто структурують (наприклад, за допомогою Pig або Hive) та аналізують інструментами бізнес-аналітики. Продукти, як-от Flume та Kafka, широко використовуються для обробки подій у реальному часі та побудови рішень безпеки (наприклад, моніторингу DDoS-атак).

Сценарій атаки. У типовому робочому процесі запити користувачів, включаючи запити атакуючих, реєструються log4j. Системний адміністратор-зрадник може маніпулювати даними серверного журналу до того, як вони будуть передані для аналізу (наприклад, через Flume до Hcatalog). Зловмисний скрипт, запущений внутрішнім користувачем з правами системного адміністрування, модифікує дані журналу, спотворюючи

результати аналізу та створюючи неправильне уявлення про безпекову ситуацію (наприклад, приховуючи сліди DDoS-атак).

Модель демонструє, як зловмисний скрипт (як внутрішнє завдання) змінює дані журналу. Рисунок 2.8 показує, як спотворюються результати, незважаючи на нормальне функціонування служб Hadoop: частина "До" показує реальні атаки, тоді як частина "Після" — модифікований вихід.

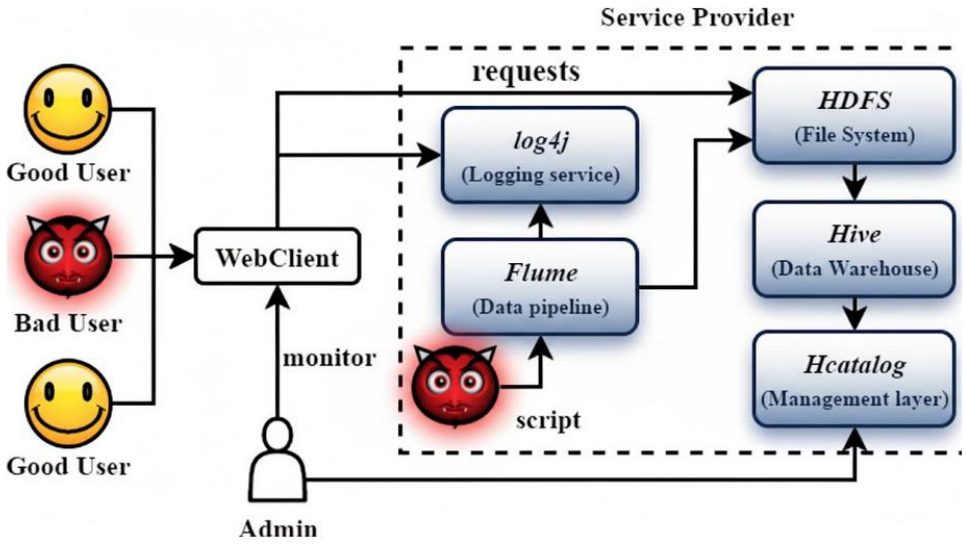
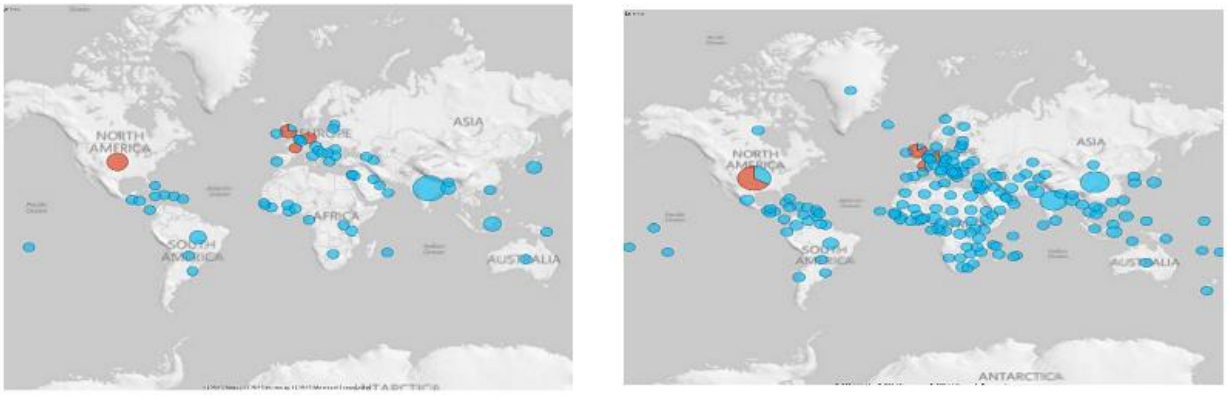


Рис. 2.8. Модель атаки деградації даних у кластері Hadoop



а) б)

Рис. 2.9. Приклад, що демонструє фактичні а) і маніпульовані б) результати

Атака розголошення даних має на меті несанкціоновану передачу даних і включає два випадки: а) використання недовіреного обладнання та б) використання довіреного програмного забезпечення (наприклад, клієнта

електронної пошти). Сценарій атаки наступний - внутрішній користувач змінює конфігурацію datanode (через файл hdfs-site.xml), додаючи нове локальне місцезнаходження до властивості каталогу даних DFS. Як наслідок, усі блоки на цьому datanode мають дві копії — одна в легітимному HDFS-каталозі, інша — у локальній папці системного адміністратора. Далі використовується скрипт для імітації передачі цих дублікатних файлів на віддалене місце призначення за допомогою електронної пошти або системних викликів, пов'язаних із USB-пристроями (симуляція через неможливість прямого підключення USB до Amazon EC2).

2.6. Аналіз потоку управління та його застосування в безпеці

Граф потоку управління (Control Flow Graph, CFG) є спрямованим графом, що являє собою програмний код, і зазвичай характеризується як розріджений (sparse) граф. CFG інкапсулює всі можливі шляхи управління в програмі, що робить його потужним інструментом для вилучення поведінки потоку управління відповідного процесу.

Вершини (Vertices) у CFG представляють рівень деталізації, наприклад, рівень інструкцій або рівень базових блоків, які не підлягають подальшому поділу. Ребра (Edges) у CFG репрезентують переходи управління і класифікуються на два типи: прямі (forward) та зворотні (backward). Прямі ребра включають інструкції розгалуження, виклики функцій, умовні та безумовні переходи. Непрямі виклики функцій та віртуальні виклики також вважаються прямими ребрами, хоча визначення їхніх пунктів призначення може бути обчислювально складним.

Зворотні ребра зазвичай представляють цикли та повернення з функцій.

Інформація, отримана з CFG, може бути використана для перевірки цілісності дублікатних процесів, що виконуються на реплікаційних вузлах системи великих даних. Перевірка подібності між логікою двох програм може бути виконана шляхом порівняння їхніх CFG на ізоморфізм. Проблема

ізоморфізму графів (Graph Isomorphism) є складною (complex), іноді класифікується як NP-повна, що робить аналіз подібності двох процесів шляхом перевірки ізоморфізму CFG часозатратним і складним.

Для зниження обчислювальної складності алгоритмів графів, CFG можуть бути зведені до дерев або підграфів перед виконанням будь-яких перевірок узгодженості або цілісності.

CFG може бути перетворений на дерево за допомогою таких методів, як обхід у глибину (Depth-First Search, DFS). Декілька деревоподібних структур, включаючи дерево домінаторів (dominator tree), мінімальне остовне дерево (Minimum Spanning Tree, MST) та мінімальна остовна арбореценція (Minimum Spanning Arborescence, MSA), можуть бути вилучені з CFG. CFG також може бути розбитий на підграфи за допомогою таких технік, як k-підграфове співпадіння та розфарбовування графів.

Припускаючи, що CFG має n вершин та m ребер, існують наступні популярні методи для зменшення та порівняння графів: на основі редагувальної відстані, на основі обходу, на основі дерев домінаторів, на основі досяжності.

У цій роботі CFG зводиться до набору мінімальних остовних арборецендій (MSA). Цей вибір обґрунтований тим, що CFG зазвичай є розрідженими графами, і, отже, розмір набору MSA є скінченним і зазвичай невеликим (менше 100). У таких контекстах алгоритм Едмондса може бути використаний для швидкого вилучення всіх MSA з орграфа.

Оскільки MSA містить усі вершини свого графа, втрати даних інструкцій програми не відбувається. Кількість ребер відрізнятиметься між CFG та його представленням MSA, залежно від зв'язності графа.

На рисунку 2.10 проілюстровано перетворення рядка коду Java на базові блоки байт-коду, CFG та, нарешті, на набір MSA. Вершини B1,B2,B3,B4 являють собою базові блоки, сформовані з байт-коду Java.

Існує алгоритм з часовою складністю O(m+nlogn) для обчислення мінімальної вартості арборецендії. Альтернативно, популярні компілятори,

такі як llvm та gcc, використовують інший підхід для перетворення CFG в MSA на основі об'єднання знаходження для цілей безпеки. Істотним недоліком використання CFG та MSA для цілей безпеки є те, що виклики динамічних бібліотек не можуть бути верифіковані.

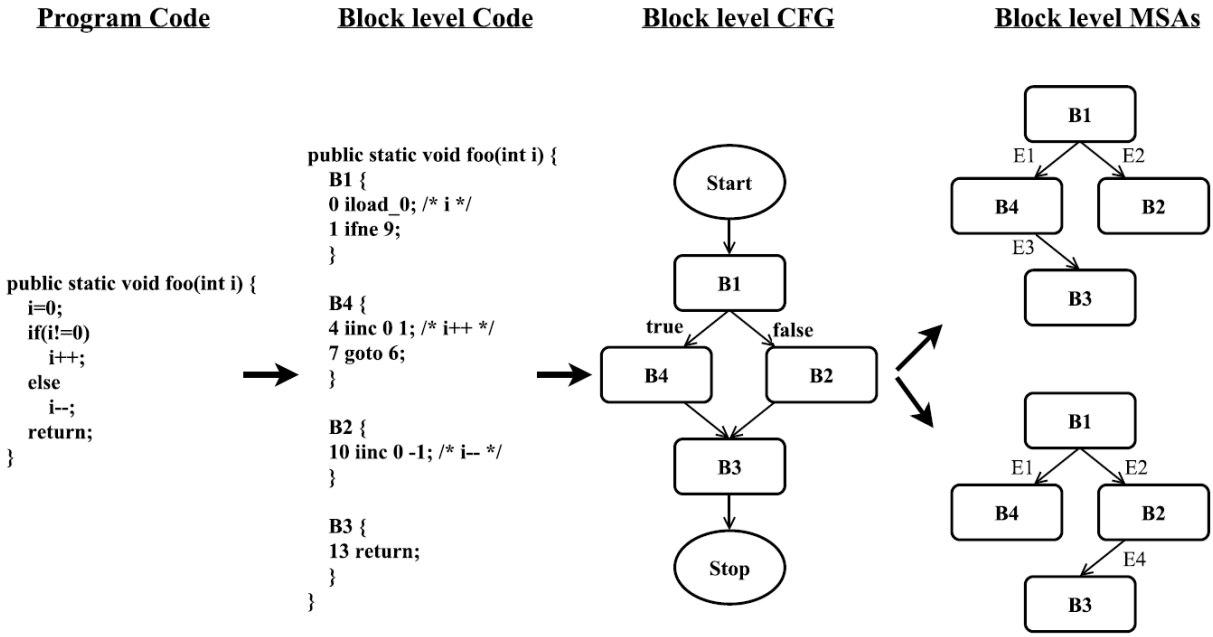


Рис. 2.10. Множинні MSA для одного CFG з дизасембльованого об'єктного коду

Традиційно системи виявлення вторгнень (IDS) перевіряють програми на наявність відомого шкідливого програмного забезпечення шляхом пошуку сигнатур у базі даних загроз. Пошук за сигнатурою з використанням точного співпадіння рядків має обмежену сферу застосування, оскільки варіанти однієї й тієї ж атаки матимуть різні сигнатури. Статичний аналіз з використанням CFG є ефективним методом виявлення вторгнень, хоча й дуже складним. Перетворення CFG у рядок та реалізація пошуку рядків є іншим способом вирішення, але результат не буде поліноміальним. Крім того, CFG на рівні базових блоків можуть мати варіанти базових блоків, які

візуально відрізняються, але виконують ту ж функцію. Для подолання цих недоліків було запропоновано багато методів наближеного співпадіння.

Відстеження додатків для отримання їх CFG — це інший підхід, який використовується в таких системах, як X-Trace, Pivot Trace. У системах великих даних, де вузли даних мають однакову архітектуру ЦП, можна припустити, що варіантів CFG на рівні байтів не буде. Таким чином, перевірки подібності CFG двох процесів достатньо для підтвердження узгодженості у вузлах.

Цілісність потоку управління (Control Flow Integrity, CFI) є популярною та ефективною технікою запобігання атакам, яка примушує виконання програми слідувати шляхом, що належить до її заздалегідь визначеного статичного CFG. Може використовуватися груба або дрібнозерниста версія CFI для профілювання програми. Однак, проблема будь-яких таких методів профілювання полягає у великих накладних витратах, особливо при віддаленому виконанні.

Існує безліч алгоритмів подібності коду на основі CFG, але вони, як правило, складні, дорогі і не мають визначених стандартів. Більшість з них покладаються на методи спрощення, такі як відбитки пальців (fingerprinting) або редагувальна відстань. Складності та невизначеності, пов'язані з генерацією та аналізом CFG, призвели до розробки нових методів аналізу потоку управління, які уникають перекладу коду програми у формальну модель. Наприклад, було запропоновано виявлення внутрішніх атак на основі символьного виконання та перевірки моделей асемблерного коду.

У даній роботі пропонується підхід для перевірки подібності потоку управління, який повністю відкидає ідею побудови CFG. Натомість, запропонована ідея базується на простому співпадінні рядків послідовностей інструкцій управління, отриманих з асемблерного коду запланованих процесів. Інший метод, символьне виконання, також страждає від вибуху шляхів (path explosion) і тому зазвичай обробляється за допомогою евристик пошуку.

2.7. Інтеграція методів глибоко навчання для аналізу шаблонів використання пам'яті в обчислювальних середовищах великих даних

Розуміння шаблонів доступу до пам'яті додатків великих даних є критично важливим для їхнього профілювання з точки зору використання даних. При спостереженні за доступом процесу до пам'яті можуть бути використані такі характеристики, як використання пропускної здатності, співвідношення читання/запису або тимчасова та просторова локальність.

Наприклад, в [24] спостерігали, що шаблони доступу до пам'яті навантажень великих даних мають багато спільного з традиційними паралельними навантаженнями, але, як правило, демонструють слабку тимчасову та просторову локальність. У розподілених обчислювальних системах вузли кластера часто є віртуальними машинами (ВМ). Наприклад, Datanode у Hadoop є процесом, який динамічно розподіляє завдання. Отже, профілювання розмірів приватних та спільних доступів до пам'яті всіх завдань дозволяє отримати агрегований шаблон доступу до пам'яті Datanode.

2.7.1 Метод аналізу головних компонент

Аналіз Головних Компонент (Principal Component Analysis, PCA) є методом лінійного перетворення без навчання (unsupervised linear transformation), метою якого є знаходження напрямків максимальної дисперсії у заданому наборі даних.

Головний компонент — це лінійна комбінація всіх змінних, яка максимально зберігає інформацію про ці змінні. При використанні для підгонки лінійної регресії, PCA мінімізує перпендикулярні відстані від даних до підігнаної моделі. Це відповідає лінійному випадку ортогональної регресії або загального методу найменших квадратів, що є ідеальним, коли відсутнє природне розмежування між предикторами та змінними відгуку.

Це робить PCA ідеальним для порівняння шаблонів доступу до пам'яті, оскільки характеристики доступу до пам'яті часто є стохастичними і не

слідують жодному відношенню предиктор-відгук. Хоча PCA є лише одним із можливих інструментів; інші методи, як-от моделі суміші Гауса (Gaussian Mixture Models, GMM) або тести на однорідність дисперсії (наприклад, тест Бокса або тест Бартлетта), також можуть бути застосовані.

Згідно з теорією ортогональної регресії з PCA, p спостережуваних змінних можуть бути апроксимовані r-вимірною гіперплощиною у p-вимірному просторі, де $r \leq p$. Вибір r еквівалентний вибору кількості компонентів для збереження в PCA. У цій роботі r і p є однаковими, оскільки немає потреби у зменшенні розмірності набору даних. Для профілювання використання пам'яті процесом та його подальшого порівняння з іншими профілями необхідна функція, здатна пояснити поведінку пам'яті. З цією метою використовуються значення T-квадрат, які можуть бути обчислені за допомогою PCA у повному просторі.

2.7.2. Довгострокова пам'ять (LSTM)

Нейронні мережі є ефективними інструментами для побудови прогностичних моделей, здатних навчатися регулярностям у послідовностях.

Базова нейронна мережа є мережею прямого поширення (feedforward network), побудованою з перцептронів. Навчання відбувається шляхом коригування ваг та зміщень з часом за допомогою функції оптимізації, відомої як алгоритм градієнтного спуску. Зворотне поширення (backpropagation) є найпопулярнішим методом для розрахунку градієнта, який вказує, наскільки швидко змінюється функція втрат (cost) при зміні параметрів мережі. Мережі прямого поширення здебільшого використовуються для задач класифікації.

Модифікація нейронних мереж, відома як рекурентні нейронні мережі (RNN), набуває популярності завдяки їхній ланцюговій структурі. Вони можуть з'єднувати перцептрони у спрямований цикл, що створює внутрішню пам'ять для зберігання частин вхідної послідовності та проміжних результатів. Однак, зворотне поширення в таких мережах є надзвичайно

часозатратним. Більше того, стандартні RNN стикаються з проблемою зникнення або вибуху градієнта (vanishing or exploding gradient), що призводить до нездатності навчатися за наявності значних часових затримок між відповідним вхідними подіями та цільовими сигналами.

Для вирішення цієї проблеми було запропоновано архітектуру довгострокової пам'яті (Long Short-Term Memory, LSTM). LSTM допомагає зберігати помилку, дозволяючи їй ефективно поширюватися назад у часі та через шари.

Мережі LSTM функціонують за допомогою комірок (cells), які складаються з набору вентилів (gates):

- Вентиль забування (Forget Gate, ft) визначає, яку частину стану комірки слід відкинути.

- Вхідний вентиль (Input Gate, it) визначає, яку частину нового вхідного сигналу слід додати до стану комірки.

- Вентиль комірки (Cell Gate, Ct) обчислює потенційний новий вміст для стану комірки.

- Вихідний вентиль (Output Gate, ot) визначає, яка частина стану комірки буде вихідним сигналом.

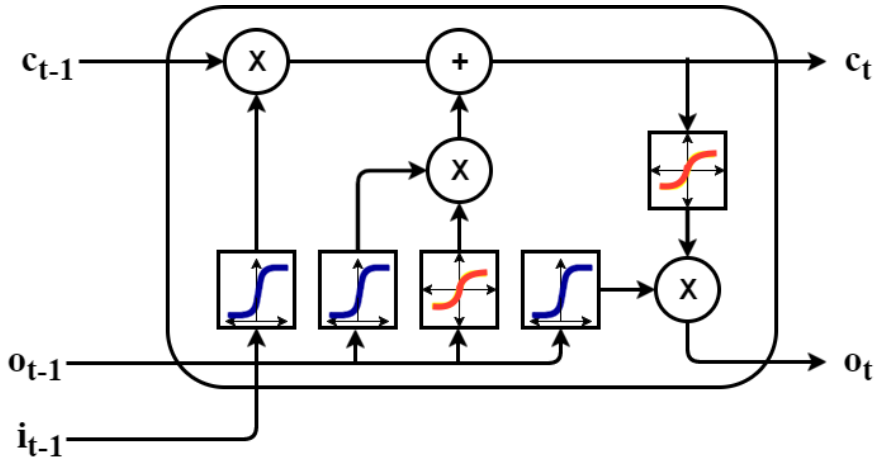


Рис. 2.11. Приклад дизайну комірки LSTM

Ці комірки та їхні вентилі допомагають зберігати, додавати або видаляти інформацію з одного проходу до іншого. Стандартна реалізація

LSTM використовує всі вентилі та покладається на функції активації σ (сигмоїда) для відкриття/закриття вентилів та tanh для пропорційного вибору даних та стану комірки.

Кожна комірка LSTM на кожному кроці часу t має три значення даних: стан комірки (ct), вхід (it) та вихід (ht). Загальний вигляд роботи комірки LSTM (як показано на рисунку 2.11) описується рівняннями:

$$
\begin{aligned}
f_t &= \sigma(W_{fg} \cdot [h_{t-1}, x_t] + b_{fg}) \\
i_t &= \sigma(W_{ig} \cdot [h_{t-1}, x_t] + b_{ig}) \\
\tilde{C}_t &= \tanh(W_{cg} \cdot [h_{t-1}, x_t] + b_{cg}) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_{og} \cdot [h_{t-1}, x_t] + b_{og}) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

де W_{fg}, W_{ig}, W_{og}, W_{cg} — ваги, b_{fg}, b_{ig}, b_{og}, b_{cg} — зміщення, а t−1,t — два послідовні кроки в послідовності.

LSTM також використовує стохастичні функції оптимізації (наприклад, Adam, RMSprop або стохастичний градієнтний спуск) для вимірювання градієнта. Оскільки немає остаточних доказів щодо найкращої функції оптимізації для LSTM, для даної роботи випадково обрано Adam. Дослідження численних варіацій LSTM не є основним фокусом цієї роботи.

Поточний стан у сфері загальної кібербезпеки переважно зосереджений на боротьбі з більш поширеними векторами ризику, такими як ін'єкції (injection flaws), міжсайтовий скриптинг (cross-site scripting, XSS) та компрометація автентифікації (broken authentication).

У рамках даного дослідження ми проводимо поглиблений аналіз проблеми внутрішніх атак (insider attacks). Цільовим середовищем для дослідження обрано системи великих даних, а основним завданням є захист чутливих даних від несанкціонованого розголошення (disclosure) та знищення (destruction).

Висновки до розділу

У другому розділі зосереджено увагу на аналізі сучасних методів виявлення інсайдерських атак, а також на дослідженні архітектурних вразливостей у середовищах великих даних. Проведено огляд наукових підходів, фреймворків і технологічних платформ, орієнтованих на виявлення вторгнень у хмарних системах. Визначено, що більшість наявних систем безпеки орієнтовані на зовнішні загрози та не забезпечують належного рівня захисту від користувачів із внутрішнім доступом. Детально досліджено вектори інсайдерських атак, пов'язані з маніпуляцією доступом до пам'яті, некоректною роботою системних викликів та використанням програмних вразливостей.

На основі експериментального моделювання інсайдерських атак доведено, що традиційні методи сигнатурного аналізу не здатні ідентифікувати складні, динамічні сценарії поведінки користувача. У зв'язку з цим запропоновано підхід до поведінкового моніторингу на основі комбінації методів аналізу головних компонент (PCA) і моделей довгострокової пам'яті (LSTM). Розроблена методологія аналізу потоків управління дозволяє ідентифікувати нетипові послідовності викликів та дії, характерні для інсайдерських загроз. Показано, що поєднання статистичного аналізу та машинного навчання підвищує точність виявлення атак і знижує кількість хибнопозитивних спрацьовувань.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МОДЕЛЕЙ ТА МЕТОДОЛОГІЇ У ФРЕЙМВОРК ЗАПОБІГАННЯ ІНСАЙДЕРСЬКИМ АТАКАМ У РОЗПОДІЛЕНИХ ОБЧИСЛЮВАЛЬНИХ СЕРЕДОВИЩАХ

3.1. Архітектура та методологія делегування безпеки великих даних апаратному забезпеченню

Запропоновано архітектурну парадигму делегування функцій забезпечення безпеки до апаратно-захищених середовищ із подальшою інтеграцією в розподілену інфраструктуру обробки великих даних. Теоретичною основою підходу слугують формалізовані властивості відмовостійкості, зумовлені механізмом багаторазової реплікації наборів даних. У межах концепції розроблено комплексний набір криптографічно та апаратно посилених методів, що експлуатують згадані властивості для підвищення рівня довіри до обчислень.

Для системної реалізації запропоновано двофазний security-фреймворк, який складається з наступних компонентів:

1. Локальна фаза верифікації програмного коду (Phase-I): поєднання статичного та динамічного аналізу на рівні окремих робочих вузлів кластеру.

2. Глобальна фаза валідації результатів обчислень (Phase-II): протокол консенсусу для зіставлення вихідних даних, отриманих незалежними реплікаційними вузлами, із формальним досягненням візантійської згоди.

3.1.1. Локальний аналіз

Перша фаза комплексного підходу передбачає локальну верифікацію програмного коду безпосередньо на вузлах зберігання даних (datanodes). Верифікація здійснюється у два послідовні етапи, що охоплюють як статичні, так і динамічні аспекти аналізу:

Статичний аналіз (Static Analysis) — виконується до запуску програми на виконання. На цьому етапі застосовуються методи компіляційної безпеки,

зокрема аналіз потоку керування, отриманий з бінарних файлів, з використанням інструкційно-рівневих правил та евристичних алгоритмів для виявлення потенційних вразливостей.

Динамічний аналіз (Runtime Analysis / Profiling) — реалізується під час виконання програми та після її завершення. Цей етап спрямований на виявлення й прогнозування атак у реальному часі, а також на пост-виконавчий аудит. У межах запропонованого підходу динамічний аналіз зосереджений на моніторингу системних викликів і патернів використання пам’яті процесом, що дозволяє фіксувати аномальну поведінку на рівні операційної системи.

3.1.2. Динамічна перевірка

На другому етапі верифікації кожен datanode ініціює міжвузлову валідацію аналітичних артефактів, отриманих від інших реплік того самого блоку даних. Процедура формалізована як задача зіставлення (matching problem), у якій локальний профіль виконання програми зіставляється з відповідними профілями, згенерованими консорціумом реплікаційних вузлів.

Для кількісної оцінки подібності артефактів застосовується параметризований набір метрик, що включає:

- графові алгоритми ізоморфного зіставлення CFG/CFG та CG/CG;

- рядкові метрики (Levenshtein, Jaro-Winkler, KL-дивергенція) для порівняння послідовностей системних викликів;

- статистичні критерії узгодженості (χ^2, Cramér-von Mises) для розподілів тактових профілів;

- глибокі сіамські нейромережі, навчені в режимі triplet loss, що проектують виконавчі трейси у метричний простір постійної розмірності.

Після завершення зіставлення вузли переходять до фази візантійського консенсусу. Для досягнення відмовостійкої згоди використовується або класичний протокол сімейства Paxos/Raft з адаптованим механізмом leader-election, що враховує реплікаційний фактор, або спеціалізований BFT-

алгоритм (наприклад, PBFT, Tendermint), оптимізований під обмежену пропускну здатність мережі HDFS.

3.1.3. Безпечна комунікація

Для забезпечення конфіденційності та цілісності міжвузлового обміну в розподіленій системі великих даних, архітектурно автономний модуль безпечної комунікації інтегровано ізоляційно від базових служб кластеру. Запропонований протокол реалізує асиметричну криптосистему з відкритим ключем, що базується на RSA-подібному обміні ключами з короткочасними парами, що динамічно ротуються.

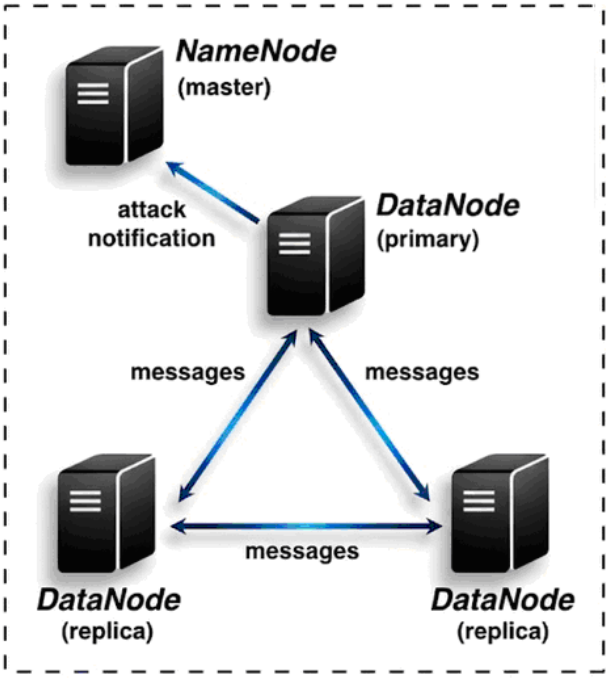


Рис. 3.1 Загальний огляд запропонованої архітектури безпеки

Генерація ключів. Джерелом ентропії слугує апаратно захищений генератор випадкових чисел (HRNG), інкапсульований у довіреному середовищі (TPM 2.0 або Intel TXT). Майстер-ключ K_m, що зберігається в недоступному для програмного рівня сховищі, використовується як seed для періодичного породження епізодичних асиметричних пар (sk_i, pk_i) з інтервалом $\Delta t \ll T$, де $T = 10$ с – час життя активної пари.

Розподіл ключів. Публічні компоненти pk_i транслюються через ізольований канал broadcast до всіх реплікаційних вузлів і головного вузла (namenode). Закриті sk_i зберігаються лише у локальному сховищі довіреного середовища та стираються одразу після закінчення інтервалу T.

Шифрування. Кожен пакет m підписується відправником цифровим підписом $\sigma = Sign(sk_i, H(m))$, після чого шифрується симетричним сеансовим ключем k_s, що, у свою чергу, упаковується асиметрично $c = Enc(pk_j, k_s)$. Отже, реалізується гібридна схема «sign-then-encrypt», що гарантує одноразову секретність і незаперечність походження.

Управління станом. На кожному вузлі підтримується циклічна черга фіксованої довжини $k = 5$, що містить актуальні та чотири попередні pk. Це забезпечує можливість верифікації пакетів, доставлених із затримкою $\leq k \cdot T$, і прискорює відновлення після зриву синхронізації годинників або DoS-атаки на службу розподілу ключів.Запропонований фреймворк є комбінацією незалежних модулів безпеки, які потребують протоколу безпечної комунікації для обміну життєво важливою інформацією про аналіз процесу.

3.1.4. Алгоритм протоколу безпечної комунікації

Протокол передбачає, що вузли періодично (кожні T одиниць часу) генерують нові пари відкритих/закритих ключів (newkpn).

Генерація. Новий закритий ключ (privn) використовується для розшифрування вхідних даних від вузла n, а відповідний відкритий ключ (pubn) передається вузлу n. Поточні закриті ключі всіх вузлів зберігаються у масиві arr_priv[].

Використання. Для відправки повідомлення (msg) реплікаційним вузлам, використовується відкритий ключ цього вузла для створення зашифрованого повідомлення (msg_e). Масив черг arr_pub[] зберігає відкриті ключі, отримані від усіх інших вузлів, необхідні для шифрування повідомлень, що надсилаються до них. Після розшифрування повідомлень інформація передається модулю співставлення для ідентифікації атак.

Лістинг 3.1. Псевдокод алгоритму протоколу безпечної комунікації

```
def SECURE_COMMUNICATION_PROTOCOL():
    # Безкінечний цикл для постійної роботи протоколу
    while True:
        # 1. ФАЗА ГЕНЕРАЦІЇ КЛЮЧІВ
        # Перевірка, чи настав час T для генерації нових ключів
        if time == T:
            # Генерація ключів для зв'язку з кожним іншим вузлом
            for n in OtherNodes:
                # Отримання нової пари публічного/приватного ключа з чіпа TPM
                newkp_n = get_new_public_private_key_pair_from_TPM_chip()

                # Вилучення відкритого ключа (для шифрування повідомлень до n)
                pub_n = get_public_key_from(newkp_n)

                # Вилучення приватного ключа (для розшифрування вхідних повідомлень
                priv_n = get_private_key_from(newkp_n)

                # Надсилання відкритого ключа вузлу n
                node_n.send(pub_n)

                # Зберігання приватного ключа для розшифрування вхідних повідомлень
                arr_priv[n] = priv_n

        # 2. ФАЗА УПРАВЛІННЯ ЧЕРГОЮ ВІДКРИТИХ КЛЮЧІВ
        # Оновлення черг відкритих ключів для кожного іншого вузла
        for n in OtherNodes:
            # Якщо черга вузла n повна, видалити найстаріший ключ
            if queue_n == full:
                dequeue(queue_n)


            # Додати щойно отриманий відкритий ключ (pub_n) до черги
            queue_n.enqueue(pub_n)

            # Оновлення масиву з посиланням на чергу вузла n
            arr_pub[n] = queue_n

        # 3. ФАЗА ВИКОРИСТАННЯ КЛЮЧІВ (ШИФРУВАННЯ/НАДСИЛАННЯ)
        # Повідомлення для надсилання всім реплікам
        msg = "to be sent to all replicas" # msg - це повідомлення, яке потрібно над

        # Цикл надсилання повідомлення кожній репліці r
        for r in Replicas:
            # Отримання поточного (найновішого) відкритого ключа репліки r з черги
            pub_r = back(arr_pub[r]) # back() повертає найновіший ключ, який знаходит

            # Шифрування повідомлення msg за допомогою відкритого ключа pub_r
            msg_e = encrypt(msg, pub_r)

            # Надсилання зашифрованого повідомлення
            send(msg_e)
```

Цей код ілюструє, як вузол періодично генерує та обмінюється ключами, керує чергами ключів для підтримки безпечного зв'язку та використовує ці ключі для шифрування повідомлень, що надсилаються реплікаційним вузлам.

3.2. Архітектура та елементи пропонованого фреймворку безпеки

Протокол реалізує періодичну ротацію асиметричних ключів із періодом T: кожен вузол і щоразу генерує нову ключову пару

$$newkpn^{(i)} = (privn^{(i)}, pubn^{(i)}), \quad privn^{(i)} \leftarrow F_2\text{^}\lambda, \quad pubn^{(i)} = g\text{^}(privn^{(i)}) \bmod p,$$

де $\lambda \geq 2048$ біт, а параметри (p, g) узгоджено на етапі ініціалізації кластеру.

У довіреній пам’яті вузла формується масив закритих ключів

$$arr\_priv[0 \ldots N-1], \quad arr\_priv[i] = privn^{(i)},$$

доступний лише всередині апаратно ізольованого середовища (TEE).

Розповсюдження відкритих ключів. Публічна складова $pubn^{(i)}$ передається решті вузлів через ізольований канал broadcast-типу; на приймальній стороні ключі інтерпретуються як елементи черги з довжиною k = 5 і зберігаються у масиві

$$arr\_pub[j][0 \ldots k-1], \quad j \neq i,$$

що дозволяє верифікувати пакети, затримані $\leq k \cdot T$.

Шифрування повідомлень. Для передачі даних msg від вузла i до вузла j формується сеансовий ключ $k_s \leftarrow \{0,1\}^{256}$, далі

$$c_1 = Enc_{pubn^{(j)}}(k_s), \quad c_2 = Enc_{k_s}(msg \| H(msg)), \quad msg\_e = (c_1 \| c_2).$$

Отримувач виконує Dec_{privn$^{(j)}$}(c_1) → k_s і, після перевірки цілісності, передає розшифрований профіль модулю зіставлення для подальшого виявлення відхилень, індикативних щодо візантійських атак. Огляд основних елементів запропонованої моделі системи надано на рисунку 3.2. Нижче наведено функціональне призначення кожного з цих елементів.

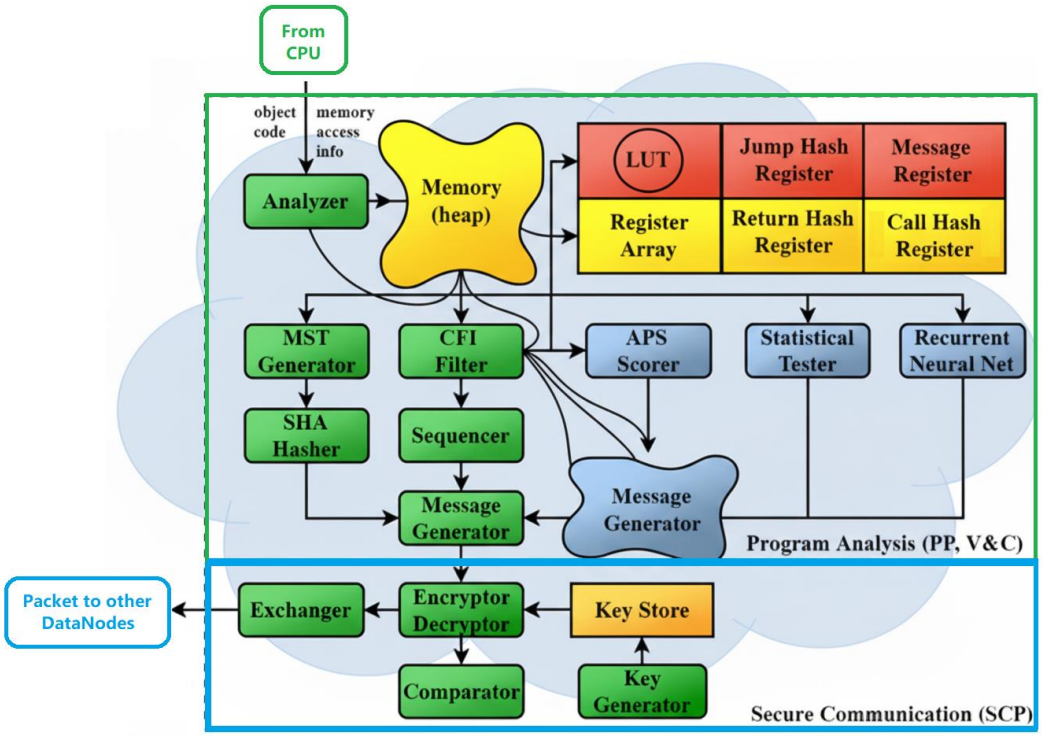


Рис. 3.2. Базова модель для запропонованого фреймворку безпеки

Таблиця 3.1.

Призначення елементів та модулів системи

Модуль / елемент	Функціональне призначення
Аналізатор	Отримує дані з віртуальної машини Hotspot, виконує початкове очищення даних та зберігає результат у пам'яті.
Фільтр CFI (CFI Filter)	Отримує вхідний набір інструкцій мови асемблера від аналізатора і фільтрує інструкції потоку управління, зберігаючи їхній порядок.
Статистичний тестер	Реалізує різні статистичні тести (наприклад, PCA, ANOVA, Tukey) для аналізу інформації про використання пам'яті.
RNN	Реалізує рекурентну нейронну мережу для прогнозування атак на основі профілів використання пам'яті.
APS Scorer (Обчислювач APS)	Обчислює оцінки ймовірності атаки (Attack Probability Scores, APS) для програм, використовуючи правила, визначені у таблиці пошуку

Модуль / елемент	Функціональне призначення
Секвенсери (Jump, Call, Return)	Три окремі модулі. Кожен секвенсер обробляє вихід модуля Фільтра CFI для формування розділеного рядка послідовності відповідних інструкцій. Потім він використовує SHA Hasher для генерації та зберігання хешу фіксованого виходу цього рядка послідовності інструкцій.
Масив регістрів	Містить 4 регістри для зберігання повідомлення, хешу інструкції стрибку, хешу інструкції виклику та хешу інструкції повернення.
Регістр повідомлень	Спеціальний регістр у Масиві регістрів, що використовується для потокобезпечного зберігання фінального повідомлення.
Генератор повідомлень	Об'єднує всі окремі хеш-виходи з регістрів, використовує SHA Hasher для генерації хешу хешів. Цей фіксований хеш об'єднується з метаданими процесу для генерації та зберігання повідомлення, що репрезентує процес.
Шифрувальник/ Дешифрувальник	Використовує сховище ключів для доступу до пари ключів та Регістр повідомлень. Шифрувальник використовує відкритий ключ реплікаційного вузла для шифрування повідомлення. Дешифрувальник використовує закритий ключ вузла для розшифрування вхідного повідомлення.
Порівнювач	Виконує порівняння рядків між локальним повідомленням (хешем хешів) та отриманим повідомленням від реплікаційного вузла.
Генератор ключів	Використовує вбудовану функціональність чіпа TPM/TXT. Застосовує жорстко закодований ключ, генератор випадкових чисел та таймер чіпа для періодичної генерації нових пар відкритих/закритих ключів.
Сховище ключів	Використовує масив пам'яті для зберігання черг відкритих ключів усіх реплікаційних вузлів та поточної пари ключів даного вузла. Зберігає три найновіші відкриті ключі кожного реплікаційного вузла.
Обмінник	Використовує протокол TCP/IP для обміну повідомленнями з іншими вузлами.

3.3. Методологія динамічного аналізу та модель загроз

Впровадження технік безпеки на етапі компіляції є важливим для мінімізації поверхні атаки системи. У межах даної роботи запропоновано

наступну групу методів, які мають динамічний характер і функціонують під час виконання програми (run-time). Якщо методи, реалізовані на компіляційному етапі, переважно сфокусовані на аналізі потоку керування, то run-time-методи зосереджені на аналізі поведінки програми в пам’яті.

Запропоновано два основних підходи:

- Виявлення інцидентів вторгнення (Intrusion Detection) після факту їхньої реалізації.

- Прогнозування ймовірності вторгнення до моменту його здійснення.

З огляду на те, що основною метою є комплексний захист кластера big-data, а не лише блокування конкретних векторів атак, і що аналіз базується на динамічній поведінці системи, обґрунтованою є розробка узагальненої моделі загроз саме для run-time-підходу. Запропоновані run-time-методи функціонують як додатковий рівень захисту до компіляційних технік, які зазвичай орієнтовані на специфічні типи атак.

У роботі застосовано програмно-центричну модель загроз, яка використовується для декомпозиції архітектури платформи big-data з метою ідентифікації операційних вразливостей. До основних категорій загроз, що розглядаються, належать: погіршення даних (data degradation), крадіжка даних (data theft) та маніпулювання конфігурацією (configuration manipulation).

Ці загрози формують підмножину інсайдерських атак, які часто недостатньо вивчені у контексті систем big-data. Усі три загрози безпосередньо корелюють із компрометацією даних, розміщених у кластері, і, відповідно, тісно пов’язані з інфраструктурою кластера та поведінкою пам’яті DataNode. При цьому модель не охоплює вразливості, що виникають поза кластером (наприклад, на мережевих або користувацьких рівнях).

Ключова увага приділяється операційним вразливостям, які виникають під час виконання (рантайму) внаслідок неправильного використання програмістами системних або бібліотечних викликів. Цей фокус обумовлений двома чинниками: (а) неможливістю апріорної оцінки впливу

таких помилок; (б) поширеним припущенням про повне усунення помилок програмування на етапі компіляції.

До моделі також включено загрози, пов'язані з неправомірним використанням інсайдером своїх прав доступу до даних, що моделюється як "зла" DataNode, яка маскується під легітимний вузол. Виявлення подібної активності можливе через аналіз шаблонів доступу до пам’яті на рівні процесу. Оскільки диференціація між "незвичайними" та "підробленими" доступами становить значну складність, у межах даного дослідження обидва випадки розглядаються як вектори загроз.

Для забезпечення коректності моделі та її застосування прийняті наступні припущення:

- Усі вузли даних (DataNode) мають ідентичну архітектуру, операційну систему (ОС) та розмір сторінки пам'яті.
- Репліки (копії) даних містять подібний контент.
- Шлях інсталяції фреймворку є уніфікованим для всіх DataNode.
- Мережа зв’язку є цілісною та функціонує без порушень.
- Вартість зв’язку між репліками є не більшою за вартість зв’язку між NameNode та DataNode.

3.4. Метод виявлення вторгнень, базований на шаблонах доступу до пам'яті та системних викликах

Виявлення вторгнень, яке спирається на зіставлення шаблонів (pattern matching), є апробованим підходом, що застосовується вже близько трьох десятиліть. У типовому зіставленні шаблонів визначається клас очікуваної або легітимної поведінки, і необроблені дані порівнюються з цим класом для ідентифікації аномалій. Поширеними параметрами, що використовуються для виявлення вторгнень, є системні виклики, активність користувача, мережеві пакети та події. Зокрема, використання системних викликів для

детекції загроз залишається популярним у галузі безпеки протягом останніх двох десятиліть.

У цій роботі ідея зіставлення шаблонів адаптується для домену безпеки великих даних (big data) із застосуванням двох системних параметрів:

а) системні виклики;

б) доступи до пам'яті.

Запропонована методологія передбачає побудову профілю поведінки процесу на основі частот системних викликів та шаблонів доступу до пам'яті. Для опису шаблону доступу до пам'яті використовуються три характеристики: резидентний набір (resident set), спільні сторінки (shared pages) та приватні сторінки (private pages). Локально побудований профіль поведінки процесу надалі порівнюється з профілем аналогічного процесу, запланованого до виконання того ж завдання на репліці DataNode.

Перший етап запропонованого рішення сфокусований на формуванні профілю поведінки для кожного процесу, що виконується на DataNode. Профіль поведінки процесу конструюється шляхом спостереження за його характеристиками. У цьому дослідженні поведінка процесу описується на основі системних і бібліотечних викликів та операцій доступу до пам'яті, здійснених під час його виконання. Системні та бібліотечні виклики відображають функціональність процесу, тоді як доступи до пам'яті — характер використання даних. Така двоаспектна інформація ускладнює маскування для потенційних зловмисників.

Алгоритм лістингу 3.2 деталізує процедуру створення профілю поведінки процесу. Структура даних, що представляє профіль поведінки процесу на DataNode, містить три ключові компоненти:

1) ідентифікатор процесу (повинен бути уніфікованим для процесу на різних DataNode),

2) відображення (map) з одним записом на кожен виклик,

3) вектор T2, отриманий за допомогою методу головних компонент (PCA) на основі інформації про доступ до пам'яті.

Лістинг 3.2. Алгоритм (псевдокод) створення профілю поведінки процесу, що базується на системних викликах та доступі до пам'яті

```
procedure BEHAVIOR PROFILE
pid ← get the process id of datanode
interval ← set periodic interval for measurement
getProfile(pid):
    Profile ← empty map
    Calls ← call getCalls(pid)
    MemAccess ← call getMemAccess(pid)
    Hash ← hash of all call paths
    Profile ← insert([Hash, Calls], MemAccess)
return Profile

getCalls(pid):
    while callstack(pid) = system or library call do
        callee ← store the callee
        signature ← store the signature of the method
        callPath ← store the path
        callCount ← +1
        hash ← hash of the path
        info ← callee, signature, path, count
    return map(hash, info)

getMemAccess(pid):
    while elapsed = interval do
        if smaps(j).type = private or shared then
            thisAccess[0] ← smaps(j).Rss
            thisAccess[1] ← smaps(j).Private
            thisAccess[2] ← smaps(j).Shared
            MemAccess ← add thisAccess
    Result ← call PCA(MemAccess)
return Result
```

Проведемо реалізацію алгоритму (лістинг 3.2) на функціональний код на Python. Оскільки реальне отримання інформації про системні виклики та доступ до пам'яті вимагає низькорівневого доступу до ядра ОС або спеціалізованих інструментів (наприклад, ptrace, /proc/smaps), у цій імплементації використано заглушки (mock functions), які імітують отримання цих даних.

Ця програма засобами мови Python, що представлена в лістингу 3.3 імітує логіку наведеного вище алгоритму (лістинг 3.2), створюючи профіль поведінки процесу на основі імітованих даних системних викликів та параметрів пам'яті.

Лістинг 3.3. Фрагмент код створення профілю поведінки процесу на основі даних системних викликів та параметрів пам'яті

```
import time
import hashlib
import random
from collections import defaultdict

# --- Функції-заглушки для імітації низькорівневого доступу ---

def get_process_id_mock(datanode_name="DataNode-1"):
    """Імітує отримання ID процесу DataNode."""
    print(f"Отримання PID для {datanode_name}...")
    return hash(datanode_name) % 10000

def get_callstack_mock(pid):
    """Імітує отримання системних/бібліотечних викликів зі стеку."""

    # Імітація послідовності викликів
    possible_calls = {
        "read_file": {"sig": "int(fd, buf, count)", "path": "/lib/libc.so", "class'
        "write_log": {"sig": "int(fd, buf, count)", "path": "/lib/libc.so", "class'
        "malloc_mem": {"sig": "void*(size)", "path": "/usr/lib/libm.so", "class": '
        "network_send": {"sig": "int(sock, buf)", "path": "/usr/lib/libnet.so", "cl
    }

    # Генеруємо випадковий набір викликів для імітації
    calls_list = random.choices(list(possible_calls.keys()), k=random.randint(5, 15

    call_info_map = defaultdict(lambda: {"callee": "", "signature": "", "callPath"

    for call_name in calls_list:
        info = possible_calls[call_name]


       # Створення унікального хешу шляху виклику
       call_path_hash = hashlib.sha1(info["path"].encode('utf-8')).hexdigest()

       # Оновлення інформації про виклик
       call_info_map[call_path_hash]['callee'] = call_name
       call_info_map[call_path_hash]['signature'] = info["sig"]
       call_info_map[call_path_hash]['callPath'] = info["path"]
       call_info_map[call_path_hash]['callCount'] += 1

   # Формат повернення: map(hash, info)
   return dict(call_info_map)

def get_smaps_mock():
    """Імітує отримання інформації про сторінки пам'яті (smaps)."""

    # Імітуємо кілька записів smaps (приватних та спільних)
    mock_smaps_data = []

    # Private pages
    mock_smaps_data.append({"type": "private", "Rss": random.randint(1000, 5000), '
    # Shared pages
    mock_smaps_data.append({"type": "shared", "Rss": random.randint(500, 2000), "P
    # Another private section
    mock_smaps_data.append({"type": "private", "Rss": random.randint(2000, 6000), '

    return mock_smaps_data
```

```
def call_pca_mock(mem_access_data):
    """Імітує обчислення вектора T-squared (T²) після застосування PCA."""
    # У реальності тут буде складна матрична математика.
    # Імітуємо повернення вектора T-squared (наприклад, середнє значення)
    total_rss = sum(d[0] for d in mem_access_data)
    total_private = sum(d[1] for d in mem_access_data)
    total_shared = sum(d[2] for d in mem_access_data)

    # Повертаємо імітований T² вектор
    return [total_rss / 3, total_private / 3, total_shared / 3]

# --- Основні функції Алгоритму 7 ---

def getCalls(pid):
    """
    Алгоритм 7, рядки 11-19: Отримує та обробляє інформацію про системні/бібліотечні вик
    """
    print(f"Збір даних про виклики для PID: {pid}")
    return get_callstack_mock(pid)

def getMemAccess(pid, interval):
    """
    Алгоритм 7, рядки 20-28: Отримує інформацію про доступ до пам'яті та застосовує PCA
    """
    print(f"Збір даних про доступ до пам'яті кожні {interval} с...")

    mem_access_data = []
```

.....

```
    # 2: pid ← get the process id of datanode
    pid = get_process_id_mock(datanode_name)

    # 3: interval ← set periodic interval for measurement
    # Встановлюємо інтервал для збору даних пам'яті в секундах
    measurement_interval = 2.0

    # Виклик головної функції профілювання
    return getProfile(pid, measurement_interval)

# --- Приклад використання ---
if __name__ == "__main__":

    # Створення профілю для тестового DataNode
    process_profile = behavior_profile_procedure(datanode_name="Hadoop-Worker-7")

    print("\n--- ЗГЕНЕРОВАНИЙ ПРОФІЛЬ ПОВЕДІНКИ ПРОЦЕСУ ---")
    print(f"Ідентифікатор процесу (PID): {process_profile['identifier']}")
    print(f"Хеш шляхів виклику (для порівняння): {process_profile['call_info_hash']}"
    print(f"Кількість унікальних викликів: {len(process_profile['calls_map'])}")
    print("Приклад інформації про виклики (перший елемент):")

    # Виведення першого елемента для прикладу
    if process_profile['calls_map']:
        first_hash, first_info = list(process_profile['calls_map'].items())[0]
        print(f"  Хеш виклику: {first_hash[:10]}...")
        print(f"  Кількість: {first_info['callCount']}, Метод: {first_info['callee']

    print("\nВектор доступу до пам'яті (Імітація T² від PCA):")
    print(f"  [Середній RSS, Середній Private, Середній Shared]: {process_profile['me
    print("\nПрофіль готовий для порівняння з реплікою.")
```

3.4.1. Аналіз системних та бібліотечних викликів

Інструкція виклику (call instruction) у програмі має потенціал передати керування за межі програмного простору, що робить її привабливою мішенню для атак. У цій роботі увага зосереджена на системних і бібліотечних викликах. Системні виклики є програмним інтерфейсом для запиту служб ядра операційної системи. Їхній перелік зазвичай константний і обмежений (наприклад, ОС Linux має близько 140 системних викликів).

Оскільки робота орієнтована на платформи big data (наприклад, Hadoop або Spark), де імпліцитно використовуються численні сторонні бібліотеки, моніторинг бібліотечних викликів також є необхідним. Перевагою моніторингу бібліотечних викликів є можливість попереднього визначення набору об'єктів (JAR-файлів та спільних бібліотек), а також наявність фіксованих шляхів інсталяції фреймворків, які змінюються лише при модифікаціях на системному рівні.

Складність трасування системних та бібліотечних викликів у рантаймі полягає в тому, що порядок їхнього виконання може бути нестійким при багаторазових запусках однієї й тієї ж програми. Це створює проблему для системи безпеки, яка вимагає зіставлення інформації між репліками DataNode, оскільки точне порівняння стека викликів є непридатним для виявлення вторгнень у розподілених обчислювальних середовищах.

Для розв'язання цієї проблеми профіль поведінки процесу розроблений для дескриптивного опису викликів, а не для використання їхнього послідовного стека. Замість стека викликів, із нього витягуються метадані системних та бібліотечних викликів. Кожен запис у профілі описує виклик за допомогою чотирьох полів:

а) повне ім'я класу отримувача (callee),

б) сигнатура методу,

в) номер рядка вихідного коду,

г) кількість разів, які цей виклик був здійснений процесом.

Для швидкого пошуку використовується хеш повного імені класу як індекс. Варіабельність у кількості здійснення конкретного виклику між різними DataNode може бути використана як індикатор вторгнення.

3.4.2. Аналіз доступу до пам'яті

Незважаючи на ефективність моніторингу викликів, цей підхід залишається вразливим до інсайдерських атак. Наприклад, скомпрометована DataNode може маскувати свою активність, надсилаючи легітимну інформацію про виклики до фреймворку безпеки до моменту фактичної атаки, що призводить до хибних негативних результатів. Крім того, системні виклики надають інформацію про використання даних лише до моменту відображення файлу чи пристрою в пам'ять. Подальші операції read() та write() відбуваються через вказівники на відображену пам'ять.

З цих причин до профілю поведінки було включено інформацію про доступ процесу до пам'яті. Цей аспект забезпечує дрібнозернисту деталізацію для реконструкції подій і надає відомості про характеристики програми, такі як розмір приватної та спільної пам'яті, кількість чистих (clean) та змінених (dirty) сторінок тощо, що ускладнює маскування.

Переваги використання шаблонів доступу до пам'яті:

- Інформацію можна збирати періодично.
- Моніторинг легко реалізується за допомогою апаратних засобів.
- Надає розуміння аспектів використання даних процесом.
- Сприяє диференційованій конфіденційності.

Проте, аналіз пам'яті в сучасних ОС (наприклад, Linux) є складним, зокрема, через використання семантики "копіювання при записі" (copy-on-write) при розгалуженні процесів (forking) та відкладеного виділення фізичної пам'яті ядром до моменту першого доступу. Були обрані відносно прості для аналізу характеристики відображення пам'яті: приватні та спільні сторінки процесу в оперативній пам'яті (RAM). Збір цієї інформації відбувається періодично (кожні 2 секунди) для мінімізації накладних витрат.

3.4.3. Верифікація профілю поведінки

Алгоритм (лістинг 3.4) представляє загальний алгоритм верифікації профілю, а алгоритм поданий в лістингу 3.5 — алгоритм порівняння для розрізнення шаблонів доступу до пам'яті.

Лістинг 3.4. Алгоритм (псевдокод) для верифікації профілю поведінки процесу, що базується на порівнянні локальних даних із даними, отриманими від реплік DataNode

```
1: procedure VERIFY PROFILE
2:     pid ← get the process id from datanode
3:     Local ← behavior profile from this node
4:     Recv ← behavior profiles from other nodes

5:     compare():
6:         for thread t in pid do
7:             result1 ← call CompareCalls(t)
8:             result2 ← call CompareMemAccess(p)
9:             result ← result1 & result2
10:            notify result                    ▷ similarity in calls & memory acce
11:
12:     CompareCalls(t):
13:         for call c in t do
14:             if hash(c_path) = Recv.find() then
15:                 if count(c_Local) ≅ count(c_Recv) then
16:                     return true
17:                 else
18:                     return false
19:         return false  # Added for completeness if no calls are found/matched

20:     CompareMemAccess(pid):
21:         if compare(t²_Recv, t²_Local) then
22:             return true
23:         else
24:             return false
```

Цей алгоритм є ключовим елементом системи динамічного виявлення вторгнень. Його основне завдання — порівняти профіль поведінки процесу, що виконується на локальному вузлі (DataNode), із профілями, отриманими від реплік (інших DataNode), які виконують те ж саме завдання, для виявлення аномалій або компрометації. Алгоритм забезпечує двофакторну перевірку поведінки процесу, використовуючи як його код/логіку (виклики), так і його взаємодію з даними (пам'ять).

Лістинг 3.5. Алгоритм (псевдокод) для порівняння профілів поведінки процесів

```
1: procedure COMPARE PROFILES
2:     t²_Local ← get the process profile from datanode
3:     t²_Recv ← received process profiles
4:     for all t²_i do
5:         filter(t²_i)            ▷ remove tailing t² values
6:         sort(t²_i)
7:     if Anova(t²_Local, t²_Recv) then
8:         compromised ← Tukey(t²_Local, t²_Recv)
9:         return true
10:    else
11:        return false
```

Інформація про системні та бібліотечні виклики зберігається у структурі даних хеш-таблиці (hash map). Різниця на рівні шляху виклику визначається шляхом обчислення хешу (SHA-1) локального шляху виклику та пошуку цього хешу в індексній множині хеш-таблиці, отриманої від репліки. Неспівпадіння або відсутність збігу свідчить про використання DataNode різних наборів викликів для виконання однакового завдання, що є необхідною, але не достатньою умовою для фіксації вторгнення. Додатковою умовою є те, що різниця в кількості здійснення конкретного виклику має бути меншою за наперед визначений поріг (є).

Шаблон доступу до пам'яті процесу представляється векторами T2 від PCA. Оскільки значення T2 підпорядковуються F-розподілу, порівняння шаблонів здійснюється у два етапи:

- ANOVA-тест на векторах T2 для перевірки відмінності шаблонів.

- Тест Тьюкі (Tukey test) на результатах ANOVA для ідентифікації атакованого DataNode (якщо p-значення ANOVA є низьким, підтверджуючи відхилення нульової гіпотези).

Такий підхід, однак, не дозволяє чітко розрізняти незвичайну, але легітимну поведінку процесу від шкідливої (корумпованої) поведінки.

Продемонстровано дві типові робочі ситуації в кластері Hadoop:

- Запис файлу об'ємом 3 ГБ у HDFS: аналіз головних компонент (PCA) відображень пам'яті DataNode (рис. 3.3, 3.4) показали, що нульова гіпотеза

про рівність дисперсії розмірів доступу до пам'яті між DataNode підтверджується (високі p-значення > 0.5), що свідчить про гармонійну поведінку.

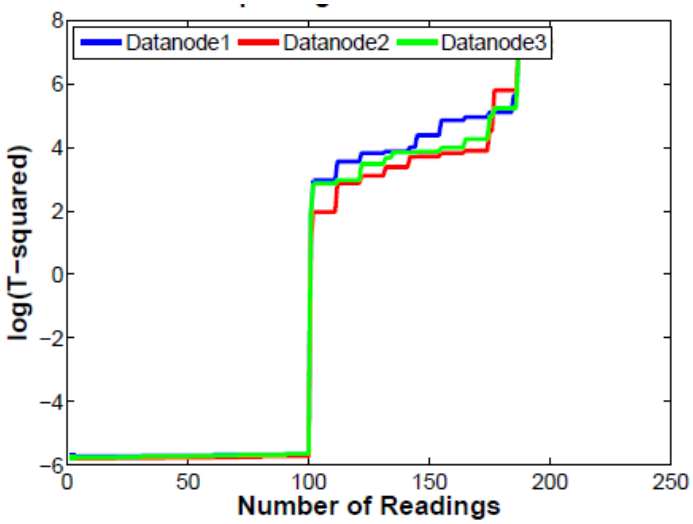


Рис. 3.3. Шаблони доступу до пам'яті вузлів даних під час розміщення 3 ГБ файлу в HDFS

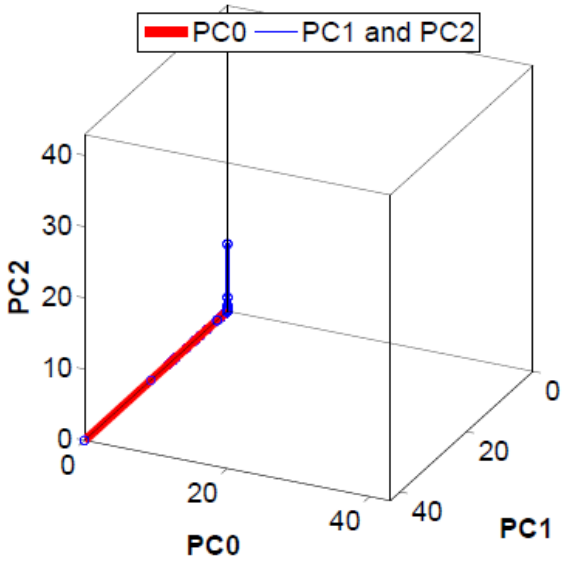


Рис. 3.4. Ортогональна регресія між трьома ознаками доступу до пам'яті

DataNode у стані простою (idle) - аналіз системних та бібліотечних викликів (рис. 3.5) виявив, що частоти і типи викликів є узгодженими на всіх

вузлах (наприклад, 275 викликів на вузол), що свідчить про узгодженість їхньої фонової активності.

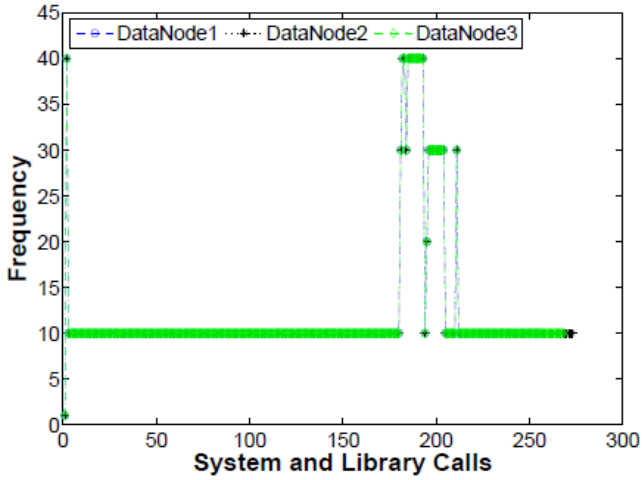


Рис. 3.5. Системні та бібліотечні виклики на підпорядкованих вузлах (Slave Nodes) у стані простою кластера

3.4.4. Архітектура фреймворку виявлення вторгнень

Запропонований алгоритм виявлення вторгнень підтримується існуючим фреймворком.

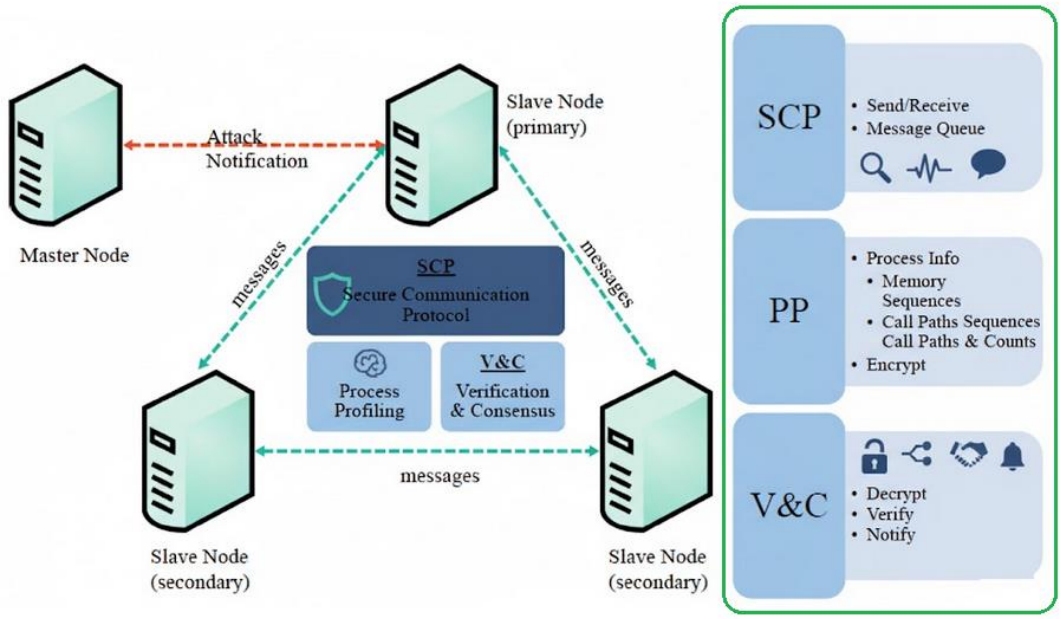


Рис. 3.6. Виявлення вторгнень під час виконання з використанням запропонованого фреймворку

Фреймворк включає безпечний протокол зв’язку (SCP) між вузлами з використанням шифрування. Його функціональність передбачається розмістити на співпроцесорі (coprocessor), який взаємодіє з центральним процесором (CPU) через захищений протокол. Фреймворк складається з двох основних фаз: профілювання процесу (PP) та верифікації й консенсусу (V&C), де використовуються алгоритми (лістинги 3.4 і 3.5). Розподілений характер цих алгоритмів дозволяє проводити фази профілювання та верифікації незалежно на кожному DataNode, що значно скорочує час виявлення вторгнень. Повідомлення про атаку надсилається на Master Node лише після досягнення консенсусу між DataNode щодо факту вторгнення.

3.4.5. Деталізація методів забезпечення безпеки на основі потоку завдань і LSTM

1. Статичний аналіз на етапі компіляції для виявлення вторгнень

Мета статичного аналізу — мінімізувати площину атаки на етапі виконання, виявляючи потенційні вразливості в коді до того, як він буде запущений на розподіленому кластері. Це особливо важливо для високопродуктивних систем великих даних, де ресурси виконання мають бути оптимізовані.

Замість аналізу всього програмного забезпечення, метод фокусується на конкретному завданні (job), яке користувач відправляє на виконання в кластері (наприклад, Hadoop MapReduce або Spark Task). Статичний аналіз застосовується до скомпільованого двійкового файлу (бінарного коду) або байт-коду цього завдання.

Методи адаптуються для перевірки цілісності потоку управління (Control Flow Integrity, CFI) завдання. Це допомагає виявити аномалії, викликані:

- неправильно розміщеними переходами (наприклад, перестрибування в несанкціоновану частину коду).

- використанням неініціалізованих аргументів.

- помилками, пов'язаними з пам'яттю (нульові або висячі вказівники).

- аналіз на рівні мови Асемблера (Low-Level Analysis).

Запропоновані методи працюють на нижчому рівні (наприклад, аналіз на рівні асемблера), що дозволяє виявляти вразливості, які можуть бути приховані на рівні високорівневої мови програмування.

Перевагою є виявлення і виправлення цих структурних недоліків на етапі компіляції дозволяє уникнути витрат ресурсів на їх моніторинг та виявлення вже під час виконання (runtime).

2. Використання LSTM для прогнозування атак

Основна ідея полягає в переході від реактивного виявлення вторгнень до проактивного прогнозування атак з метою мінімізації часу впливу та зменшення збитків.

Дані у великих розподілених системах (журнали, метрики продуктивності, шаблони доступу) часто містять нелінійні та складні часові залежності. Глибокі нейронні мережі ефективно моделюють ці нелінійні властивості. LSTM є типом рекурентної нейронної мережі (RNN), спеціально розробленої для роботи з послідовними даними та часовими рядами. На відміну від звичайних RNN, LSTM може зберігати інформацію протягом тривалих періодів часу (вирішує проблему "зникаючого градієнта"), що критично важливо для аналізу поведінкових шаблонів (наприклад, послідовності дій користувача-зрадника).

Механізм прогнозування в кластері:

1. LSTM тренується на послідовностях системних подій (журнали доступу, операції з файловою системою, шаблони доступу до пам'яті).

2. Модель ідентифікує корельовані шаблони використання та підготовчі дії, які передують відомим інцидентам внутрішніх атак.

На основі поточних послідовностей подій, модель прогнозує ймовірність того, що процес або користувач незабаром здійснить несанкціоновану дію, дозволяючи системі вжити превентивних заходів. На відміну від традиційного використання LSTM для роботи на кластерах (для

прискорення обчислень), тут LSTM використовується для аналізу даних усередині кластера з метою безпеки.

Висновки до розділу

У третьому розділі описано фреймворк запобігання інсайдерським атакам у розподілених обчислювальних середовищах, який інтегрує методи глибокого навчання, динамічного аналізу та апаратного делегування безпеки.

Запропоновано архітектуру, що включає три рівні:

1. Локальний аналіз, який відповідає за моніторинг поведінки користувачів і процесів;

2. Динамічну перевірку, що забезпечує реакцію на виявлені аномалії;

3. Безпечну комунікацію, яка гарантує цілісність даних і захищений обмін між вузлами системи.

Описано алгоритм протоколу безпечної комунікації, який реалізує криптографічний обмін ключами та забезпечує довіру між компонентами розподіленого середовища.

Розроблено модель загроз, яка враховує внутрішні вектори атаки, включаючи компрометацію користувача, зловживання правами доступу та несанкціоновані виклики бібліотек. Запропонований метод виявлення вторгнень ґрунтується на аналізі системних і бібліотечних викликів, а також на спостереженні за шаблонами доступу до пам’яті. Це дозволяє будувати поведінкові профілі процесів та здійснювати верифікацію їх легітимності в реальному часі.

Інтеграція моделі LSTM у фреймворк дала змогу підвищити здатність системи прогнозувати поведінку користувача та своєчасно виявляти потенційні інсайдерські дії до моменту їх активної реалізації.

ВИСНОВКИ

У магістерській роботі вирішено науково-прикладну задачу — підвищення рівня інформаційної безпеки розподілених обчислювальних середовищ шляхом розроблення моделей та методів запобігання інсайдерським атакам. Дослідження охоплює аналіз архітектурних, організаційних та поведінкових аспектів безпеки систем обробки великих даних, а також розроблення фреймворку, що забезпечує ефективне виявлення та попередження внутрішніх загроз.

Проведено системний аналіз предметної області забезпечення безпеки у розподілених обчислювальних середовищах. Визначено основні виклики, пов’язані з масштабованістю, динамічністю ресурсів, мультиорендністю та розподіленим доступом до даних. Виявлено, що інсайдерські атаки становлять одну з найбільш небезпечних загроз для таких систем, оскільки здійснюються з легітимними правами доступу та складно ідентифікуються традиційними методами контролю. Проаналізовано сучасні підходи та фреймворки виявлення атак у середовищах великих даних. Встановлено, що більшість існуючих рішень орієнтовані на зовнішні вторгнення, тоді як внутрішні загрози часто залишаються поза сферою ефективного моніторингу.

Досліджено типологію інсайдерських атак і методи їх аналізу. Сформовано класифікацію інсайдерських загроз залежно від мотивів, векторів дії та способів приховування. Розроблено підхід до поведінкового аналізу користувачів на основі моніторингу системних викликів і шаблонів доступу до пам’яті. Запропоновано методологію виявлення вторгнень, що поєднує метод аналізу головних компонент (PCA) та нейронні мережі з довгостроковою пам’яттю (LSTM). Такий підхід дозволяє виявляти аномальні дії в режимі реального часу, прогнозувати потенційні інсайдерські атаки та мінімізувати кількість хибнопозитивних спрацьовувань.

Розроблено архітектуру фреймворку безпеки, який реалізує багаторівневий механізм моніторингу, включно з локальним аналізом

поведінки, динамічною перевіркою процесів та захищеною комунікацією між вузлами системи. У фреймворк інтегровано протокол безпечної взаємодії, що забезпечує цілісність і достовірність обміну даними.

Проведено експериментальне моделювання інсайдерських атак у середовищі великих даних. Результати показали, що запропоновані методи виявлення забезпечують вищу точність і стійкість до прихованих загроз порівняно з традиційними сигнатурними системами.

Доведено ефективність інтеграції глибокого навчання у процеси безпеки розподілених систем. Застосування LSTM-мереж для аналізу поведінкових патернів дозволило формувати адаптивні профілі користувачів і виявляти відхилення від нормальної активності на ранніх стадіях атаки.

Результати дослідження підтвердили доцільність використання гібридного підходу, який поєднує поведінковий аналіз, машинне навчання та архітектурні механізми безпеки. Запропонована модель здатна забезпечити динамічну адаптацію системи до нових загроз, підвищити рівень довіри між компонентами розподіленого середовища та знизити ризики компрометації внутрішніх ресурсів.

Практичне значення одержаних результатів полягає у можливості застосування запропонованого фреймворку у хмарних та гібридних платформах обробки великих даних для побудови інтелектуальних систем моніторингу й захисту корпоративних обчислювальних інфраструктур.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems By Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca / https://www.cs.purdue.edu/homes/bb/cs542-22Spr/readings/others/pivot-tracing-cacm-202003.pdf
2. Hadoop - Architecture – GeeksforGeeks - https://www.geeksforgeeks.org/data-engineering/hadoop-architecture/
3. Duncan, A. (2015). An overview of insider attacks in cloud computing. Concurrency and Computation: Practice and Experience. DOI:10.1002/cpe.3243.
4. G. Deep (2022). “Insider threat prevention in distributed database as a service”. Computers & Security, 112, 102464.
5. Kim, S.Y. (2024). “LSTM Autoencoder-Based Insider Abnormal Behavior Detection”. Proceedings of the 2024 International Conference on Cybersecurity.
6. Solani, R. (2021). “iCOPS: insider attack detection in distributed file systems”. International Journal of Secure Computing and Parallel Systems, 9(4).
7. Ye, X., et al. (2025). “Research on insider threat detection based on federated learning”. Scientific Reports, 15, 04029.
8. Babu, C.V.S., Subhash, S., Vignesh, M., Jeyavasan, T. (2024). “Securing the Cloud: Understanding and Mitigating Data Breaches and Insider Attacks in Cloud Computing Environments”. In Analyzing and Mitigating Security Risks in Cloud Computing (pp. 1-23). IGI Global.
9. Sharma, A., Pokharel, R. (2020). “User Behavior Analytics for Anomaly Detection Using LSTM-based Autoencoder”. ACM CCS Workshop on Insider Threats.

10. Aditham, S. (2017). Mitigation of Insider Attacks for Data Security in Distributed Computing Environments (Doctoral Thesis). University of South Florida.
11. Bhandari, D. (2023). “Insider Threat Detection using LSTM”. Journal of Security, Technology & Trust, 9(1).
12. Tao, X. (2025). “An insider threat detection method based on improved Test-Time Training (TTT)”. Computers & Security, 130, 104126.
13. Koli, L., Kalra, S., Thakur, R., Saifi, A., Singh, K. (2025). “AI-Driven IRM: Transforming insider risk management with adaptive scoring and LLM-based threat detection”. arXiv preprint arXiv:2505.03796.
14. Rastogi, N., Ma, Q. (2021). “DANTE: Predicting Insider Threat using LSTM on system logs”. arXiv preprint arXiv:2102.05600.
15. Huang, Z., Tang, X., Li, H., Cao, X., Cheng, J. (2024). “TabSec: A Collaborative Framework for Novel Insider Threat Detection”. arXiv preprint arXiv:2411.01779.
16. Yuan, S., Wu, X. (2021). “Deep learning for insider threat detection: Review, challenges and opportunities”. Computers & Security, 104, 102221.
17. Zeadally, S., Yu, B., Jeong, D.H., Liang, L. (2012). “Detecting insider threats: Solutions and trends”. Information Security Journal: A Global Perspective, 21(4), 183-192.
18. Le, D.C., Zincir-Heywood, N. (2021). “Anomaly detection for insider threats using unsupervised ensembles”. IEEE Transactions on Network and Service Management, 18(2), 1152-1164.
19. Roy, K.C., Chen, G. (2024). “GraphCH: A Deep Framework for Assessing Cyber-Human Aspects in Insider Threat Detection”. IEEE TDSC, 21(5), 4495-4509.
20. Nikiforova, O., Romanovs, A., Zabiniako, V., Kornienko, J. (2024). “Detecting and Identifying Insider Threats Based on Advanced Clustering Methods”. IEEE Access, 12, 30242-30253.

21. Vanitha, M., Navya Patel, M., Madhumitha, K., Sathvika, J. (2024). “Enhancing Insider Threat Detection in Cloud Environments Through Ensemble Learning”. International Journal of Communication Networks and Information Security (IJCNIS), 16(5), 638-647.
22. Sridevi, D., Kannagi, L., Vivekanandan, G., Revathi, S. (2023). “Detecting Insider Threats in Cybersecurity Using Machine Learning and Deep Learning Techniques”. In 2023 International Conference on Communication, Security and Artificial Intelligence (ICCSAI), IEEE.
23. Mittal, A., Garg, U. (2023). “Prediction and Detection of Insider Threat using Emails: A Comparison”. In Second International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT), IEEE.
24. Kumar, R. (2023). “Machine Learning Analysis of Data Granularity for Insider Threat Detection”. In 4th IEEE Global Conference for Advancement in Technology (GCAT), Bangalore, India.
25. Whitelaw, F., Riley, J., Elmrabit, N. (2024). “A Review of the Insider Threat, a Practitioner Perspective Within the U.K. Financial Services”. IEEE Access, 12, 34752-34768.
26. Kothari, N., Bhardwaj, C., Mishra, S., Satapathy, S.K., Mallick, P.K. (2024). “Towards Insider Threat Resilience: A Proposed Mitigation Model”. In 2024 International Conference on Emerging Systems and Intelligent Computing (ESIC), IEEE.
27. Eftimie, S., Moinescu, R., Răcuciu, C. (2020). “Insider Threat Detection Using Natural Language Processing and Personality Profiles”. In 13th International Conference on Communications (COMM), Bucharest, Romania.
28. Orizio, R., Vuppala, S., Basagiannis, S., Provan, G. (2020). “Towards an Explainable Approach for Insider Threat Detection: Constraint Network Learning”. In International Conference on Intelligent Data Science Technologies and Applications (IDSTA), Spain.

29. Alohaly, M., Balogun, O., Takabi, D. (2022). “Integrating cyber deception into attribute-based access control (ABAC) for insider threat detection”. IEEE Access, 10, 108965-108978.
30. S. Song, Gao, N., Zhang, Y., Ma, C. (2024). “BRITD: behavior rhythm insider threat detection with time awareness and user adaptation”. Cybersecurity, 7(1).
31. Oladimeji, T.O., Ayo, C.K., Adewumi, S.E. (2019). “Review on Insider Threat Detection Techniques”. Journal of Physics: Conference Series, 1299(1), 012046.
32. Wei, Z., Rauf, U., Mohsen, F. (2024). “E-Watcher: insider threat monitoring and detection for enhanced security”. Annals of Telecommunications, 79(11), 819-831.
33. Bin Sarhan, B., Altwaijry, N. (2022). “Insider Threat Detection Using Machine Learning Approach”. Applied Sciences, 13(1), 259.
34. Pantelidis, E., Bendiab, G., Shiaeles, S., Kolokotronis, N. (2021). “Insider Threat Detection using Deep Autoencoder and Variational Autoencoder Neural Networks”. In IEEE International Conference on Cyber Security and Resilience (CSR), IEEE.
35. Diop, A., Emad, N., Winter, T. (2020). “A Parallel and Scalable Framework for Insider Threat Detection”. In IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC), Pune, India.
36. S. Erola, Agrafiotis, I., Goldsmith, M., Creese, S. (2022). “Insider-threat detection: Lessons from deploying the CITD tool in three multinational organisations”. Journal of Information Security and Applications, 67, 103167.
37. Prasad, P.S.S., Nayak, S.K., Krishna, M.V. (2024). “Enhanced Insider Threat Detection Through Machine Learning Approach With Imbalanced Data Resolution”. Journal of Theoretical and Applied Information Technology, 102(3).