Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки SIEM для протидії атакам у комп’ютерній мережі”

Виконав здобувач вищої освіти
IV курсу, групи КБ-21
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Козлов Я.О.
«____» _____ 2025 р.

Керівник проекту
доктор технічних наук, професор
_____ Смірнов О.А.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Козлову Яну Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки SIEM для протидії атакам у комп’ютерній мережі*

2. Керівник роботи *Смірнов Олексій Анатолійович, доктор техн. наук, професор*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 57-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки SIEM для протидії атакам у комп’ютерній мережі*

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

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*3 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Смірнов О.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Козлов Я.О.
(прізвище та ініціали)

АНОТАЦІЯ

Козлов Я.О. Програмне забезпечення системи кібербезпеки SIEM для протидії атакам у комп'ютерній мережі. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення системи кібербезпеки SIEM для протидії атакам у комп'ютерній мережі.

Метою розробки є програмне забезпечення системи кібербезпеки SIEM для протидії атакам у комп'ютерній мережі.

Результат роботи – програмна реалізація системи кібербезпеки SIEM для протидії атакам у комп'ютерній мережі.

В процесі роботи над програмною моделлю виконано аналіз існуючих програмних засобів та методів протидії кібератакам. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програмний комплекс може використовуватися на ПЕОМ з ОС Windows 10/11 та ОС Linux.

Пакет програмного забезпечення складається з програмних компонентів з відкритим кодом. Наведено сценарії встановлення та налаштування компонентів на ОС Windows 10/11 та ОС Linux. Також наведено програмну реалізацію алгоритму виявлення аномалій RCF.

Ключові слова: кібербезпека, SIEM, машинне навчання

ABSTRACT

Kozlov Ya.O. Software of the SIEM cybersecurity system for countering attacks in a computer network. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this graduation thesis for the first (bachelor) level of higher education, the software of the SIEM cybersecurity system was developed, which is intended to counter attacks in a computer network.

The purpose of the development is the software of the SIEM cybersecurity system to counter attacks in a computer network.

The result of the work is the software implementation of a cybersecurity system SIEM to counter attacks in a computer network.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The software package can be used on a PC running Windows 10/11 OS or Linux.

The software package consists of open-source software components. Scripts for installing and configuring the components on Windows 10/11 and Linux are provided. The software implementation of the RCF anomaly detection algorithm is also presented.

Keywords: cybersecurity, SIEM, machine learning

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ІТ	–	інформаційні технології
ПЗ	–	програмне забезпечення
ІКС	–	інформаційно-комунікаційні системи
ШІ	–	штучний інтелект
AI	–	Artificial Intelligence (штучний інтелект)
SIEM	–	Security Information and Event Management (система управління безпековою інформацією та подіями)
False positive	–	Хибне спрацьовування системи (помилкова тривога)
ML	–	Machine Learning (машинне навчання)
DL	–	Deep Learning (глибинне навчання)
DDoS	–	Distributed Denial of Service (розподілена атака відмови в обслуговуванні)
IDS	–	Intrusion Detection System (система виявлення вторгнень)
IPS	–	Intrusion Prevention System (система запобігання вторгненням)
MITRE ATT&CK	–	База знань тактик і технік кіберзагроз
UEBA	–	User and Entity Behavior Analytics (аналіз поведінки користувачів та об’єктів)
БД	–	база даних
NoSQL	–	Not only SQL, нереляційна база даних
SQL	–	Structured Query Language (мова запитів до реляційних баз даних)
SOC	–	Security Operations Center (центр кібербезпеки)
EDR	–	Endpoint Detection and Response (засоби виявлення і реагування на кінцевих пристроях)

XDR	–	Extended Detection and Response (розширені засоби виявлення і реагування)
SOAR	–	Security Orchestration, Automation, and Response (автоматизація та оркестрація безпеки)
NTA	–	Network Traffic Analysis (аналіз мережевого трафіку)
TTP	–	Tactics, Techniques, and Procedures (тактики, техніки та процедури кіберзагроз)
Дашборд	–	Панель моніторингу (інтерфейс звітності)
HIDS	–	Host-based Intrusion Detection System (система виявлення вторгнень на вузлі)
REST	–	Representational State Transfer (архітектурний стил API)
API	–	Application Programming Interface (інтерфейс програмування додатків)
RESTful API	–	API, що реалізує принципи REST
FIM	–	File Integrity Monitoring (моніторинг цілісності файлів)
Rootkit	–	Приховане шкідливе ПЗ
CVE	–	Common Vulnerabilities and Exposures (база вразливостей)
PCI DSS	–	Payment Card Industry Data Security Standard (стандарт безпеки платіжних карток)
HIPAA	–	Health Insurance Portability and Accountability Act (закон про медичну інформацію в США)
GDPR	–	General Data Protection Regulation (загальний регламент захисту даних ЄС)
SOC2	–	Стандарт аудиту безпеки інформаційних систем
ISO	–	International Organization for Standardization (міднародна організація стандартизації)
IEC	–	International Electrotechnical Commission (міднародна електротехнічна комісія)

SaaS	–	Software as a Service (ПЗ як послуга)
KQL	–	Kusto Query Language (мова запитів до деяких NoSQL БД)
ETL	–	Extract, Transform, Load (витяг, трансформація та завантаження даних)
LDAP	–	Lightweight Directory Access Protocol (протокол доступу до каталогів)
SAML	–	Security Assertion Markup Language (мова безпечної автентифікації)
OpenID	–	Відкрита система автентифікації
ОС	–	операційна система
JSON	–	JavaScript Object Notation (формат обміну даними)
RCF	–	Random Cut Forest (алгоритм «лісу випадкових розрізів)
Плагін	–	Розширення функцій програми
Webhook	–	Механізм оповіщення про події шляхом HTTP запиту
HTTP	–	HyperText Transfer Protocol (протокол передачі гіпертексту)
TLS	–	Transport Layer Security (протокол шифрування даних у мережі)
CA	–	Certificate Authoirty (центр сертифікації)
Triage	–	швидке сорутвання інцидентів за пріоритетом в SOC
Ad-hoc	–	разове, незаплановане завчасно решіення або дія
Zero-day	–	невідома вразливість без виправлення

ВСТУП

Актуальність теми. SIEM (Security Information and Event Management) є важливим елементом системи кібербезпеки, який дозволяє збирати, аналізувати та корелювати інформацію про події безпеки з різних джерел у реальному часі. SIEM-системи забезпечують централізований моніторинг, що дозволяє відстежувати та виявляти потенційні загрози або аномалії в роботі мережі, серверів, баз даних та інших компонентів інформаційної інфраструктури організаціїї. Оскільки кіберзагрози постійно змінюються та стають дедалі складнішими, SIEM-системи відіграють ключову роль у зменшенні ризиків для організацій, автоматизуючи процеси виявлення та реагування на інциденти безпеки.

Загрози можуть бути різноманітними: від відомих атак, як DDoS, до нових, невідомих раніше, які вимагають більш гнучкого підходу до їх виявлення. Традиційно SIEM-системи працюють на основі заздалегідь визначених правил, які виявляють конкретні патерни чи аномалії. Однак цей підхід має обмеження, адже складні атаки або нові техніки можуть не підпадати під встановлені шаблони. В таких випадках має сенс скористатися можливостями штучного інтелекту.

Штучний інтелект (ШІ), зокрема машинне навчання (Machine Learning, ML) та глибинне навчання (Deep Learning, DL), має значний потенціал для підвищення ефективності SIEM-систем. Інтеграція ШІ у такі системи має наступні переваги:

– Виявлення нових загроз. Традиційні системи часто обмежені правилами, що базуються на історичних даних. ШІ здатен виявляти нові патерни та відхилення від звичайної поведінки, що дає змогу виявляти складні атаки що не підпадають під заздалегідь визначені критерії.

– Покращення кореляції подій. Великим викликом для SIEM є правильне

корелювання різноманітних подій з різних джерел, як от мережевих пристроїв, кінцевих точок та інших. Виявляючи більш складні зв'язки між подіями, ШІ може виявляти більш складні атаки, що складаються с кількох етапів або використовують методи приховування.

– Зменшення помилкових спрацьовувань (false positives). Оскільки традиційні SIEM-системи часто працюють на основі встановлених правил, вони можуть генерувати велику кількість помилкових тривог, що призводить до «втоми від тривог» (alert fatigue) у команди реагування та знижує їхню ефективність. ШІ може знизити кількість помилкових спрацьовувань, оскільки здатен точніше виявляти аномалії, враховуючи контекст та історію подій, та таким чином дозволяє зосереджуватись на більш критичних інцидентах.

– Адаптивність та навчання. Завдяки здатності самонавчатися, ШІ може адаптуватися до змін у поведінці системи, користувачів та навіть методів атаки. Це дає змогу системі безпеки постійно вдосконалюватися та підвищувати ефективність виявлення аномалій у нових умовах, що критично важливо в умовах динамічного оточення та загроз.

– Аналіз великих обсягів даних у реальному часі. SIEM-системи збирають величезну кількість даних з різних джерел, що ускладнює їхню обробку та аналіз. ШІ може ефективно обробляти великі обсяги даних та знаходити патерни чи аномалії, які важко виявити вручну. Це дає можливість швидше реагувати на інциденти, які потребують додаткового аналізу, що надзвичайно важливо для аналітиків безпеки .

Мета й завдання дослідження. Метою роботи є реалізація та впроваждження програмного забезпечення системи кібербезпеки SIEM для протидії атакам у комп'ютерній мережі.

Для досягнення поставленої мети у роботі визначено наступні завдання дослідження:

– аналітичний огляд сучасних SIEM-систем, що використовують методи штучного інтелекту, та підходів до використання цих методів в архітектурі SIEM;

– дослідження та адаптація конкретного методу ШІ, придатного для підвищення точності виявлення аномалій та зниження рівня хибнопозитивних спрацювань SIEM-систем;

– розробка та впровадження архітектури SIEM-системи для протидії атакам у комп’ютерній мережі.

Практична цінність отриманих результатів полягає в тому, що реалізовані методи дозволять підвищити ефективність SIEM-систем за рахунок зменшення кількості помилкових виявлень загроз та покращеної якості кореляції подій, що в свою чергу дає змогу виявляти більш складні загрози.

Таким чином, реалізація програмного забезпечення системи кібербезпеки SIEM для протидії атакам у комп’ютерній мережі є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для вчасного виявлення та протидії кіберінцидентам та кібератакам в комп'ютерних мережах та ІКС організацій, в тому числі за допомогою методів штучного інтелекту. Метою бакалаврського проекту є створення програмного зазпечення системи кібербезпеки SIEM для протидії атакам, а також підвищення ефективності виявлення потенційних загроз інформаційній безпеці шляхом автоматизованої обробки подій та застосування інтелектуального аналізу даних за допомогою методів штучного інтелекту, зокрема машинного навчання. Система забезпечує обробку великого обсягу подій у режимі, наближеному до реального часу, з можливістю адаптації до змін у поведінці мережі. Для досягнення зазначених цілей у межах бакалаврського проєкту були вирішені наступні завдання:

– проведено аналіз існуючих архітектур SIEM-систем, принципів їхної побудови та основних функціональних компонентів;

– досліджено можливості застосування методів штучного інтелекту в SIEM-системах, зокрема в кореляції подій інформаційної безпеки для виявлення відхилень від типових шаблонів поведінки;

– розроблено загальну архітектуру SIEM-систем з урахуванням інтеграції аналітичного модуля для виявлення аномалій;

– розглянуто та порівняно кілька алгоритмів виявлення аномалій, що застосовуються у сфері інформаційної безпеки, з метою вибору оптимального для реалізації в рамках розробленох архітектури;

– підібрано відповідні компоненти SIEM-системи, які дозволяють інтеграцію із зовнішніми модулями обробки даних та підтримують масштабування;

– реалізовано комплекс програмних компонентів, який включає модулі збору, зберігання, обробки та аналітики подій інформаційної безпеки, інтегрований з алгоритмом виявлення аномалій;

– розроблено механізми передачі даних між компонентами системи, забезпечено їхню сумісність та взаємодію в єдиному програмному середовищі;

– створено сценарії автоматизованого розгортання та налаштування системи, що значно спрощують її впровадження та адаптацію в реальних умовах експлуатації.

Результатом роботи є прототип SIEM-системи, орієнтований на виявлення нетипової поведінки у корпоративному середовищі, який може бути основою для подальшого розвитку у напрямку автоматичного реагування на інциденти.

1.2 Область застосування

Областю застосування розробленої системи є корпоративні мережі. Під корпоративнох мережею мається на увазі інфраструктура підприємства, що забезпечує взаємодію внутрішніх підрозділів, серверів, робочих станцій та інших IT-ресурсів для підтримки бізнес-процесів організації. Основною особливістю таких мереж є обмежений доступ: користувачами корпоративної мережі виступають виключно співробітники підприємства, а сама мережа зазвичай не надає послуг стороннім огранціаціям чи користувачам.

Корпоративні мережі можуть бути різного масштабу – від мереж окремого відділу до розподілених інфраструктур великого підприємства з офісами в різних регіонах. Такі мережі зазвичай є територіально розподіленими та об’єднують декілька майданчиків або філій, використовуючи загальні політики безпеки, маршрутизацію, централізоване керування та моніторинг.

У сучасних корпоративних мережах функціонує велика кількість інфомаційних сервісів, таких як файлові сховища, внутрішні вебсайти, поштові сервери, VPN-тунелі для безпечного віддаленого доступу, тощо. Для

забезпечення безперервності роботи таких сервісів необхідно впроваджувати інструменти, що забезпечують моніторинг, виявлення, реєстрацію та аналіз подій інформаційної безпеки.

Розроблена SIEM-системи призначена для збору, обробки та аналізу подій з різних корпоративної мережі інформаційно-комунікаційних систем (журнали доступу, мережевий трафік, дії користувачів тощо), а також для виявлення потенційно небезпечної або аномальної активності. Особливістю системи є використання методів штучного інтелекту, що дозволяє ефективно аналізувати поведінкові патерни та виявляти інциденти без необхідності створення великої кількості правил.

Система може бути розгорнута як на фізичній інфраструктурі підприємства (on-premise), так і у віртуалізованому або змарному середовищі – залежно від вимог, бюджету та обранох архітектури. Хоча хмарне середовище не є обов'язковим для функціонування системи, її компонентна структура дозволяє гнучкоадаптуватися до різних варіантів розгортання, включаючи гібридні сценарії.

Таким чином, розроблення програмного забезпечення системи кібербезпеки SIEM для протидії атакам у комп'ютерній мережі є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

У сучасному світі інформаційної безпеки SIEM-системи (Security Information and Event Management) є навід'ємною складовою захисту інформаційних ресурсів організацій та ядром, навколо якого будується інфраструктура центрів кібербезпеки (SOC). Вони поєднують функції збору, нормалізації централізованого зберігання, кореляції та аналізу подій безпеки з різноманітних джерел, як то мережеве обладнання, сервери, робочі станції та прикладне програмне забезпечення.

Однією з ключових можливостей SIEM-систем є кореляція подій – виявлення зв'язків між інцидентами, які, на перший погляд, не є суттєвими, для побудови загальної картини загроз. Сучасні системи також можуть інтегрувати інструменти оцінки ризиків, реагування на інциденти та автоматизації дій відповідно до політик безпеки.

У зв'язку зі зростанням обсягів даних та складністю атак, все більше рішень у цій сфері впроваджують методи штучного інтелекту та машинного навчання для підвищення точності виявлення загроз і зменшення кількості хибнопозитивних спрацювань (false positives).

Splunk Enterprise Security

Splunk Enterprise Security – потужна SIEM-платформа, що забезпечує збір, індексацію, моніторинг та аналіз великих обсягів машинних даних у реальному часі. Splunk активно використовується у великих та середніх організаціях для виявлення загроз, реагування на інциденти та забезпечення відповідності вимогам безпеки.

Особливістю Splunk є орієнтація на роботу з **неструктурованими даними** з різноманітних джерел без необхідності їхнього попереднього форматування.

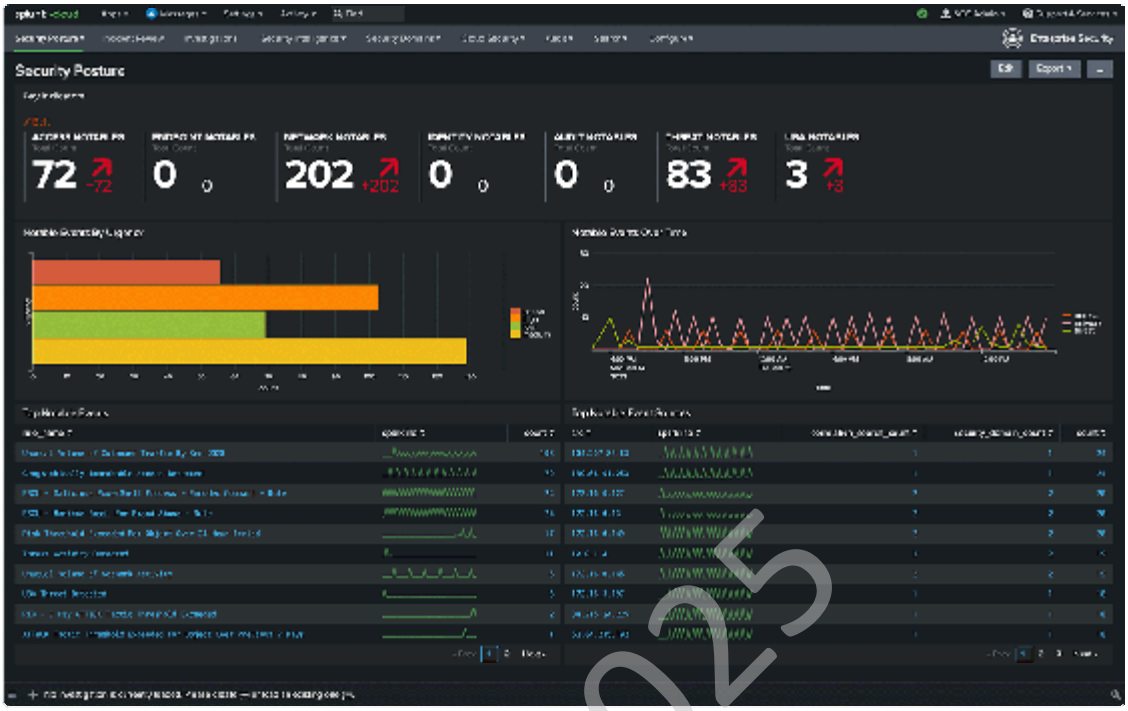

Рисунок 2.1 – Панель адміністратора Splunk Security Enterprise

Splunk має модульну архітектуру, яка складається з декількох основних елементів:

– **Forwarders** – агенти, що встановлюються на джерела даних (сервери, пристрої, додатки) для збору і передачі даних у Splunk-середовище. Forwarders бувають двох типів: Universal (універсальні) там Heavy (з попередньою обробкою даних).

– **Indexers** – сервери, що здійснюють прийом даних, їхню обробку, індексацію та зберігання у внутрішньому форматі для подальшого швидкого пошуку та аналізу.

– **Search Head** – сервер або кластер серверів, через які користувачі здійснюють пошук, створення дашбордів, звітів та аналітики. Саме тут виконується логіка обробки запитів та візуалізація даних.

– **Deployment Server** – сервер для централізованого керування розгортанням та оновленням конфігурацій агентів.

– **Cluster Mater / Indexer Cluster** – використовується для високодоступних (HA, High Availability) та масштабованих рішень, де дані реплікуються між кількома індексаторами.

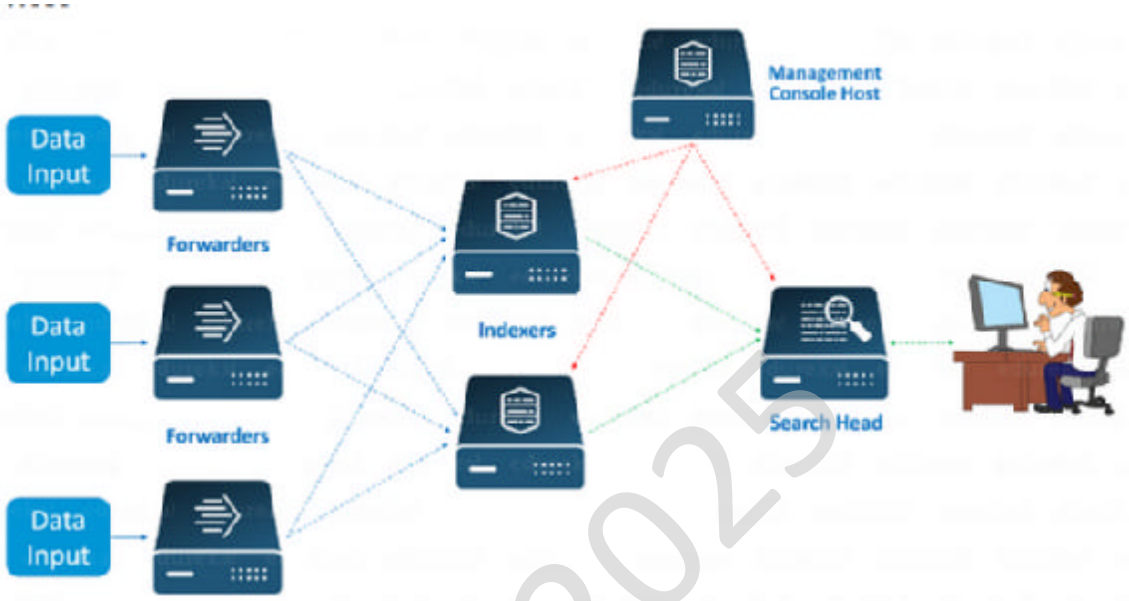

Рисунок 2.2 – Розподілена архітектура Splunk

Splunk працює зі спеціалізованою мовою запитів SPL (Splunk Processing Language), що використовується аналітиками для аналізу, агрегації, кореляції та візуалізації даних. Splunk також надає можливість створення багатовимірних моделей даних для полегшеного пошуку та аналітики без написання складних запитів.

Як і більшість сучасних рішень у різних сферах, Splunk маж відкритий REST API інтерфейс для інтеграції із зовнішніми системами.

Splunk активно інтегрує штучний інтелект і машинне навчання у свої рішення для виявлення складних кіберзагроз:

– **Machine Learning Toolkit (MLTK)**. Надає можливість створювати, тренувати, тестувати та застосовувати моделі машинного навчання безпосередньо в Splunk; підтримуються типові алгоритми для класифікації, кластеризації, регресії та виявлення аномалій. Можливе автоматичне навчання на базі

історичних даних для прогнозування майбутніх загроз або змін у поведінці мережі.

– **Splunk Assistive Intelligence**. Використовує попередньо навчені моделі для автоматичного виявлення підозрілих патернів без необхідності глибокого налаштування; інтеграція можливостей UEBA (User and Entitiy Behavior Analytics), які автоматично аналізують поведінку користувачів і систем для виявлення відхилень від нормальної діяльності.

– **Splunk SOAR (раніше Phantom)**. Інструмент для автоматизації, оркестрації та реагування, який включає функції аналізу загроз за допомогою ШІ та автоматичного прийняття рішень на основі попередньо визначених політик.

Попри усе, Splunk має свої недоліки. Вартість ліцензій дуже висока через те, що залежить від обсягу даних, які індексуються щодня. Це особливо відчутно для великих організацій або компаній з високим рівнем логування. Окрім того, Splunk вимогливий до ресурсів через свою архітектуру, а також може бути складним у налаштуванні та потребувати кваліфікованих фахівців для створення складних запитів, налаштування машинного навчання або безпеки самої системи тощо.

IBM QRadar

IBM QRadar – одна з провідних SIEM-систем, яка орієнтована на централізований моніторинг подій безпеки, виявлення інцидентів та управління ними в режимі реального часу. Рішення від IBM особливо популярне серед великих підприємств завдяки глибокому аналізу мережевих потоків, подій і даних з багатьох джерел. QRadar об’єднує в собі функції SIEM (Security Information and Event Management, «керування безпековою інформацією та подіями»), NTA (Network Traffic Analysis, «аналіз мережевого трафіку») та UEBA (User and Entity Behavior Analytics, «поведінкова аналітика користувачів та сутностей»).

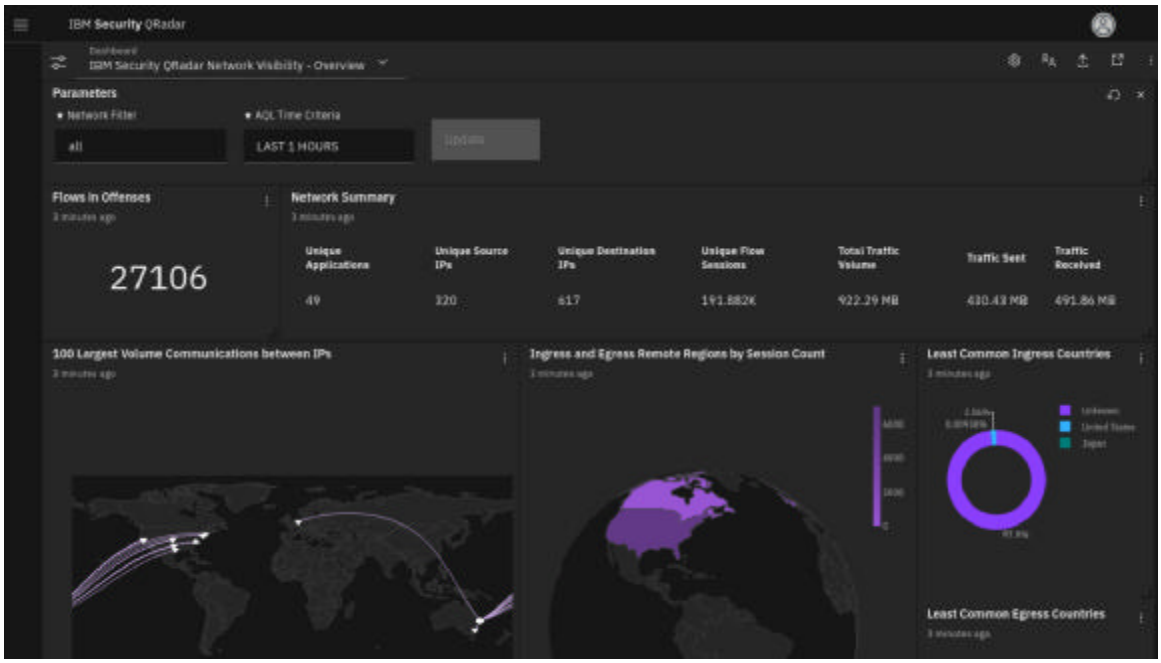

Рисунок 2.3 – Панель адміністратора IBM QRadar SIEM

IBM QRadar складається з кількох основних компонентів:

– Event Collectors – збирають події з різних джерел, як то логи, системм безпеки, пристрої тощо.

– Flow Collectors – збирають мережеві потоки NetFlow, sFlow, IPFIX для аналізу мережевої активності.

– Event Processors – нормалізують, зберігають та корелюють події.

– Console – інтерфейс користувача для моніторингу подій, створення правил кореляції, роботи з інцидентами та звітами.

– Data Nodes – допоміжні вузли для масштабування обсягу зберігання даних.

QRadar може здійснювати високоточний аналіз подій та мережевих потоків у реальному часі, що дозволяє оперативно реагувати на інциденти безпеки та мінімізувати час між виявленням загрози і початком розслідування відповідного інциденту. Однією з ключових функцій системи є автоматичка нормалізація логів – QRadar приводить дані з різнорідних джерел (мережевих екранів, IDS/IPS, серверівб додатків, баз даних тощо) до уніфікованого формату, що значно полегшує аналітику та подальшу кореляцію подій.

Платоформа підтримує широке коло вбудованих інтеграцій з мережевим обладнанням, системами захисту, сервісами та платформами, що роблять її сумісною з більшістю інфраструктур корпоративного рівня. Аналітики мають доступ до централізованої панелі управління, яка дозволяє зручно моніторити події, створювати користувацькі правила кореляції, запускати розслідування інцидентів, формувати звіти та контролювати загальний стан інформаційної безпеки організації. Додатковою перевагою є підтримка MITRE ATT&CK – фреймворку, який допомагає класифікувати події відповідно до тактик і технік зловмисників, тим самим полегшуючи виявлення складних і багатоетапних атак.

Особливістю QRadar є впровадження елементів штучного інтелекту та машинного навчання через окремі модулі. Модуль “QRadar Advisor with Watson” використовує когнітивні можливості платформи IBM Watson, яка здатна аналізувати дані з відкритих джерел, технічнох документації, баз знань та внутрішніх журналів організації. Watson надає аналітикам висновки щодо природи інциденту, потенційних векторів атаки, контексту подій і можливих дій для реагування.

Інший важливий компонент – “QRadar User and Entity Behavior Analyics (UEBA)” – аналізує поведінку користувачів і пристроїв за допомогою алгоритмів машинного навчання. Він виявляє нетипові дії, які можуть вказувати на компрометацію облікових записів, інсайдерські загрози або автоматизовану активність. UEBA дозволяє створювати профілі нормальної поведінки користувачів, а в разі відхилення від них – автоматично генерувати інциденти для подальшого аналізу.

Однак, навіть будучи висококласним корпоративним рішенням для SIEM, IBM QRadar залишається менш придатним для середніх та малих підприємств через високу вартість ліцензування та складність налаштування і обслуговування.

Elastic SIEM

Elastic SIEM (на базі Elastic Stack) – потужний відкритий стек інструментів для збору, обробки, зберігання та візуалізації логів і даних безпеки. Elastic Stack,

інакше відомий як "ELK Stack» (абревіатура від перших літер компонентів), складається з трьох основних складових: **Elasticsearch** – пошуковий і аналітичний рушій: **Logstash** – система обробки і транспортування даних; **Kibana** – платформа для візуалізації та аналізу даних. Іноді до стеку відносять ще **Beats** – невеликі агенти для збору даних із різних джерел, тому часто можна зустріти назву “**Elastic Stack**”. Спочатку ELK не створювався як SIEM, але завдяки відкритості та гнучкості став дуже популярним для побудови інших SIEM-рішень.

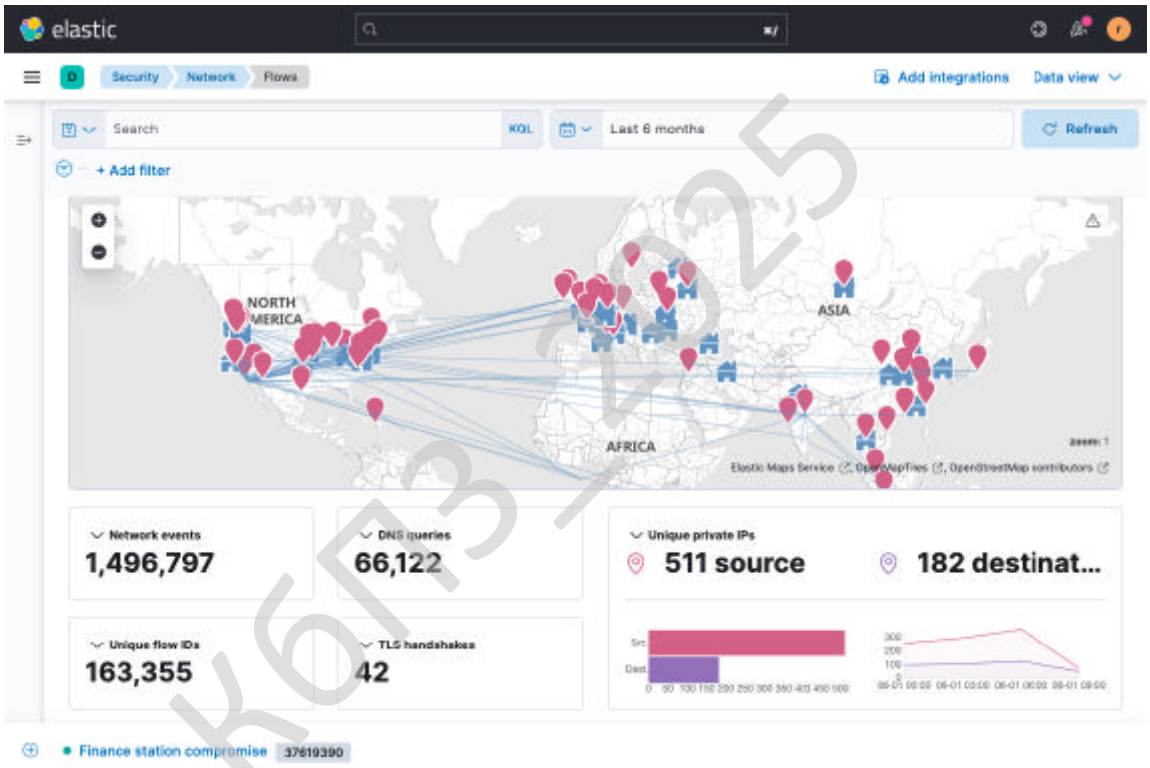

Рисунок 2.4 –Панель візуалізації Elastic SIEM

Кожний з компонентів виконує свою окрему роль, забезпечуючи гнучкість, масштабованість і високу продуктивність системи.

Beats – це сімейство легких агентів, які встановлюються безпосередньо на джерела даних (сервери, кінцеві пристрої, тощо) і призначені для збору специфічної інформації. Наприклад, Filebeat передає логи з файлів журналів, Metricbeat збирає метрики з систем і сервісів, Packetbrat аналізує мережеві пакети,

а Winlogbeat працює з журналами подій Windows. Beats мають мінімальне споживання ресурсів і є зручним для масового розгортання.

Logstash виступає як потужний механізм обробки та маршрутизації даних. Він дозволяє агрегувати потокт даних з різних джерел, фільтрувати, перетворювати, збагачувати та нормалізувати їх перед збереженням. Завдяки великій кількості доступних плагінів Logstash підтримує інтеграцію з десятками форматів логів, систем і протоколів, а також може слугувати буфером перед БД Elasticsearch.

Elasticsearch – це високопродуктивна розподілена NoSQL база даних, побудована на основі пошукової бібліотеки Apache Lucene. Вона оптимізована для швидкого повнотекствого пошуку та аналітики великих обсягів даних. Elasticsearch забезпечує горизонтальне масштабування, що дозволяє обробляти мільйони подій за секунду з низькою затримкою.

Kibana – надає зручний веб-інтерфейс для візуалізації зібраних та проаналізованих даних. З його допомогою користувачі можуть створювати інтерактивні панелі, виконувати пошук за логами, будувати графіки та діаграми, а також проводити глибокий аналіз інцидентів безпеки. Kibana також включає можливості для свторення алертів, фільтрів безпеки і взаємодії з модулями машинного навчання (у розширених конфігураціях).

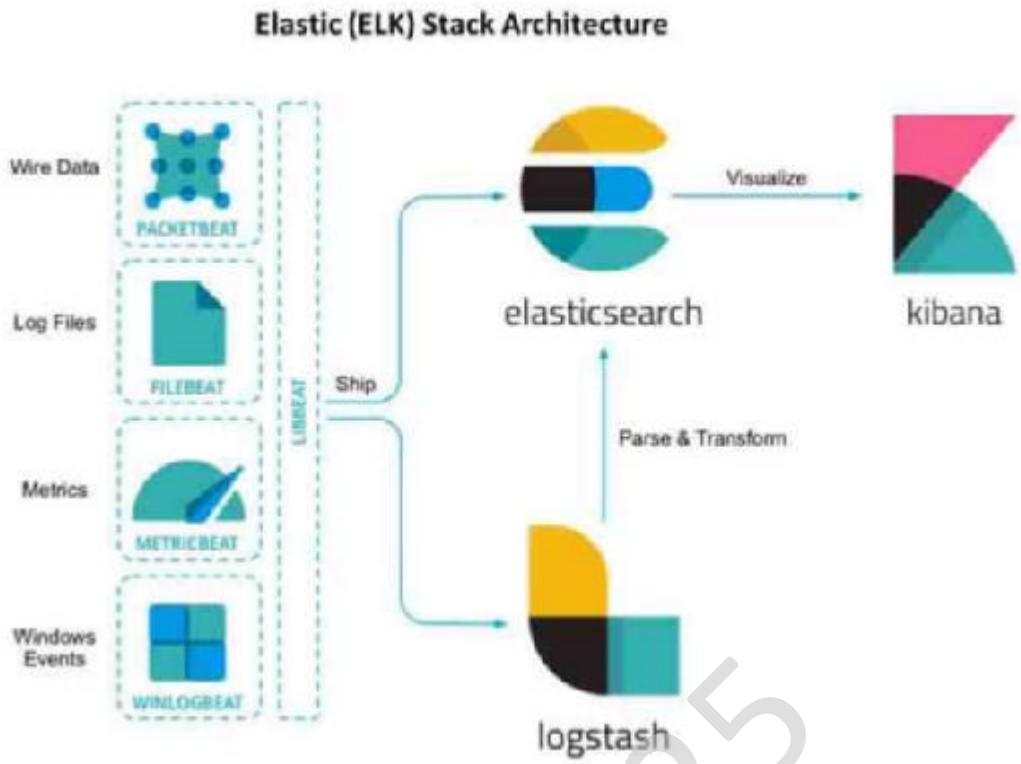

Рисунок 2.5 – Архітектура Elastic Stack

Окрім основних компонентів перерахованих вище, Elastic Stack пропонує низку доповнень, що розширюють функціональність системи в контексті забезпечення безпеки. Вони дозволяють реалізувати повноцінне SIEM-рішення з централізованим управлінням, аналітикою та автоматизованим виявленням загроз.

Elastic Security (раніше відомий як Elastic SIEM) –це спеціалізований модуль у Kibana, який дозволяє побудувати SIEM-систему на основі зібраних логів та подій безпеки. Він забезпечує візуалізацію активностей, механізми кореляції подій, створення правил виявлення загроз, інтеграцію з MITRE ATT&CK та підтримку реакції на інциденти. Інтерфейс Elastic Security включає окремі вкладки для перегляду сигналів (alerts), аналізу сесій, зведеної активності користувачів і хочтів, що суттєво полегшує роботу аналітиків SOC.

Fleet і Elastic Agent – це нова архітектура для централізованого розгортання, оновлення та управління агентами. Замість окремих Beats для кожного типу даних, Elastic Agent є універсальним агентом, який може

виконувати функції Filebeat, Metricbeat, Endpoint Security тощо одночасно. Fleet, у свою чергу, забезпечує централізовану консоль для керування великою кількістю агентів, налаштування політик збору даних, застосування конфігурацій і моніторингу стану кожного стану кожного агента в реальному часі. Це значно спрощує масштабування та підтримку системи.

Machine Learning – це комерційне (платне) розширення Elastic Stack, яке дозволяє реалізувати автоматизоване виявлення аномалій у потоках подій. Система машинного навчання створює профілі нормальної поведінки систем, користувачів чи мережі, а потім виявляє відхилення від цих профілів у реальному часі. Це особливо корисно для виявлення складних та раніше невідомих атак, які неможливо ідентифікувати за допомогою традиційних сигнатур або правил. Інтеграція ML із Elastic Security дозволяє генерувати сигнали на основі аномалій і включати їх у загальний контекст безпекової аналітики.

Elastic Stack у поєднанні з модулем Elastic Security забезпечує широкий спектр можливостей для створення потужної SIEM-системи. Рішення дозволяє централізовано збирати та зберігати логи з серверів, мережевого обладнання, програмного забезпечення та засобів захисту. Завдяки Logstash дані проходять попередню обробку, трансформацію та нормалізацію перед збереженням. Elasticsearch відповідає за високошвидкісний пошук і аналітику великих обсягів інформації, а Kibana надає інструменти для візуалізції даних та подій безпеки завдяки інтерактивним панелям.

Окрім основних функцій, Elastic Stack легко інтегрується з іншими системами захисту, зокрема SOAR, EDR, XDR, IDS/IPS та мережевими екранами. Elastic Security підтримує інтеграцію з фреймворком MITRE ATT&CK, що дає змогу виявляти та відстежувати тактики, техніки та процедури (TTP) зловмисників. Kibana надає можливість створювати та керувати оповіщеннями (alert'ами), які спрацьовують при виявленні ознак відповідно до заданих правил. Крім того, платформа підтримує розширене розслідування інцидентів,

дозволяючи аналітикам об’єднувати журнали подій, дані з мережевого трафіку та кінцевих пристроїв в єдиний контекст.

Elastic Stack надає можливість інтеграції машинного навчання (ML), що доступна в межах платнох ліцензії. Ці функції дозволяють суттєво посилити виявлення загроз завдяки автоматичному аналізу даних. Зокрема, система може:

– виявляти аномалії в мережевому трафіку, поведінці користувачів або систем, що сигналізує про потенційні інциденти безпеки;

– створювати поведінкові моделі, наприклад, типову кількість автентифікацій у конкретний проміжок часу, щоб помічати підозрілу активність;

– прогнозувати тренди, зокрема навантаження чи відхилення, заздалегідь попереджаючи про можливі проблеми;

– використовувати Elastic Machine Learning для розробки власих моделей, що враховують особливості інформаційно-комунікаційного середовища організації.

Варто зазначити, що ML-функціональність не замінює класичні підходи жо виявлення загрози, а доповнює їх, дозволяючи ефективніше виявляти нові або ще не класифіковані атаки.

ELK Stack є надзвичайно потужною та гнучкою платформою для побудови SIEM-системи, особливо для організацій, які прагнуть мати максимальний контроль над своєю інфраструктурою безпеки. Завдяки відкритості та можливості налаштування, ELK Stack дозволяє створити рішення, яке ідеально підходить під конкретні потреби. Водночас важливо враховувати, що розгортання системи може бути складним, вимагати значних ресурсів і технічної експертизи. Крім того, частина критично важливих функцій, таких як машинне навчання, доступна лише за платною підпискою, що може бути обмеженням для невеликих підприємств та організацій.

Wazuh

Wazuh – безкоштовне SIEM-рішення з відкритим кодом, яке поєднує збір і обробку логів, функціональність HIDS (Host-based Intrusion Detection System,

«сисетма виявлення вторгнення на рівні хоста»), моніторинг цілісності файлів і відповідність вимогам безпеки. Wazuh розвивався як розширення до іншої HIDS системи, OSSEC, але згодом став самостійним проєктом із власною архітектурою та великою спільнотою. Побудований на основі компонентів Elastic Stack, тож добре інтегрується з існуючим стеком.

Wazuh складається із наступних компонентів:

– **Wazuh Agent**: встановлюється безпосередньо на кінцеві точки (сервери, робочі станції) та виконує низку критичних функцій. Агент відповідає за збір логів, моніторинг процесів, контроль зміну у файлах, перевірку цілісності системних компонентів та відповідність політикам безпеки. Агент підтримує роботу з різними операційними системами (Windows, Linux, macOS), що дозволяє охопити різноманітну IT-архітектуру.

– **Wazuh Manager**: серверна частина, яка приймає та обробляє дані, надіслані агентами. Менеджер виконує кореляцію подій, застосовує правила виявлення загроз, аналізує логи, контролює відповідність вимогам стандартів (наприклад, PCI DSS, GDPR, HIPAA тощо) і керує політиками безпеки для агентів. Він також формує сповіщення на основі виявлених подій.

– **Elasticsearch**: висупає сховищем для подій, зібраних системою. Він забезпечує можливість високошвидкісного пошуку, фільтрації та аналітики подій безпеки в реальному часі. Elasticsearch може бути інтегрований локально або використовуватись в якості сервіса у хмарі (Elastic Cloud).

– **Kibana**: слугує візуальним інтерфейсом користувача для взаємодії з даними. Через Kibana можна створювати власні панелі, переглядати сигнали тривоги, аналізувати тенденції та проводити розслідування інцидентів. Wazuh надає власні розширення до Kibana, які дають спеціалізовані панелі для аналізу подій безпеки.

– **Wazuh API**: RESTful API, який відкриває можливості для інтеграції Wazuh з іншими зовнішніми системами (наприклад, з інструментами автоматизації або зовнішніми SIEM-рішеннями). Через API також можна

реалізувати автоматичне управління агентами, отримання звітів, зміну конфігурацій та виконання сценаріїв реагування.

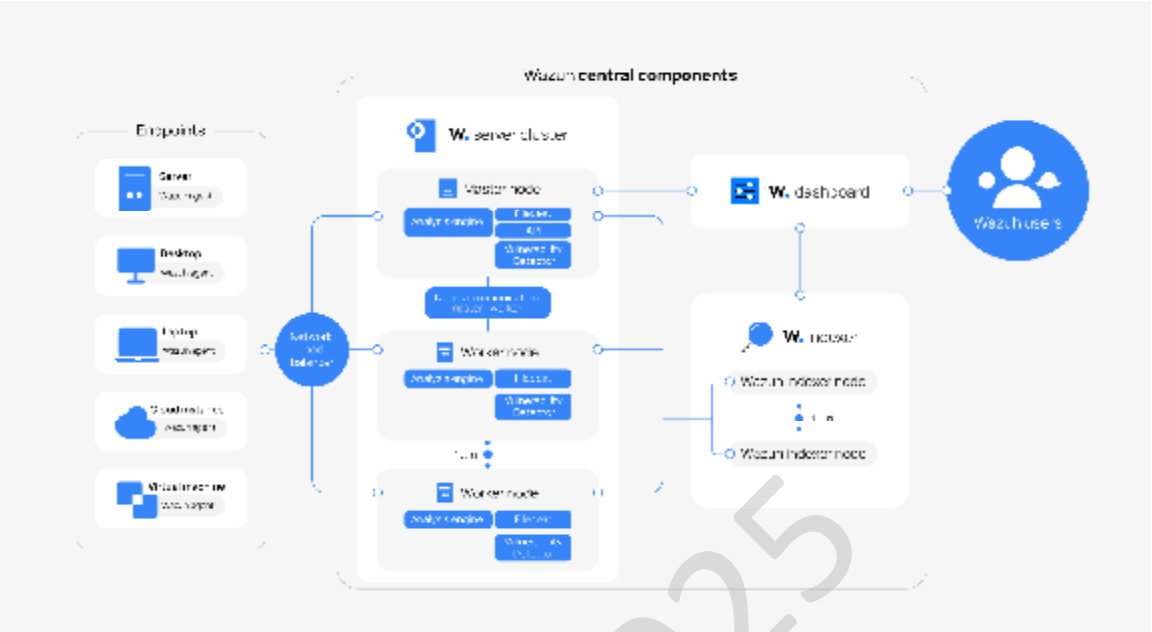

Рисунок 2.6 – Архітектура Wazuh

Ця модульна архітектура робить Wazuh масштабованим рішенням, яке підходить як для малих організацій, так і для великих підприємств. Його відкритий код, активна спільнота та регулярні оновлення роблять систему конкурентною альтернативою багатьом комерційним SIEM-продуктам.

Додатково Wazuh підтримує низку функціональних розширень, які значно покращують можливості системи в сфері кібербезпеки:

– **File Integrity Monitoring** (FIM, «моніторинг цілісності файлів») – відстежує зміни у критичних системних файлах та конфігураціях, що дозволяє оперативно виявляти підозрілу активність або несанкціоновані зміни.

– **Rootkit Detection** («виявлення руткітів») – аналізує систему на наявність руткітів, які можуть приховувати шкідливі процеси або файли, що робить цю функцію важливою для виявлення глибоко інтегрованих загроз.

– **Vulnerability Detection** («виявлення вразлвостей») – виконує сканування систем та програмного забезпечення на наявність відомих вразливостей,

використовуючи інтеграцію з публічними базами, зокрема CVE (Common Vulnerabilities and Exposures).

– **Active Response** («активне реагування на інциденти») – реалізує автоматичні дії у відповідь на виявлені інциденти безпеки, наприклад, блокування IP-адреси атакуючого або завершення шкідливого процесу на кінцевій точці.

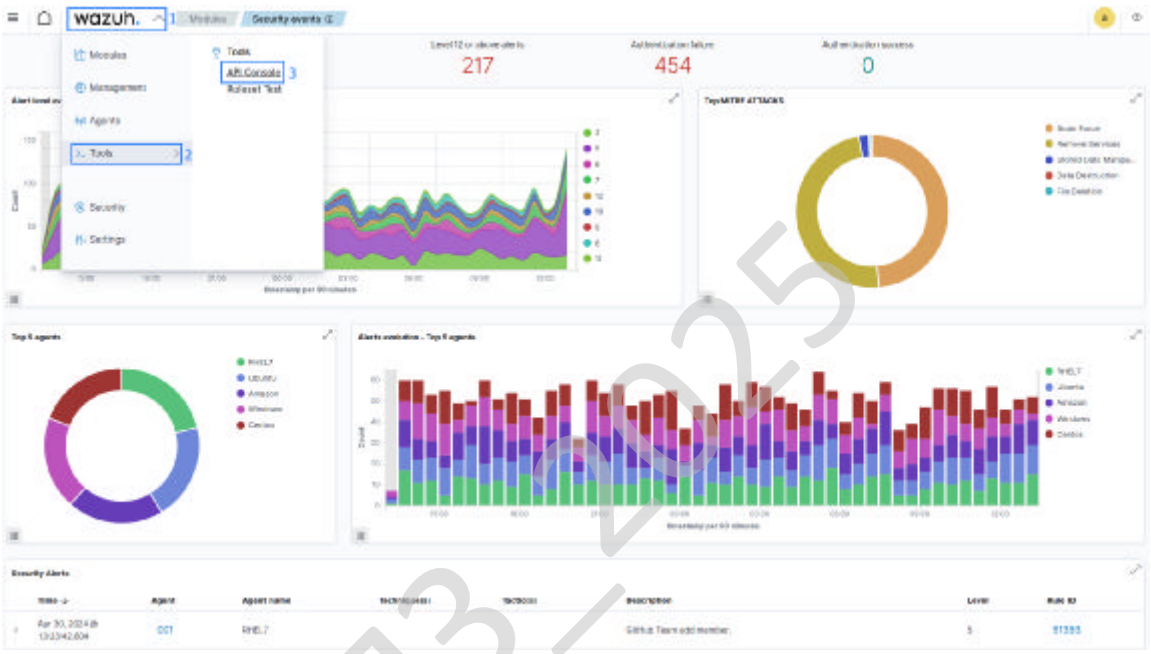

Рисунок 2.7 – Інтерфейс оператора Wazuh

Wazuh підтримує більшість операційних систем (Linux, Windows, macOS, BSD та ін.) та сучасні обчислювальні середовища, включаючи хмарні платформи (AWS, Azure, Google Cloud), а також контейнеризовані інфраструктури на базі Docker та Kubernetes. Це робить систему універсальною для інтеграції у сучасні корпоративні та гібридні інфраструктури.

Завдяки тісній інтеграції з Elastic Stack (Elasticsearch, Logstash, Kibana), а також власним агентам, Wazuh добре масштабується – від невеликих середовищ із кількома вузлами до великих інфраструктур із тисячами пристроїв і розподіленими дата-центрами.

Окремою перевагою є потужні можливості для аудиту безпеки та перевірки відповідності міжнародним стандартам, таким як PCI DSS, HIPAA, ISO/IEC 27001, SOC2, GDPR тощо. Wazuh надає набір політик, перевірок та дашбордів для оцінки відповідності цим вимогам. Це особливо важливо для організацій, які зобов'язані дотримуватись нормативних вимог у сфері обробки та зберігання даних.

Активна спільнота користувачів і розробників, наявність детальної документації та регулярні оновлення значно полегшують використання Wazuh, сприяють вирішенню проблем і вдосконаленню функціоналу.

Будучи побудованою на основі Elastic Stack, Wazuh частково наслідує його недоліки, як то складність в розгортанні та обслуговуванні, що потребує певнох тезнічнох експертизи, або доступ до розширених функцій лише за наявності платної ліценції.

На поточний момент Wazuh не має вбудованих засобів машинного навчання або штучного інтелекту у своїй базовій архітектурі. Проте система дозволяє інтегруватися з зовнішніми ML/AI-платформами або використовувати платні можливості Elastic Machine Learning для виявлення аномалій у логах. Також є можливість побудови власних аналітичних рішень на основі даних, що зберігаються в Elasticsearch, що забезпечує гнучкість і адаптацію до потреб конкретної організації.

Microsoft Sentinel

Microsoft Sentinel – хмарна SIEM та SOAR-платформа, розроблена компанією Microsoft на базі хмарної інфраструктури Azure. Замість розгортання на власних серверах, Sentinel надається як сервіс (SaaS), що пропонує масштабовану аналітику загроз, машинне навчання для виявлення аномалій та автоматизацію процесів безпеки, інтегрується з Microsoft 365 Azure та сторонніми рішеннями.

Архітектура Microsoft Sentinel включає кілька ключових компонентів, які спільно забезпечують повноцінну функціональність SIEM та SOAR-систем у хмарному середовищі Azure:

– Data Connectors – спеціалізовані мезанізми, що забезпечують збір даних з широкого спектра джерел: хмарних сервісів (Microsoft 365, Azure AD, AWS), локальної інфраструктури, мережевих пристроїв, систем захисту (антивіруси, фаєрволи) та багатьох інших. Це дозволяє централізовано агрегувати та аналізувати події безпеки з усієї IT-інфраструктури

– Log Analytics Workspace – основна платформа для зберігання, обробки та аналітики логів, побудована на базі Azure Monitor. Всі дані надходять до цього компонента, де над ними можна проводити складну аналітику з використанням Kusto Query Language (KQL) – потужної мови запитів, що використовується для аналізу, побудови власних правил детекції, візуалізації та створення інформативних дашбордів.

– Analytics Rules –набір аналітичних правил, які дозволяють виявляти загрози, корелювати події, формувати інциденти безпеки та запускати автоматичні дії. Правила можуть бути як вбудованими (на основі найкращих практик), так і користувацькими, створеними під специфіку окремого середовища.

– Playbooks – автоматизовані сценарії реагування, реалізовані за допомогою Azure Logic Apps. Вони забезпечують SOAR-функціональність: наприклад, після виявлення інциденту можна автоматично заблокувати IP-адресу, надіслати повідомлення, зібрати контекстну інформацію або запустити внутрішнє розслідування.

– Hunting – інструменти для практичного виявлення загроз, що дозволяють фахівцям з кібербезпеки проводити ручний або напівавтоматичний аналіз підозрілої активності. Тут також застосовується KQL, що дає змогу ефективно працювати з великими обсягами даних і виявляти складні, приховані загрози.

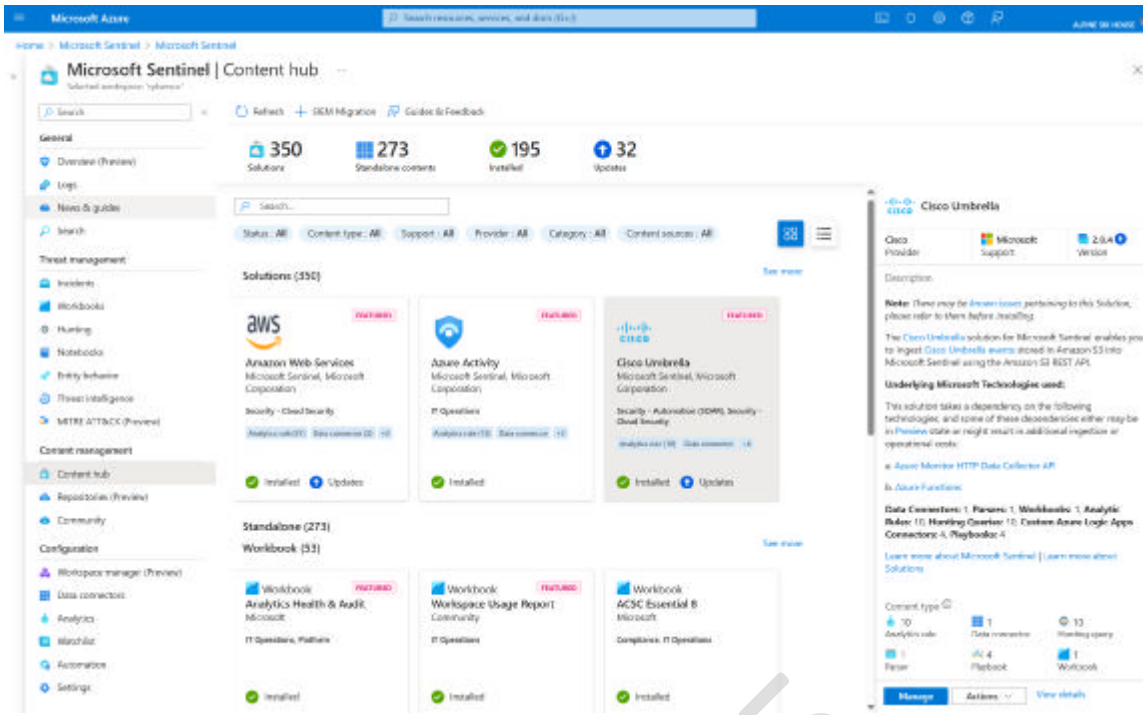

Рисунок 2.8 – Головна сторінка оператора Microsoft Sentinel

Microsoft Sentinel автоматично застосовує моделі машинного навчання для виявлення аномалій поведінки користувачів та пристроїв (UEBA). Крім того, Microsoft постійно оновлює готові шаблони аналітики, які базуються на глобальній аналітиці загроз, яку збирають їхні центри безпеки по всьому світу. Завдяки інтеграції з Microsoft Defender і XDR-рішеннями, Sentinel має змогу отримувати оброблені дані загроз із використанням поведінкових моделей, моделей виявлення атак на основі MITRE ATT&CK і навіть автоматичного аналізу інцидентів.

Microsoft Sentinel має повністю хмарну природу, вбудовану інтеграцію з усіма основними продуктами Microsoft (Azure, Microsoft 365, Defender тощо), що полегшує доступ до великих кількості джерел даних без складного налаштування. Playbooks дозволяють створювати дуже складні сценарії реагування без програмування. Також Sentinel дає впевненість у відповідності стандартам безпеки і комплаєнсу (наприклад, GDPR, HIPAA тощо).

Водночас, якщо організація не використовує екосистему Azure або її інфраструктура базується переважно на інших хмарних платформах, інтеграція

Microsoft Sentinel може виявитись складнішою. У таких випадках налаштування збору даних, автентифікація та автоматизація вимагатиме більше зусиль і не матиме таких ж глибокої інтеграції як у середовищі Microsoft. Це може зменшити ефективність використання Sentinel в якості SIEM-рішення.

Кожна з представлених SIEM систем має сильні сторони, що роблять їх більш влучними для певних сценаріїв використання. Вибір конкретного рішення залежить від потреб організації, наявної інфраструктури, бюджету та вимог до масштабованості, автоматизації й інтеграції.

Splunk – потужна, гнучка та масштабована платформа, яка підходить для великих організацій з високими вимогами до аналітики даних. Має багаті можливості візуалізації, пошуку й кореляції подій, а також добре розвинену екосистему доповнень. Однак її ліцензування є дорогим, що може бути суттєвим бар'єром для невеликих компаній.

ELK Stack (Elastic Stack + Elastic Security) – ідеальне рішення для організацій, які прагнуть до повного контролю над власною інфраструктурою безпеки. Це надзвичайно гнучка та потужна система з відкритим кодом, що дозволяє будувати SIEM під потреби організації. Потребує більше зусиль для налаштування та обслуговування, але забезпечує високу адаптивність. Особливо ефективна в поєднанні з функціями машинного навчання (в платній версії).

IBM QRadar – зріле корпоративне рішення з глибокими можливостями кореляції подій, виявлення складних атак і інтеграції з великим спектром систем. Найкраще підходить для великих компаній, фінансових і державних установ. Має високу точність, гнучку аналітику та можливість масштабування. Разом із цим, складне розгортання й висока вартість володіння роблять це рішення несумісним з середніми та маилими організаціями.

Microsoft Sentinel – хмарне SIEM-рішення з нативною інтеграцією в екосистему Microsoft. Підходить для організацій, що активно використовують Azure, Microsoft 365 і Microsoft Defender. Має потужні можливості SIEM, аналітики та автоматизації через Playbooks. Найкращий вибір для компаній,

орієнтованих на хмарні технології Microsoft. Водночас, інтеграція з іншими платформами може вимагати додаткових зусиль.

Wazuh – безкоштовне рішення з відкритим кодом, яке добре масштабується та пропонує широкий спектр засобів моніторингу, відповідності та реакції. Завдяки тісній інтеграції з Elastic Stack і можливості повного контролю на всіх рівнях, Wazuh є чудовою основою для побудови SIEM-системи. Арзітектурно система легко адаптується до різних середовищ, включаючи хмари та контейнери. Незважаючи на відсутність вбудованих модулів машинного навчання, її структура дозволяє інтегрувати зовнішні ML/AI-рішення для покращення виявлення загроз.

У підсумку, всі ці продукти мають сильні сторони. Однак саме Wazuh, завдяки своїй відкритості, гнучкості та архітектурній готовності до розширення, є особливо привабливою платформою для організацій, які хочуть поступово впроваджувати розширені аналітичні можливості, включаючи машинне навчання.

Одним із ключових напрямів розвитку SIEM-систем є **інтеграція машинного навчання та штучного інтелекту** у процеси виявлення загроз. Завдяки застосуванню алгоритмів машинного навчання системи здатні самостійно аналізувати величезні обсяги даних і виявляти закономірності, що виходять за межі нормальної поведінки. Замість традиційного підходу, який базується на статичних сигнатурах чи заздалегідь визначених правилах, системи з ШІ можуть виявляти нові, ще невідомі атаки (zero-day загрози), а також визначити складні, багатоступеневі атаки, що розгортаються поступово у часі. Крім цього, машинне навчання дозволяє зменшити кількість хибних спрацювань, оскільки система з часом «вчиться» краще розрізняти звичайні дії користувачів від потенційно шкідливої активності.

Важливою складовою такого підходу є **використання технологій UEBA (User and Entity Behavior Analytics)**. UEBA зосереджується на аналізі поведінки користувачів та об’єктів середовища (таких як сервери, робочі станції, мережеве обладнання) для виявлення нетипових або підозрілих дій. Система будує профілі

нормальної поведінки для кожного користувача та пристрою на основі історичних даних і згодом порівнює поточну активність із цими профілями. Наприклад, якщо працівник раптом починає отримувати доступ до файлів, з якими раніше не працював, або підключається до систем у незвичайний час, UEBA може сигналізувати про можливе порушення політик безпеки або компрометацію облікового запису. Таким чином, UEBA забезпечує більш глибокий і контекстуальний рівень контролю порівняно з класичними SIEM-методами.

Ще одним сучасним трендом є **архітектури на основі хмарних технологій**, які значно змінюють підхід до побудови SIEM-систем. Хмарні архітектури дозволяють організаціям не обмежуватися власними фізичними ресурсами для обробки та зберігання даних, а використовувати практично необмежені можливості масштабування в хмарі. Це дає змогу ефективно збирати й обробляти величезні потоки даних з численних джерел без ризику перевантаження системи або виникнення проблем із продуктивністю. Крім того, хмарні рішення значно спрощують розгортання нових компонентів, забезпечують високу доступність та резервування даних, а також автоматичні оновлення безпеки й функціональності. Усе це робить хмарні SIEM-рішення привабливими як для великих корпорацій, так і для середніх і навіть малих підприємств.

Не менш важливою інновацією є **використання SOAR-платформ (Security Orchestration, Automation and Response)** для автоматизації реагування на інциденти та їхнього подальшого розслідування. SOAR дозволяє автоматизувати значну частину рутинних дій, які раніше вимагали ручної участі аналітиків безпеки. Наприклад, при виявленні підозрілої активності платформа може автоматично заблокувати обліковий запис користувача, ізолювати заражений пристрій у мережі, зібрати необхідні артефакти для подальшого розслідування і навіть повідомити відповідальних осіб за допомогою інтегрованих комунікаційних каналів. Більш того, SOAR-платформи часто дозволяють будувати складні сценарії розслідувань, об’єднуючи дані з різних джерел, що

прискорює процес аналізу інцидентів і підвищує ефективність роботи команд безпеки.

Виявлення аномалій за допомогою машинного навчання є пріоритетним напрямком для дослідження та впровадження в сфері SIEM-систем, оскільки дозволяє переходити від реактивної до проактивної моделі захисту, виявляючи невідомі або малопомітні атаки, які не фіксуються традиційними сигнатурними методами. Такий підхід забезпечується здатністю до донавчання та адаптації до змін у поведінці користувачів і систем, знижує кількість хибнопозитивних спрацювань, зменшуючи навантаження на аналітиків безпеки, та підвищує ефективність виявлення складних загроз у реальному часі. Усе це робить машинне навчпння ключовою технологією для створення більш надійних, масштабованих та розумних систем виявлення кіберзагроз.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

OpenSearch – це відкритий проект для пошуку, аналітики та візуалізації даних, який був створений на базі відкритого коду Elasticsearch і Kibana після їхньої комерціалізації компанією Elastic. OpenSearch активно підтримується Amazon Web Services (AWS) та спільнотою розробників. Завдяки своїй відкритості, масштабованості й високій гнучкості, OpenSearch поступово стає популярним рішення не тільки для зберігання та пошуку логів, але й як повноцінна основа для побудови SIEM-систем.

У контексті побудови SIEM-платформи на основі Opensearch важливо розуміти роль і функціональність його основних компонентів, які забезпечують повний цикл роботи з даними: від збору та обробки до зберігання, аналізу й візуалізації.

– **OpenSearch Dashboards** – це графічний веб-інтерфейс, що дозволяє візуалізувати дані безпеки, будувати аналітичні дашбордиЮ виконувати

пошукові запити та налаштовувати сповіщення. Він забезпечує зручний інструментарій для аналітиків безпеки, що дозволяє працювати з даними в інтерактивному режимі, аналізувати події та щвидко реагувати на інциденти.

– **OpenSearch Core** – виконує роль основного рушія системи. Це розподілена NoSQL-база даних, оптимізовнана для індексації та аналітики великих обсягів структурованих і неструктурованих даних. Вона дозволяє здійснювати повнотекстовий пошук, агрегацію, фільтрацію та обробку у реальному часі. Такий підхід забезпечує масштабованість та високу продуктивність про роботі з логами та подіями безпеки.

– **Data Prepper** – компонент для підготовки даних до індексації, який працює як ETL-конвеєр (Extract – Transform – Load, з англ. «витяг – трансформація – завантаження»). Він приймає вхідні дані з різних джерел, виконує трансформацію, нормалізацію, збагачення (наприклад, додавання інформації про геолокацію, категоризацію, марування типів подій) та передає їх до OpenSearch для зберігання. Завдяки Data Prepper можна забезпечити уніфікований формат даних незалежно від джерела.

– **Alerting Plugin** – дозволяє створювати правила виявлення подій і тригерів на основі аналітичних запитів. При настанні певних умов, система може генерувати сповіщення, надсилати повідомлення електронною поштою, запускати зовнішні сценарії реагування або інтегруватися з іншими системами. Це дозволяє автоматизувати реагування на потенційні інциденти та зменшити час на їх виявлення.

– **Security Plugin** – забезпечує безпеку самої SIEM-платформи. Він впроваджує механізми автентифікації користвачів через LDAP, SAML, OpenID або внутрішні облікові записи, реалізує розмежування прав доступу на основі ролей, контроль доступу до індексівЮ дашбордів та API. Крім того, підтримується аудит дій користувачів, шифрування даних у транзиті та на диску, що критично важливо для кібербезпеки.

– Anomaly Detection Plugin – окремий модуль, який дозволяє виявляти нетипову поведінку або аномалії в потоці даних за допомогою алгоритмів машинного навчання. Цей компонент автоматично навчається на історичних даних, будує моделі типових шаблонів поведінки та сигналізує про відхилення, які можуть свідчити про підозрілі дії або атаки. Він є одним з перших кроків до впровадження поведінкового аналізу та автоматизованої аналітики в SIEM-системах на базі OpenSearch.

У складі SIEM-системи OpenSearch виконує ключові функції, які забезпечують повний цикл роботи з логами та подіями безпеки. Насамперед, платформа відповідає за збір даних з широкого спектру джерел – це можуть бути сервери, мережеві пристрої, робочі станції, хмарні сервіси, засоби захисту кінцевих точок тощо. Зібрані дані проходять попередню обробку, зокрема нормалізацію, після чого індексуються для забезпечення ефективного пошуку, фільтрації та аналізу.

Важливою складовою функціональності OpenSearch в контексті SIEM є можливість виконання кореляції подій. Завдяки побудові складних запитів можна встановлювати взаємозв'язки між подіями, що надходять з різних джерел, і виявляти ознаки потенційних атак або порушень безпеки. Цей процес суттєво підвищує якість і точність виявлення інцидентів.

Для зручності аналітиків та команди оперативного реагування OpenSearch надає широкі можливості візуалізації. За допомогою інтерактивних дашбордів можна отримувати загальну картину стану інформаційної безпеки, відстежувати ключові показники в режимі реального часу, проводити розслідування інцидентів і виявляти тренди.

Також платформа дозволяє налаштовувати автоматичні оповіщення. Коли вхідні дані задовольняють визначеним умовам, або виявляються аномальні події, система може автоматично повідомити відповідальних осіб через електронну пошту, API або зовнішні сервіси, що прискорює реагування на загрози.

Одним з ключових інструментів OpenSearch для підвищення ефективності виявлення загроз є **Anomaly Detection Plugin**. Цей модуль використовує моделі машинного навчання (зокрема алгоритми на основі рекурентних нейронних мереж та детекторів сезонних аномалій) для побудови профілю «нормальної» поведінки системи за історичними даними. Після навчання модель аналізує нові дані в реальному часі й виділяє ті події, що істотно відрізняються від звичних шаблонів.

Anomaly Detection у OpenSearch має наступні особливості:

– Підтримка багатовимірного аналізу, що дозволяє враховувати кілька метрик одночасно (наприклад, трафік, кількість помилок, частоту доступів).

– Можливість налаштування чутливості моделей виявлення аномалій.

– Інтеграція із системою сповіщення, що дозволяє автоматично надсилати повідомлення при виявленні підозрілих відхилень.

– Робота як у режимі реального часу, так і на історичних даних для ретроспективного аналізу.

OpenSearch має певні переваги, як то відкритість коду та безкоштовна ліцензія, висока гнучкість налаштувань і розширюваність завдяки плагінам. OpenSearch може бути масштабована від невеликих локальних установок до розподілених систем обробки петабайтів даних, а також має глибоку інтеграцію із хмарними сервісами AWS та іншими інструментами.

Завдяки вбудованим можливостям виявлення аномалій і модульності, OpenSearch дозволяє створити адаптивне середовище для моніторингу безпеки. Проте повноцінна реалізація SIEM-функціоналу на базі OpenSearch потребує глибоких знань у галузі кібербезпеки, інженерії даних та розробки спеціалізованих правил виявлення загроз.

Крім готових SIEM-продуктів, існують також технології та архітектурні підходи, що дозволяють підвищити ефективність виявлення загроз у корпоративних та хмарних середовищах. Ці рішення не тільки розширюють

можливості традиційних SIEM-систем, але й закладають основу для створення більш адаптованих, автоматизованих і точних систем безпеки.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає ПЗ системи виявлення та моніторингу кіберзагроз та протидії кібератакам, побудована на основі відкритих компонентів SIEM (Security Information and Event Management) із впровадженням методу штучного інтелекту, зокрема машинного навчання для виявлення аномалій, з метою підвищення ефективності виявлення інцидентів інформаційної безпеки.

У процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих рішень у сфері SIEM-систем і технологій виявлення аномалій за допомогою ШІ; виявити їхні сильні та слабкі сторони; результати аналізу врахувати під час проєктування власного рішення;

б) обґрунтувати вибір програмної архітектури SIEM-системи з відкритими компонентами, придатної до інтеграції з модулями машинного навчання; розробити загальну структурну схему системи та визначити функціональні компоненти;

в) реалізувати програмне забезпечення SIEM-системи з включенням таких компонентів, як збір, зберігання, нормалізація, кореляція подій, а також модуль виявлення аномалій на основі машинного навчання; побудувати блок-схеми основних алгоритмів роботи та взіємодії підсистем;

г) організувати інтерфейс користувача з можливістю перегляду подій, графіків, сповіщень та повідомлень про потенційні загрози й аномальну активність в системі;

д) розробити рекомендації щодо організаційних та методичних заходів,що

сприятимуть впровадженню системи в практичне використання, а також забезпечать її подальшу підтримку та масштабування;

е) провести розрахунки для визначення економічної доцільності впроваждення створеної системи порівняно з альтернативними рішеннями;

ж) розробити заходи з охорони праці при впровадженні ті аикористанні системи, а такоє заходи з цивільного захисту в умовах потенційних інформаційних загроз;

з) сформулювати висновки щодо виконаного обсягу робіт, досягнутих результатів та перспектив подальшого розвитку системи.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Під час реалізації бакалаврського проєкту було проведено дослідження сучасного стану, тенденцій та перспектив розвитку систем забезпечення інформаційної безпеки в корпоративних мережах. Особливу увагу приділено аналізу архітектури та принципів функціонування SIEM-систем (Securtiy Information and Event Management), які поєднують функції збору, зберігання, кореляції та аналізу подій безпеки з різних джерел у межах інформаційної інфраструктури підприємства.

У роботі розглянуто типовий цикл обробки подій у SIEM-системі, який охоплює такі етапи: збір логів і подій з мережевого та серверного обладнання, уніфікація і нормалізація даних, кореляція за заданими правилами, збереження у сховищі, реєстрація інцидентів, реагування на загрози та звітність. На основі аналізу виявлено, що традиційні підходи, засновані виключно на сигнатурних правилах, не завжди здатні виявляти складні атаки, а також схильні до великої кількості хибних спрацювань.

У зв’язку з цим запропоновано інтегрування до SIEM-системи метод виявлення аномалій на основі алгоритму машинного навчання Random Cut Forest (RCF, з англ. «ліс випадкових зрізів»). Модель RCF навчена на історичних даних поведінки користувачів і систем, і дозволяє виявляти відхилення, які потенційно свідчать про зловмисну активність. Алгоритм функціонує на етапі після нормалізації та збереження даних, обробляючи вхідні події в режимі реального часу та формуючи оцінку аномальності для кожного запису. У разі перевищення порогового значення система формує сигнал про потенційний інцидент, який надсилається в подільший процес обробки.

У системі Wazuh версії 4.7 для збору інформації з контрольованих пристроїв використовуються агенти, які встановлюються безпосередньо на цільові системи. Ці агенти здійснюють збір даних, зокрема системних журналів, подій безпеки, конфігураційних змін тощо, після чого передають їх на центральний компонент – Wazuh Manager. Менеджер здійснює попередню обробку отриманих даних: виконується декодування за допомогою вбудованих декодерів, а також застосовуються правила кореляції для виявлення потенційних інцидентів безпеки. Після аналізу сформовані структуровані події передаються на компонент Wazuh Indexer, побудований на базі OpenSearch, для індексації та довготривалого зберігання. Передача даних між менеджером і індексатором реалізується через вбудований REST API клієнт у форматі JSON. У результаті події з агентів зберігаються у спеціалізованих індексах OpenSearch (наприклад, "*wazuh-alerts-**"), що забезпечує ефективний пошук, візуалізацію та подальший аналіз зібраної інформації. Індексом у даному контексті називається логічна структура для організації, зберігання та пошуку даних, подібна до таблиць в реляційних базах даних.

Обробка та індексація даних у OpenSearch

Отримані з Wazuh Manager події надходять у OpenSearch як структуровані JSON-документи. Індексатор OpenSearch застосовує індексні шаблони та Ingest Pipeline (що надаються Wazuh), які автоматично визначають відповідні поля (мітка часу, ідентифікатор хоста, тип події, серйозність тощо) і записують дані у часові індекси. Наприклад, поле з часом фіксації події розпізнається як *timestamp*, поля деталей – як текстові чи числові. Такий підхід гарантує, що дані будуть оптимізовано представлені в індексах для швидкого пошуку, агрегації та подальшого аналізу. У результаті всі події зберігаються в OpenSearch і доступні для запитів та виконання аналітики.

Побудова моделей у Anomaly Detection Plugin

У плагіні Anomaly Detection OpenSearch кожний процес виявлення аномалій називається детектором. Детектор налаштовується на конкретний

індекс (або набір індексів) подій та певний часовий проміжок (наприклад, агрегування кожні 5 хвилин). Ключовою складовою детектора є набір ознак (features). Кожна ознака – це обране поле індексу з агрегацією: середнє (*average()*), сума (*sum()*), кількість (*count()*), мінімум (*min()*) або максимум (*max()*). Наприклад, ознакою може бути "кількість невдалих входів за період часу" з агрегацією *count()*. Детектор може містити одну або кілька ознак; багатовимірна модель враховує кореляцію між цими ознаками, хоча надто багато вимірів може знизити чутливість через "прокляття розмірності".

«Прокляття розмірності» (англ. curse of dimensionality) в машинному навчанні – це явище, при якому зростання кількості вимірів (тобто ознак) у даних ускладнює аналіз, обробку й моделювання. У контексті виявлення аномалій, з кожною новою ознакою модель працює у все більш багатовимірному просторі. У такому просторі відстані між точками стають менш інформативними: усі точки стають майже однаково «далекими» одна від одної. Це ускладнює виявлення схожих або аномальних шаблонів, бо різниця між нормою та аномалією «розмивається». Більше вимірів також потребує більше даних для якісного навчання – інакше модель буде недонавченою або нестабільною.

Алгорим виявлення аномалій, який реалізовано у плагіні Anomaly Detection платформи OpenSearch, ґрунтується на методі **Random Cut Forest (RCF)** – «лісі випадкових розрізів». Цей алгоритм належить до класу **неконтрольованого навчання (unsupervised learning)**, що не потребує попередньох розмітки даних чи наявності еталонів «нормальної» та «аномальної» поведінки. RCF будує множину дерев, кожне з яких формує ієрархічну структуру поділу простору ознак за допомогою **випадкових розрізів**. Кожний такий розріз – це гіперплощина, яка випадково розділяє множину точно у просторі ознак. Таким чином, модель представляє **структуру густини розподілу даних**, де точки, що потрапляють у **розріжені** області простору, характеризуються вищою ймовірністю бути аномаліями.

Вхідні дані подаються на вхід алгоритму в режимі **потоку (streaming**

mode), тобто обробляються послідовно, у міру наджодження. Для кожного вектору ознак, що описує нове спостереження, алгоритм обчислює **оцінку аномальності (anomaly grade)** – значення в діапазоні від 0 до 1, де 0 означає повну відповідність нормальній поведінці, а значення, близькі до 1 – високу ймовірність аномалії. Окрім того, алгоритм генерує **показник достовірності (confidence score)**, який вказує на надійність розрахованої оцінки.

Етапи навчання та виявлення аномалій

При стовреннні детектора в плагіні Anomaly Detection вказують історичний інтервал для початкового тренування моделі. Плагін агрегує дані за цей період (наприклад, попередні декілька днів) і застосовує алгоритм Random Cut Forest (RFC) для побудови початкової моделі поведінки на основі отриманих даних та заданих ознаках. Після завершення навчання детектор переходить у **режим виявлення в реальному часі**. Періодично, в залежності від заданого параметру *detection_interval*, він виконує пошук нових даних у зазначених індексах, обчислює для них поточні агреговані значення ознак, формує вектор ознак для поточного часового вікна, та передає його до RCF-моделі для обчислення показників аномальності. Алгоритм оцінює аномальність у кожному інтервалі, поступово оновлюючи модель (streaming) та генеруючи результати. Для забезпечення стабільності оцінок модель використовує механізм шінглів (shingles) – послідовностей попередніх векторів, що створюють контекст для аналізу. Модель починає генерувати результати лише після накопичення достатньої кількості таких послідовностей.

Таким чином, **коротші** часові інтервали забезпечують швидший початок роботи, тоді як **довші** потребують більше часу для накопичення достатньої кількості точок, що зумовлює затримку в генерації перших оцінок. У підсумку для кожного інтервалу часу детектор обчислює два ключові параметри:

– **anomaly grade** – числова оцінка ступеня аномальності, в діапазоні від 0 (норма) до 1 (максимальне відхилення);

– **confidence** – рівень впевненості моделі у достовірності результату, що враховує кількість накопичених даних і стабільність оцінок.

Приклади моделей

Модель для однієї ознаки. У випадку використання моделі для однієї ознаки, наприклад, для аналізу кількості невдалих спроб входу за кожні 5 хвилин, конфігурується детектор з єдиною ознакою – полем даних, таким як *login.failures*, із агрегацією типу *count()*. Плагін Anomaly Detection у OpenSearch здійснює пошук відповідних подій у заданому індексі за допомогою фільтра пошукового запиту, рахує кількість подій у кожному часовому вікну та формує послідовність числових значень. Ця послідовність передається до алгоритму Random Cut Forest (RCF), який спочатку навчається на історичних даних, що відображають нормальну частоту таких подій, а надалі в режимі реального часу отримує нові значення, аналізує їх у контексті попередніх спостережень і формує оцінки аномальності для кожного інтервалу. Одновимірна модель у такому випадку працює з одновимірним часовим рядом і оцінює відхилення від типового патерну.

Модель для кількох ознак. У разі використання моделі з кількома ознаками, наприклад, для одночасного аналізу кількох спроб входу (*login.attempts*) і кількості помилок автентифікації (*auth.errors*) за 5-хвилинні інтервали, у конфігурації детектора задіюються дві ознаки, кожна з агрегацією типу *count()*. Для кожного часового вікна формужться двовимірний вектор (кількість спроб входу, кількість помилок автентифікації), який подається на вхід моделі Random Cut Forest. У цьому випадку алгоритм оперує у двовимірному просторі ознак, створюючи «випадковий ліз розрізів», що дозводяє враховувати кореляцію між ознаками – такий підхід забезпечує виявлення не лише оремих екстремальних значень, але й аномальних поєднань, які виходять за межі характерного патерну для заданих ознак. Водночас слід враховувати, що зі збільшенням числа ознак (вимірів) зростає ризик «прокляття розмірності»: малочастотні аномалії в багатовимірному просторі важче вловити. Тому на

практиці обира.ть до трьох ключових ознак, що найкраще відображають поведінку системи.

В обох випадках підготовка вхідних даних для моделі здійснюється шляхом задання фільтрів і типів агрегації у параметрах детектора через графічний інтерфейс або API. На основі цих налаштувань плагін автоматично здійснює вибірку подій із заданого індексу OpenSearch, виконує обчислення значень ознак для кожного часового інтервалу та передає сформовані вектори до моделі Random Cut Forest. Результати обробки, включно з показником аномальності (anomaly grade), рівнем достовірності (confidence) та допоміжними метаданими, зберігаються в окремому індексі або журналі результатів, доступному для подільшого аналізу.

Перетворення аномалій у сповіщення

Коли плагін виявляє, що значення показника аномалії перевищує заданий поріг, відпоідну подію можна автоматично трансформувати у сповіщення. Для цього передбачено кілька механізмів інтеграції: наприклад, у самому OpenSearch доступний плагін Alerting, який дозволяє створити монітор (Monitor) на базі індексу результатів аномалій, визначити умови тригера (наприклад, якщо *anomaly grade > 0.5*) та сконфігурувати спосіб доставки сповіщення (електронна пошта, webhook тощо). Альтернативно, можлива передача інформації про виявлену аномалію до менеджера Wazuh – через API або лог-файл – де правило кореляції Wazuh аналізує отримані дані і генерує стандартне сповіщення або інцидент, використовуючи вбудовані засоби, такі як Active Response.

Причини вибору RCF для потокових даних

Алгоритм Random Cut Forest був обраний для виявлення аномалій у потокових даних завдяки своїм архітектурним і алгоритмічним особливостям, які ідеально відповідають вимогам до обробки телеметричних і логових подій у реальному часі в розподілених середовищах:

– RCF реалізує **потокову (streaming) обробку** – кожна нова точка даних (агрегований вектор ознак за часовий інтервал) може бути безпосередньо

оброблена без потреби зберігати чи перечитувати повну історію спостережень; це дає змогу працювати з великими обсягами даних, характерними для систем моніторингу;

– модель підтримує **безперервне (інкрементальне) навчання**: структура «лісу» оновлюється динамічно з надходженням нових точок, шляхом модифікації або обмеження перебудови дерев, що дозволяє моделі адаптуватися до змін поведінки системи без повного перенавчання;

– **ресурсна ефективність**: реалізація RCF не вимагає великого обсягу пам'яті чи обчислювальних ресурсів і може виконуватись на тому ж кластері OpenSearch, де працює основна аналітика, зберігання та пошук; алгоритм добре масштабується, оскільки RCF-моделі можуть бути розподілені між вузлами кластера;

– **відсутність потреби в помічених даних**: оскільки у практичних сценаріях подій не існує безумовного маркування «аномалія/норма», RCF як неконтрольований (unsupervised) метод здатний виявляти статистично рідкісні або нетипові шаблони поведінки самостійно, без участі оператора;

– алгоритм підкримує **багатовимірний аналіз**, що дозволяє виявляти не лише окремі аномальні ознаки, а й складні кореляції між кількома метриками, робить RCF придатним для моделювання поведінки у сценаріях, де критичні порушення можуть проявлятись через взаємозалежні зміни в кількох ознаках.

З огляду на особливості задач виявлення аномалій у SIEM-системах – а саме обробку потокових логів безпеки, непередбачуваність подій, відсутність апріорного маркування та вимоги до масштабованості – алгоритм Random Cut Forest (RCF) демонструє високу ефективність порівняно з іншими підходами машинного навчання. Його здатність донавчання дає змогу обробляти вхідні дані в режимі реального часу без збереження повної історії подій, що є критично важливим для систем з високою часткою логів. Крім того, RCF підтримує інкрементальне навчання, що дозволяє адаптувати модель до змін у поведінці інфраструктури без необхідності новного перенавчання.

Інші популярні підходи мають свої обмеження в контексті SIEM. Наприклад, **One-Class SVM**, хоч і призначений для неконтрольованого навчання, є статичним та слабо масштабованим, вимагає нормалізції ознак та попереднього формування навчальної вибірки, що обмежує його застосування у потокових сценаріях. **Autoencoder** і **LSTM** добре працюють з послідовностями та складними залежностями, але їх навчання потребує розмічених даних, значних обчислювальних ресурсів та часу. Вони важко інтерпретуються, що знижує прозорість в умовах аудиту подій безпеки. **Isolation Forest**, як і RCF, є неконтрольованим методом, але не підтримує інкрементальне оновлення моделі, що робить його менш придатним до динамічних IT-середовищ.

Таким чином, RCF поєднує в собі низку ключових властивостей, що роблять його придатним саме для задач моніторингу подій безпеки: робота в реальному часі, підтримка багатовимірних ознак, здатність виявляти рідкісні або кореляційні аномалії, стійкійсть до шуму та розподіленість. Це лозволяє інтегрувати його в інструменти на зразок OpenSearch та ефективно використовувати в рамках побудови SIEM-систем, де важлива не лише точність, але й час реагування та масштабованість у розподіленому середовищі.

3.2 Розробка структурної схеми

Побудова структурованої та модульної архітектури SIEM-системи з інтегрованими елементами штучного інтелекту є ключовою передумовою для ефективного виявлнення загроз та оперативного реагування на інциденти інформаційної безпеки. Забезпечення чіткого розподілу функціональних обов’язків між компонентами системи – зокрема, засобами збору подій, механізмами попередньої обробк, сховищем даних, аналітичним ядром та підсистемою сповіщень – є необхідною умовою для досягнення високої гнучкості, масштабваності та надійності у великих корпоративних і промислових інфраструктурах.

У подальшій частині розділу представлено уточнений склад компонентів системи та описано логіку обміну даними між ними на прикладі SIEM-рішення, побудованого на базі Wazuh та OpenSearch. Відповідна структурна схема наведена нажче на рисунку 3.1.

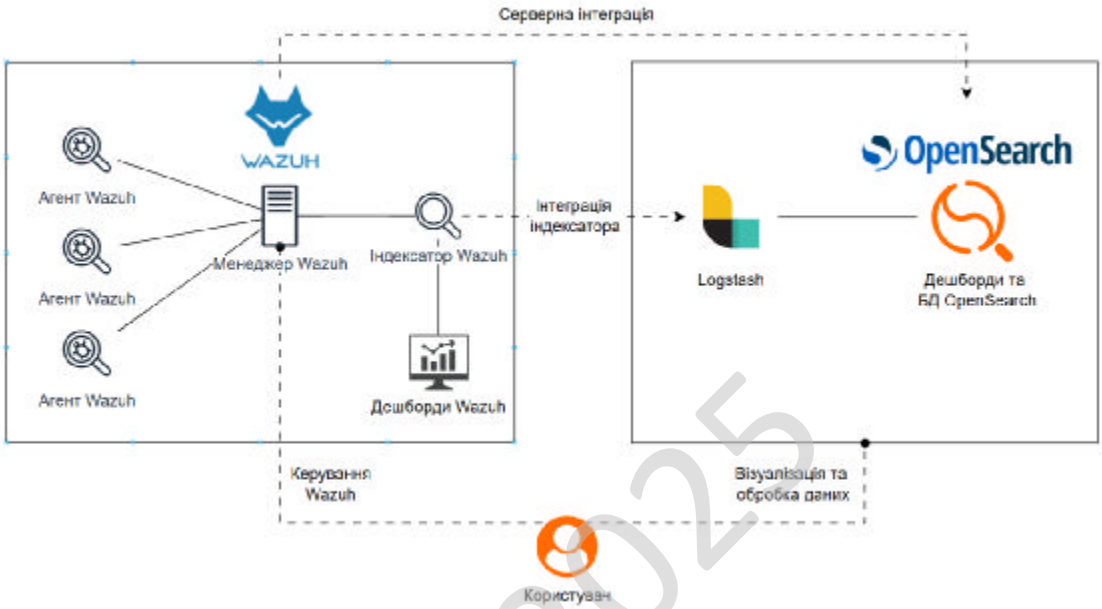

Рисунок 3.1 – Структурна схема SIEM-системи

Склад системи та інтеграція компонентів

Wazuh Agents – це легкі програмні компоненти, що розгортаються на кінцевих вузлах інфраструктури, включаючи сервери, робочі станції та віртуальні машини. Основне призначення агента полягає у безперервному зборі подій безпеки та телеметрії з локального хосту. До типових джерел даних належать системні журнали (наприклад, Syslog, Windows Event Log), журнали прокладних сервісів, події моніторингу змін у файловій системі (FIM), інформація про активні процеси, мережеві з'єднання, конфігурації служб безпеки та інші артефакти.

Зібрані події нормалізуються у формат структурованих JSON-документів і тімчасово зберігаються у локальній черзі на вузлі. Передача даних до цетрального компонента – Wazuh Manager – здійснюється через захищений канал зв'язку (TCP з TLS-шифруванням). Таким чином, Wazuh Agent виконує роль

основного елемента збору та попередньої обробки інформації в архітектурі SIEM-систем.

Wazuh Manager – це центральний аналітичний та координаційний компонент системи, відповідальний за обробку, кореляцію та класифікацію подій безпеки. Він приймає структуровані повідомлення від Wazuh Agent через захищене мережеве з'єднання (TCP/TLS) та виконує подільшу обробку отриманих даних.

Ключовими функціями менеджера є:

– **Декодування**: трансформація вхідних логів в уніфікований формат з метаданими, необхідними для подальшого аналізу;

– **Застосування правил**: використання вбудованого рушія кореляції для виявлення потенційно шкідливих або аномальних дій, включаючи спроби несанкціонованого доступу, змін у критичних файлах, порушень політик безпеки тощо.

Wazuh Manager забезпечує базову інтелектуальну обробку потоків подій, створюючи на основі них алерти, які потім можуть бути передані до зовнішніх систем зберігання, візуалізації або подальшої аналітики.

Logstash у даній архітектурі виконує роль посередника між аналітичним ядром Wazuh Manager та системою зберігання й пошуку OpenSearch. Його завдання – забезпечити узгоджену передачу подій, які надходять від менеджера безпеки, у форматі, придатному для індексації та подальшої аналітики. Роль Logstash полягає в адаптації структури повідомлень до вимог OpenSearch, фільтрації або мінімальній нормалізації даних за потреби, та уніфікації різнорідних джерел (такі що не підтримують Wazuh Agents, як то мережеве обладнання тощо) у спільному форматі для централізованого зберігання.

У цьому контексті Logstash не є повноцінним інструментом попереднього аналізу, а радше – транспортно-конферсійним вузлом, який дозволяє узгодити логіку обробки даних у системі та мінімізувати розрив між компонентами з різною структурою виводу. Його наявність особливо виправдана у випадках, коли

необхідно реалізувати спільну точку контролю або конвертації потоку логів до надходження в основне сховище.

OpenSearch (Core/Cluster) – центральний компонент системи, що виконує роль високопродуктивного сховища даних та аналітичного рушія. Побудований на основі Elasticsearch, OpenSearch забезпечує горизонтальне масштабування, розподілене зберігання та пошук даних у реальному часі.

У контексті SIEM-системи, OpenSearch приймає події, що надходять від попередніх компонентів, таких як Wazuh Manager (оброблені дані з Wazuh Agent), Filebeat або Logstash, через HTTP API. Ці події індексуються у вигляді документів у спеціалізованих індексах. Для забезпечення правильної обробки цих подій попередньо налаштовуються шаблони індексів (index templates) та канали інжесту (ingest pipelines), які автоматично визначають структуру даних і застосовують відповідні трансформації.

Крім того, OpenSearch підтримує плагіни, такі як Security для реалізації рольової моделі безпеки, та Anomaly Detection для виявлення аномалій за допомогою моделей машинного навчання, наприклад, Random Cut Forest.

Таким чином, OpenSearch забезпечує централізоване зберігання, обробку та аналіз логів, що є критично важливим для ефективного функціонування SIEM-системи.

OpenSearch Dashboards виконує роль основного інтерфейсу користувача для візуального доступу до подій безпеки, аналітики та керування інцидентами в системі. Цей веб-інтерфейс дає змогу аналітикам SOC, адміністраторам та інженерам спостерігати за станом інфраструктури в режимі реального часу, здійснювати пошук у журналах подій, досліджувати інциденти й будувати інформативні панелі та звіти.

У межах запропонованої архітектури Dashboards виступає центральною точкою взаємодії з користувачем і містить низку інтегрованих модулів:

– **Wazuh App** – спеціалізований модуль для візуалізації подій, що надходять від Wazuh Manager. Надає доступ до детальної інформації про хости,

правила безпеки, користувачів і джерела загроз, що дозволяє швидко ідентифікувати інциденти.

– **Anomaly Detection UI** – інтерфейс для налаштування і моніторингу моделей виявлення аномалій, які автоматично аналізують поведінку системи та фіксують відхилення від норми.

– **Alerting** – інструмент створення умовних тригерів для автоматичного надсилання сповіщень за визначеними подіями (наприклад, перевищення критичного рівня ризику). Дає змогу інтегрувати систему з каналами сповіщень (електронна пошта, API, webhook тощо) або зовнішніми платформами для автоматизованого реагування (SOAR).

Таким чином, OpenSearch Dashboards об’єлнує функції моніторингу, розслідування інцидентів та операційного управління, забезпечуючи завершене робоче середовище для взаємодії з SIEM-системою. Зв’язок з ядром OpenSearch здійснюється через REST API, що гарантує динамічне оновлення даних і швидкий відгук на запит користувача.

Загальна архітектура передачі та обробки подій

Wazuh Agent встановлюється на кінцевих вузлах і виконує збір системних подій, логів додатків, результатів моніторингу цілісності файлів тощо. Агент формує події у форматі JSON і передає їх по захищеному каналу (TLS через TCP порт 1514) на центральний вузол **Wazuh Manager**.

Wazuh Manager приймає події від агентів, проводить базову обробку – декодування, нормалізацію та застосування сигнатурних правил. Оброблені повідомлення зберігаються локально і паралельно передаються до компонента **Logstash** для додаткової обробки, зокрема фільтрації, трансформації та нормалізації подій. Відправка даних до **Logstash** забезпечує гнучкість у налаштуванні обробки подій, даючи можливість застосовувати складні фільтри та перетворення, що необхідно для оптимальної інтеграції даних з OpenSearch. Після цього оброблені події передаються в **OpenSearch** для зберігання та подальшого аналізу.

Logstash є потужним інструментом для обробки даних, який використовується для прийому, обробки і передачі подій в систему зберігання даних, таку як OpenSearch. За допомогою плагінів, таких як **Grok**, **Mutate**, **Translate**, Logstash здатний здійснювати складну обробку даних: розпізнавати та перетворювати складні формати логу в структуровані поля, видаляти зайву інформацію чи додавати додаткові метадані тощо. Після обробки дані відправляються в **OpenSearch**, де вони індексуються та зберігаються для подальшого аналізу. Такий підхід дозволяє більш гнучко керувати даними, а також оптимізувати процеси передавання інформації до сховища, в особливосіі для складних структур даних чи нестандартних форматів логу.

OpenSearch приймає події, які були надіслані від Logstash або іншого попереднього етапу обробки, і додаково обробляє їх за допомогою **ingest pipelines**. На цьому етапі здійснюється фінальне розпізнавання полів, трансформація даних, як то коректне форматування часових міток, розбір IP-адрес, а також обробка інших метаданих, які необхідні для подальшого аналізу. Це забезпечує правильне структурування подій відповідно до вимог індексів, що дозволяє забезпечити оптимальний пошук та аналіз даних. Після цього дані індексуються у відповідні індекси в кластері **OpenSearch**, де вони зберігаються для швидкого пошуку й аналітики. Інтеграція з **OpenSearch Dashboards** дозволяє зручно візуалізувати та моніторити ці дані через інтерфейс, а також створювати різноманітні звіти та дашборди для реагування на інциденти та загрози.

OpenSearch Dashboards звертається до **OpenSearch** через REST API для отримання даних. У веб-інтерфейсі відображаються як класичні інциденти, що виникають за результатами обробки подій Wazuh, так і аномалії, що виявлені за допомогою плагіна **Anomaly Detection**. Крім того, в Dashboards можна здійснювати аналітику та моніторинг подій у реальному часі, а також створювати й налаштовувати монітори для спостереження за різними метриками. За допомогою плагіна **Alerting** в Dashboards задаються умови спрацювання оповіщень, наприклад, при перевищенні порогу значення **anomaly_score**. Коли ці

умови виконуються, система генерує оповіщення, яке може бути надіслано відповідальним особам через електронну пошту, webhook, або інтегровано з зовнішніми **SOAR**-платформами для автоматизації подальших дій.

Anomaly Detection Plugin в складі **OpenSearch** здійснює обробку структурованих даних шляхом агрегації метрик, що дозволяє виявляти відхилення від нормальних шаблонів. Плагін використовує методи машинного навчання для створення детекторів аномалій, які навчаються на історичних даних з метою розпізнавання шаблонів поведінки. Після навчання, ці детектори використовуються для реального часу: кожна нова подія порівнюється з уже навченою моделлю для оцінки її аномальності. Визначені аномальні події отримують відповідний бал аномальності, що відображає ступінь відхилення. Ці результати обробляються і зберігаються для подальшого аналізу та можуть бути використані для генерування сповіщень або запуску автоматизованих дій через систему оповіщень.

Типовий ланцюг подій в системі включає наступні етапи: **Wazuh Agent** збирає події з моніторованих вузлів і передає їх на **Wazuh Manager** для початкової обробки, включаючи декодування та нормалізацію. Потім оброблені дані можуть бути додатково оброблені в **Logstash**, після чого передаються до **OpenSearch** для індексації через ingest pipelines. Після індексації дані доступні для подальшого аналізу, включаючи виявлення аномалій, що здійснюється через **Anomaly Detection Plugin**. Результати аналізу доступні для візуалізації та моніторингу через **OpenSearch Dashboards**, а також для генерації сповіщень через плагін Alerting. Останні можуть бути передані на зовнішні платформи для автоматизованої реакції через SOAR або інші механізми. Усі етапи комунікації між компонентами здійснюються за допомогою стандартних протоколів, таких як TLS для захищеного з'єднання та HTTP/REST для інтеграції з **OpenSearch**. Система має здатність масштабуватися як вертикально (через додавання ресурсів на рівні окремих вузлів), так і горизонтально (через додавання нових вузлів до кластеру).

3.3 Розробка функціональної схеми

Для повноцінного розуміння принципів роботи системи моніторингу інформаційної безпеки важливо не лише знати її структуру, а й усвідомлювати логіку функціонування кожного компонента та їхню взаємодію. Функціональна схема дозволяє уявити систему як послідовність обробки даних – від моменту збору до формування аналітичних висновків і реагування на виявлені інциденти.

У даному розділі наведено логічний поділ схеми на основні функціональні блоки, кожен з яких виконує окремий етап у загальному потоці обробки подій безпеки. Такий підхід дає змогу оцінити роль кожного компонента в рамках загальної архітектури SIEM-рішення та визначати послідовність передачі даних між ними.

Обрана архітектура системи базується на використанні поширених компонентів SIEM-систем, таких як: Wazuh Agent, Wazuh Manager, Logstash, OpenSearch (з інжест-пайплайнами) та OpenSearch Dashboards. Кожний компонент виконує чітко визначену роль, що забезпечує масштабованість, розширюваність та адаптивність системи до різних джерел подій та типів аналізу.

Функціональна схема, що була побудована, відображає дві основні гілки надходження подій:

– події, що генеруються на системах зі встановленим агентом Wazuh, обробляються на рівні вузла-джерела, шифруються та надсилаються на центральний Wazuh Manager;

– безагентні події (наприклад, Syslog з мережевого обладнання) передаються напряму до Logstash, минаючи обробку та шифрування агентом.

Wazuh Manager, окрім виявлення та обробки подій, виконує роль постачальника даних до обробника логів Logstash. Останній збагачує та нормалізує події, готуючи їх до передачі в OpenSearch. Впровадження Logstash у цій схемі обумовлене потребою у гнучкій обробці подій, а також модливістю масштабування логіки трансформації за потреби.

На стороні OpenSearch використовується Ingest Pipeline як останній етап обробки, що дозволяє застосовувати додаткові правила трансформації пбо валідації на рівні зберігання. Після цього події індексуються та стають доступними для пошуку, візуалізації, побудови аналітичних запитів, моніторингу та подальшої автоматичної обробки через підсистеми сповіщень та виявлення аномалій.

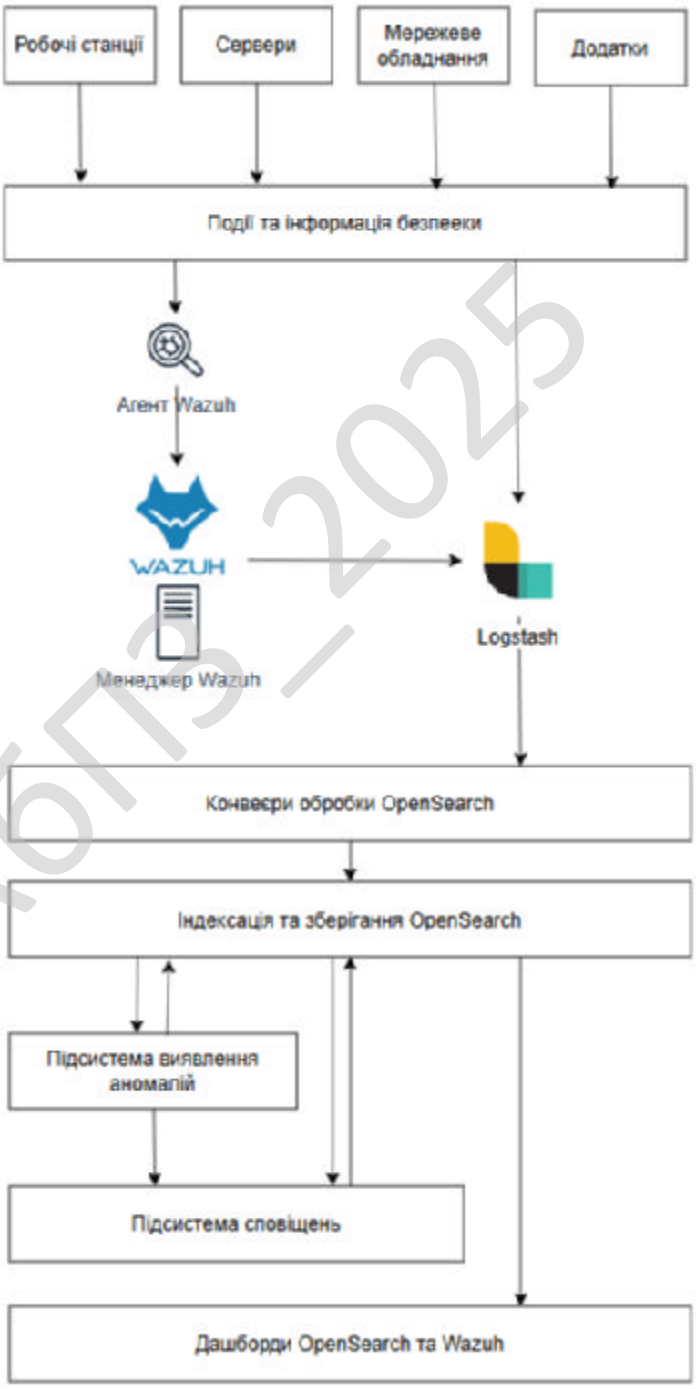

Рисунок 3.2 – Функціональна схема SIEM-системи

Нижче наведено функціональна будова та логіка роботи системи.

1. Джерела подій

В контексті SIEM-системи джерела подій відіграють ключову роль як первинні постачальники інфомації для аналізу безпеки. До таких джерел можуть належати сервери, робочі станції, маршрутизатори, мережеві комутатори, міжмережеві екрани, вебсервери, поштові служби, прикладні сервіси та системи керування доступом. Кожне з цих джерел генерує дурнали подій (логи), які відображають активність користувачів, системні процеси, помилки, спроби аутентифікації, зміни конфігурацій або підозрілі дії. У загальній архітектурі SIEM саме ці джерела становлять базовий рівень, із якого починається формування інформаційного потоку, необхідного для подальшої обробки та виявлення інцидентів.

2. Події безпеки

Події та інформація про безпеку охоплюють усі дані, що можуть бути інтерпретовані як потенційно релевантні до моніторингу стану інформаційної безпеки. Це не лише системні логи або журнали доступу, а й повідомлення IDS/IPS, сигнали з антивірусних систем, записи про зміну файлів, спроби підключення до захищених портів або виявлення підозрілих процесів. Функціонально ці дані є вхідним матеріалом для наступних етапів: агрегації, нормалізації, аналізу, кореляціх та збереження. Вони трансформуються в уніфіковані події, які можна фільтрувати, групувати, співставляти з шаблонами атак і виявляти як частину потенційного інциденту.

3. Агент Wazuh

Агенти Wazuh виконують функцію початкового збору та попередньох обробки подій на кінцевих пристроях. Вони встановлюються на ті вузли, які є джерелами подій – це можуть бути як сервери, так і робочі станції. Агент виконує збір логів, контроль цілісності файлів, моніторинг реєстру, процесів, користувацьких дій, а також виконує локальну нормалізацію подій перед передачею їх на цетралізований сервер – Wazuh Manager. Таким чином, агенти

діють як розподілена мережа сенсорів, що забезпечують масштабований, контрольований і надійний спосіб доставки подій у систему для подальшого аналізу. Вони дозволяють зменшити навантаження на центральну інфраструктуру та забезпечити вискокий рівень деталізації та точності даних.

4. Менеджер Wazuh

Wazuh Manager виконує функцію центрального компонента обробки подій у SIEM-системі. Він отримує дані від великох кількості агентів Wazuh, а також – за потреби – від зовнішніх джерел, що не мають агентів. Основне завдання менеджера полягає в аналізі, нормалізації та кореляції подій відповідно до вбудованих правил безпеки. Ці правила охоплюють типові сценарії загроз, а також дозволяють будувати власні цмови виявлення інцидентів. На основі аналізу менеджер формує повідомлення про події безпеки, класифікує їх за рівнем критичності, джерелом, типом загрози та іншими атрибутами. Крім того, Wazuh Manager веде облік активності агентів, контролює їхній статус та забезпечує централізовану політику оновлення правил. У загальній схемі він є головним процесом, який узагальнює дані з усіх точок системи, фільтрує зайве та формає уніфіковані записи для подальшого збереження.

5. Logstash

Logstash висупає проміжною обробною ланкою, що дозволяє адаптувати, трансформаувати та маршрутизувати події між джерелами та системою зберігання. У контексті цієї SIEM-архітектури Logstash отримує потоки подій від Wazuh Manager (або безпосередньо від окремих систем) і виконує додаткові дії: розбір даних за шаблонами, приведення до єдиного формату, збагачення подій метаданими, переклад кодів у зрозумілі імена, а також – за потреби – фільтрацію. Завдяки гнучкій систмі фільтрів, Logstash дозволяє розширити та модифікувати логіку обробки без зміни конфігурації агента чи менеджера. Він також підтримує надійні механізми доставки – наприклад, буферизацію у випадку тимчасової недоступності OpenSearch, таким чином, Logstash виконує роль універсального адаптера, що забезпечує сумісність, чистоту та повноту даних перед передачею

до сховища.

6. Конвеєри обробки OpenSearch (Ingest Pipelines)

Конвеєри обробки OpenSearch, або Ingest Pipelines, є вбудованим механізмом OpenSearch, який дозволяє здійснювати обробку подій безпосередньо перед їхнім збереженням. Вони приймають дані з Logstash або інших джерел через API, і перед тим як зберегти подію у відповідному індексі, виконують повну послідовність операцій: вирізання полів, перетворення форматів, геолокацію IP-адрес, розбиття тексту, а також збагачення інформацією з довідників чи зовнішніх джерел. На відміну від Logstash, який працює як окремий сервіс, Ingest Pipelines виконуються всередині OpenSearch, що дає змогу розвантажити зовнішні компоненти. Функціонально ці конвеєри відіграють роль останнього етапу нормалізації, після якого події набувають фінального вигляду та стають готовими до індексації, пошуку та аналітики. Це критично важливо для уніфікованості даних та коректної роботи аналітичних механізмів та візуалізацій.

7. Індексація та зберігання в OpenSearch

Індексація та зберігання в OpenSearch є основною функцією ядка системи, що забезпечує довготривале, масштабоване та ефективне зберігання подійі безпеки. Коли дані проходять обробку через Ingest Pipelines, вони потрапляють до відповідних індексів OpenSearch, де кожна подія зберігається як окремий документ у форматі JSON. Індекси структуруються за шаблонами (index templates), що визначають які поля повинні бути присутні, які типи даних вони мають та які політики зберігання або обмеження часу застосовуються. Індексація дозволяє здійснювати щвидкий пошук, фільтрацію, агрерацію та аналіз подій у режимі реального часу. У контексті SIEM-системи це забезпечує можливість гнучкого розслідування інцидентів, побудови звітності та автоматизованих аналітичних запитів.

8. Підсистема виявлення аномалій (Anomaly Detection)

Підсистема виявлення аномалій (Anomaly Detection Plugin) в OpenSearch дозволяє автоматично аналізувати поведінкові патерни в даних і виявляти

нетипові відхилення, які можуть свідчити про потенційні загрози. Вона використовує методи машинного навчання, зокрема алгоритм Random Cut Forest, який дозволяє будувати модель звичнох поведінки на основі історичних даних, а потім співставляти нові події з цією моделлю. Якщо нові дані не відповідають звичному шаблону, підсистема фіксує аномалію із зазначенням її рівня критичності (anomaly score), Ця функціональність особливо корисна у випадких складних, малопомітних атак, коли класичні правила не спрацбовують. Таким чином, підсистема доповнює сигнатурний аналіз поведінковим, розширюючи спектр виявлення загроз.

9. Підсистема сповіщень (Alerting)

Підсистема сповіщень (Alerting Plugin) реалізує автоматизовану реакцію на зазадлегідь визначені умови, виявлені в ході аналізу даних. Вона дозволяє створювати моніторингові правила (monitors), які постійно виконують запити до індексів та перевіряють, чи киконуються умови спрацювання – наприклад, надмірна кількість невдалих входів, висока оцінка аномалії або виявлення конкретного шаблону події. Якщо така умова виконується, система формує тригер, який може запускати одну або кілька дій – надіслати сповіщення електронною поштою, викликати webhook, надіслати повідомлення в корпоративний месенджер, або передати подію до зовнішньої SOAR-системи для автоматизованої обробки. У системі SIEM ця підсистема забезпечує критично важливий етап реагування –швидке інформування відповідальних осіб або підсистем.

10.Дашборди OpenSearch та Wazuh

Дашборди OpenSearch та Wazuh забезпечують візуалізацію, інтерпретацію та зручну взаємодію з великими обсягами даних. OpenSearch Dashboards – це веб-інтерфейс на основі Kibana, який дозволяє аналітикам та адміністраторам створювати графіки, таблиці, теплові мапи, часові діаграми та інші компоненти для моніторингу стану системи безпеки. Інтерфейс інтегрує спеціальні розширення: Wazuh App – для глбокого перегляду даних, отриманиих з агентів

Wazuh; Anomaly Detection UI – для керування моделями виявлення аномалій; Alerting – для управління сповіщеннями. Дашборди дозволяють будувати як загальні оглядові панелі для оперативного моніторингу, так і спеціалізовані – для розсліжування інцидентів або аудиту. Вони служать основним інструментом взаємодії з SIEM для оперативних команд і значною мірою визначають швидкість та ефективність реагування на події.

Функціональна схема розробленої SIEM-системи побудована з урахуванням сучасних вимог до масштабованості, гнучкості та адаптивності до різних джерел подій. Ключова особливість цієї архітектури полягає у гібридному підході до збору та обробки даних: частина подій надходить через агенти Wazuh, інша – безпосередньо через Logstash. Це дозволяє об'єднувати інформацію як з кінцевих пристроїв, так і з зовнішніж джерел, включаючи мережеве обладнання та хмарні сервіси.

На відміну від класичних SIEM-систем, які переважно базуються на жорстко визначених правилах та централізованому зборі логів, дана структура реалізує гнучку багатоступеневу обробку подій із застосуванням інжест-конвеєрів, машинного навчання для виявлення аномалій та автоматизованих механізмів сповіщення. Це не лише підвищує точність виявлення загроз, але й скорочує час реакції на інциденти. Роль кожного компонента чітко визначена, що забезпечує логічну прозорість та технічну надійність системи, а розділення функцій між агентами, брокерами обробки та аналітичним ядром дозволяє легко масштабувати рішення відповідно до потреб організації.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.3.

Рисунок 3.3 – Діаграма взаємодії процесів

Діаграма процесів ілюструє послідовний життєвий цикл події безпеки – від моменту її виникнення до реакції оператора та вдосконалення системи.

Перший критичний етап починається з підключення агентів Wazuh до всіх релевантних джерел подій: серверів, робочих станцій, хмарних сервісів тощо. Саме тут забезпечується найважливіший збір телеметріх, адже локальні агенти не лише збирають сирі журнали подій, але й виконують попередню нормалізацію та фільтрацію, зменшуючи обсяг шуму ще до передачі даних у центральний вузол.

У Wazuh Manager відбувається перший рівень інтелектуальної обробки: декодування кожного повідомлення, застостування правил кореляції та раннє виявлення шаблонів відомих атак. Ця стадія – ядро rule-based аналітики, яка миттєво реагує га класичні сигнатури й дає змогу зупинити загрозу на ранніх стадіях. Надалі події передаються до Logstash, де відбувається гнучке перетворення даних; за допомогою grok-шаблонів, mutate-операцій та збагачень метаданими уніфікується структура логів, що робить їх придатними до зберігання в індексах OpenSearch.

Унікальною рисою цієї архітектури є застосування Ingest Pipelines безпосередньо в OpenSearch. Після прийому потоків Logstash кожна подія проходить через внутрішній конвеєр, де на фінальному кроці додаються останні штрихи: коригужться тип даних, здійснюється геолокація IP-адрес і т.д. Лише після цього документи індексуються у часові індекси OpenSearch, що гарантує їхню доступність для миттжвого пошуку, агрегації та аналітики.

Аналітика реалізована двома шляхами. Перший – через OpenSearch Dashboards, який забезпечує візуальний контроль та ad-hoc пошук подій, дозволяючи аналітикам SOC буквально «побачити» мережевий стан в режимі реального часу. Другий шлях – це підсистема виявлення аномалій на основі Random Cut Forest. Вона безперервно донавчається на історичних даних та оцінює нові події за “*anomaly score*”, вказуючи на нетипові ситуації, які не вкладаються в звичні патерни. Цей метод підвищує чутилвість системи до zero-day загроз і багатоетапних атак, що розгортаються непомітно.

Коли будь-яка з обох аналітичних гілок фіксує підозру – сигнал надходить до підсистем сповіщень. Тут гнучкі монітори перевіряють умови тригерів і

запускають відправку повідомлень електронною поштою, через webhook або інтеграцію з SOAR-платформами. Цей механізм гарантує, що відповідальний аналітик або автоматизований оркестратор отримає чітке повідомлення з усіма контекстними даними, необхідними для негайного реагування.

Замкнутий цикл забезпечується звороним зв'язком: результати розслідувань та оперативних дій фіксуються аналітиком в Dashboards і використовуються для оновлення правил кореляції, допрацювання інжест-конвеєрів або донавчання RCF-моделі. Це безперервне вдосконалення гарантує, що SIEM-система не лише пристосовується до нових загроз, а й з часом підвищує точність, зменшуючи кількість хибних спрацювань і скорочуючи час від виявлення до нейтралізації інциденту.

Таким чином, після аналізу загального опису системи, структурної та функціональної схем, а також діаграми взаємодії основних процесів, доцільно перейти до розгляду блок-схем, що відображають логіку роботи ключових алгоритмів (зокрема, алгоритму виявлення аномалій RCF), а також сценаріїв, конфігураційних шаблонів та налаштувань, які забезпечують функціонування всієї системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНОІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків:

– Надходження сирих логів і подій із джерел (серверів, мережевих пристроїв, додатків).

– Попередня обробка даних: нормалізація форматів, фільтрація зайвих полів.

– Агрегація зібраних подій за визначений часовий інтервал (наприклад, 5 хвилин).

– Формування вектора ознак на основі детекторів:

- *failed_login_error* – кількість невдалих входів;
- *auth_error_count* – кількість помилок аутентифікації;
- *avg_response_time* – середній час відповіді тощо.

– Перевірка вектора на відомі сигнатури атак (rule-based):

- якщо знайдено збір за правилом – відразу генерується тривога (alert);
- якщо збірів немає – переходимо до аналізу аномалій.

– Передача вектора ознак до моделі Random Cut Forest (RCF);

– Обчислення anomaly score:

- кожне дерево лісу рахує «глибину ізоляції» нової точки;
- середня глибина перетворюється на числову оцінку аномальності.

– Порівняння отриманого anomaly score з пороговим значенням:

- якщо оцінка перевищує поріг –генерується тривога (alert);
- якщо ні – система продовжує моніторинг.

– Оновлення внутрішнього буферу моделі RCF:

- додається нова точка, видалається найстаріша за необхідності;
- за потреби виконується локальне оновлення дерев.

– Цикл завершено: система готова обробляти наступну точку.

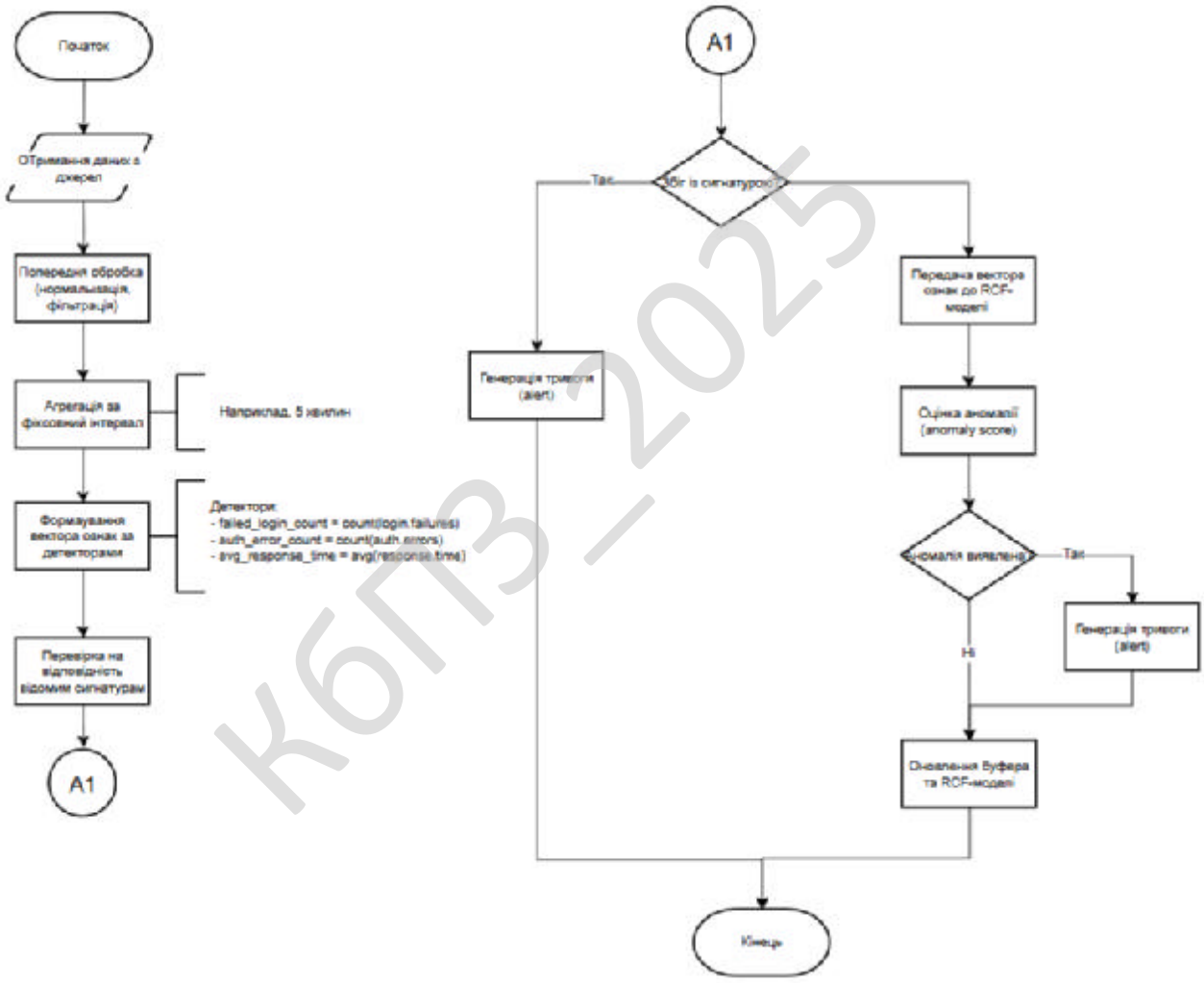

Рисунок 4.1 – Блок-схема основної програми

Опис алгоритмів функціонування системи.

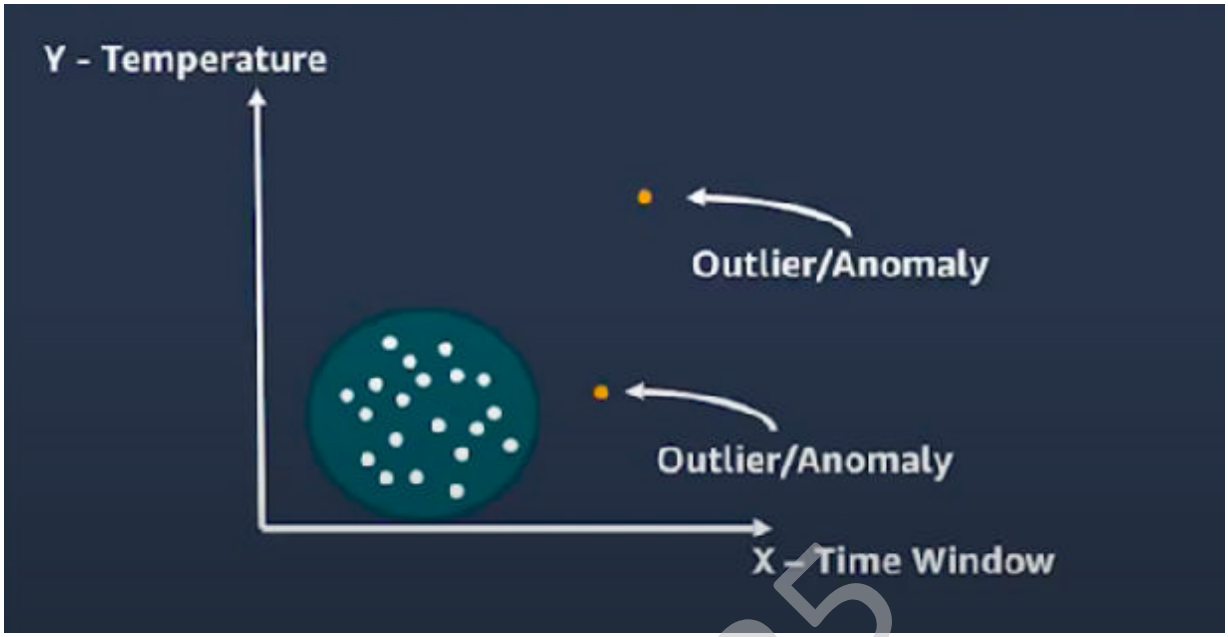

Рисунок 4.2 – Визначення аномалій на графіку на основі датасету

Алгоритм **Random Cut Forest (RCF)** – це неконтрольований (unsupervised) метод виявлення аномалій, який аналізує вхідні дані як множину точок у багатовимірному просторі. Його суть – побудувати «ліс випадкових дерев-розрізів», де кожне дерево рекурсивно розбиває простір даних на невеликі області. Розбивання робиться так: для набору точок спочатку визначають граничний прямокутник (bounding box), потім випадково обирають одну з ознак (зазвичай зважену за дисперсією) і проводять гіперплощину (random cut) під випадковим значенням в цьому вимірі.

Процедуру повторюють, доки кожна точка не опиниться в окремому листі. Аномальною вважається та точка, яку в середньому **ізолюють найшвидше** – тобто вона досягає листа на малій глибині дерева. Іншими словами, при вставці аномалії в дерево модель змінюється мінімально (дерево не «губить» багато часу на її опис). Цей принцип візуально видно на рисунку 4.3 нижче.

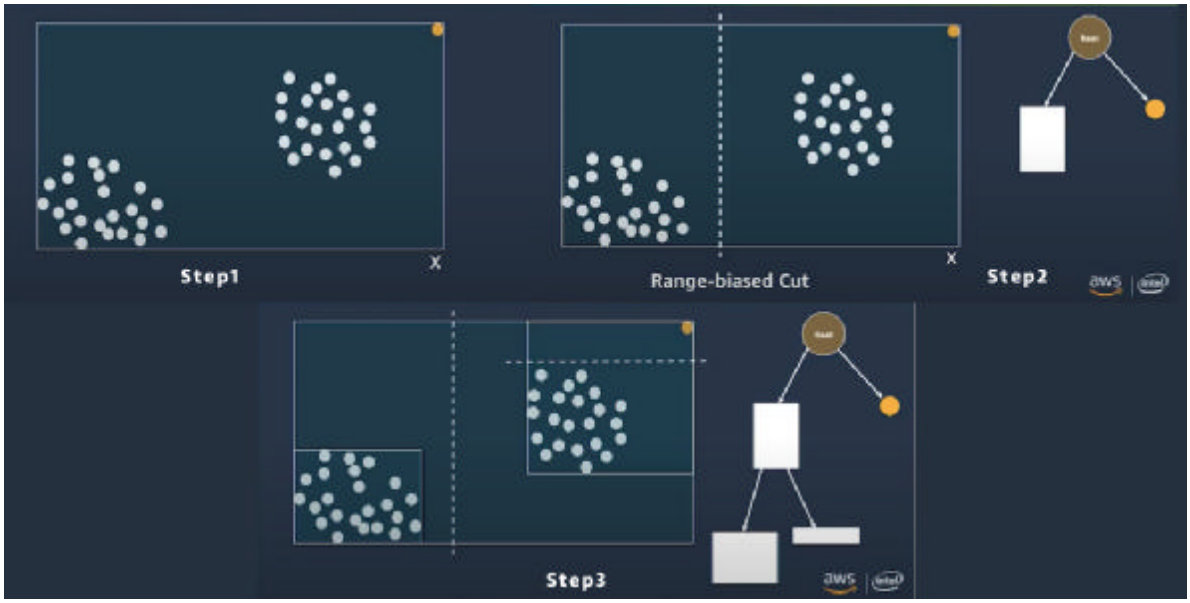

Рисунок 4.3 – Принцип поділу даних в алгоритмі RCF

При роботі з новим записом (вектором ознак) RCF умовно «вставляє» його в кожне дерево і обчислює отриману глибину (шлях від кореня до листа). Потім усереднює ці глибини по всіх деревах і формує **бальну оцінку аномалії**, що обернений глибині: короткий середній шлях дає високий бал аномалії. Таким чином, точки, які значно відрізняються від основного кластеру даних, отримують більший бал. Наприклад, в Amazon SageMaker пропонують вважати аномальними ті записи, у яких аномальний бал перевищує середнє більш як на 3 сигми.

Формування моделі та підготовка даних

Побудова моделі RCF традиційно починається зі збору вибірки історичних даних (наприклад, останніх записів журналу подій) і випадкового відбору з неї підмножини для кожного дерева. В реальному часі це може бути організовано через **ковзаюче вікно**: модель підтримується на основі останнього фрагмента потоку даних. Наприклад, у OpenSearch-детекторі аномалій кожна нова агрегація подій надходить по черзі, і RCF поступово оновлює свій «скетч» (приблизне зображення щільності даних) для реального часу.

У контексті SIEM дані зазвичай отримуються як журнали подій з обмеженим набором полів. RCF потребує числових вхідних векторів, тому перед аналізом проводять інженерію ознак: витягують та перетворюють значення полів журналу в

кількісні метрики. Зокрема, часто обчислюють **агреговані ознаки** – наприклад, лічильники подій чи статистики по часу. У OpenSearch при налаштуванні детектора аномалій кожну ознаку (feature) задають як агрегат (count, avg, sum, min, max) над певним полем індексу. Наприклад, можна відстежувати **кількість вдалих логінів за хвилину** або **середню кількість помилкових спроб входу за годину**. Такі агрегації по суті формують числові вектори, які й передаються в RCF-модель.

Оцінка аномалії

Після формування лісу дерев RCF для кожного нового вхідного вектора (наприклад, нового стану системи або агрегованої метрики) обчислюється оцінка аномальності. Вона будується так: точку «вставляють» в кожне дерево, вимірюють глибину її розміщення, потім усереднюють по всіх деревах. Чим **менше середня глибина**, тим рідше ця подія траплялася в тренувальних даних і тим вище її аномальний бал. Фактично RCF визначає аномалію як подію з низькою щільністю даних навколо неї. За результатами моделі кожному запису присвоюється число (аномальний бал), і звичайно вводять поріг (наприклад, значення вище певного рівня, як-от три стандартних відхилення від середнього для визначення, що саме є аномалією.

В OpenSearch вихідні значення включають **anomaly grade** (оціночний коефіцієнт аномалії) та **confidence score**. Аномальний бал (grade) змінюється від 0 (норма) до 1 (сильна аномалія) – значення 0 означає, що поточна точка в межах очікуваного (розподіл щільності не рідкісний), а будь-яке невиконується означає підозрілий випадок. Таким чином RCF сигналізує про те, що в нових даних з’явилося щось рідкісне у порівнянні з навчальною вибіркою.

На рисунку 4.4 наведено блок-схему, яка відображає процес побудови дерева в рамках початкового навчання моделі Random Cut Forest. На початковому етапі здійснюється перевірка вхідного набору даних на валідність, після чого модель послідовно проходить через процедуру ствîрення окремих дерев. Для кожного дерева обирається випадкова підмножина даних з буфера, що імітує незалежне навчання компонентів. На основі обраних точок рекурсивно

генеруються випадкові гіперплощини – так звані «розрізи», які поділяють простір даних на підмножини, формуючи вузли дерева. Цей процес продовжується доти, доки в оркемих гілках дерева не залишиться поодиноких елементів. У результаті кожне дерево модулі фіксує свою унікальну інтерпретацію структури вхідних даних. Після завершення побудови всіх дерев вважається готовою до інференсу – тобто подальшого використання для виявлення аномалій у нових точках, що надходитимуть в систему.

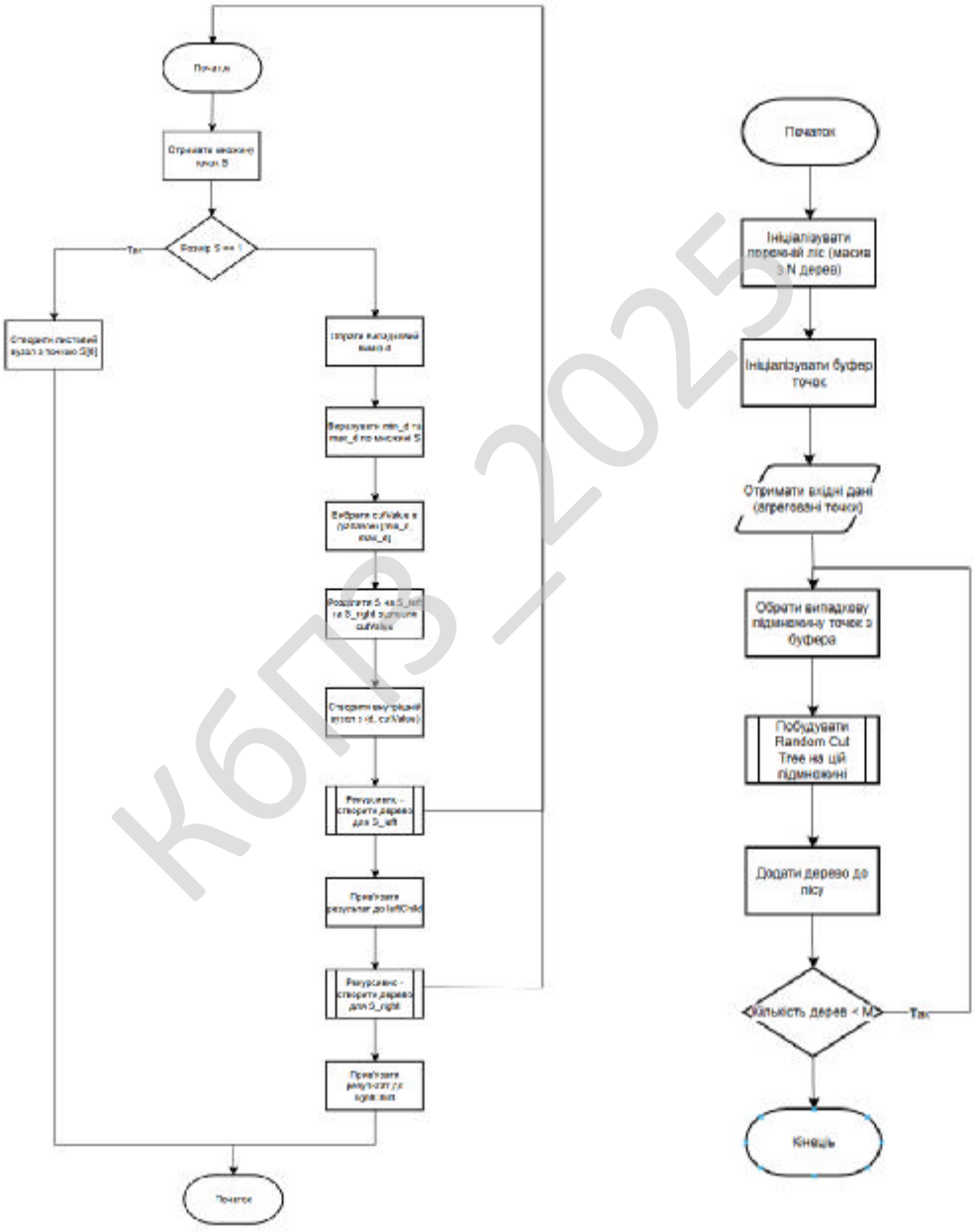

Рисунок 4.4 – Блок-схеми алгоритму RCF (побудова дерева, попереднє навчання)

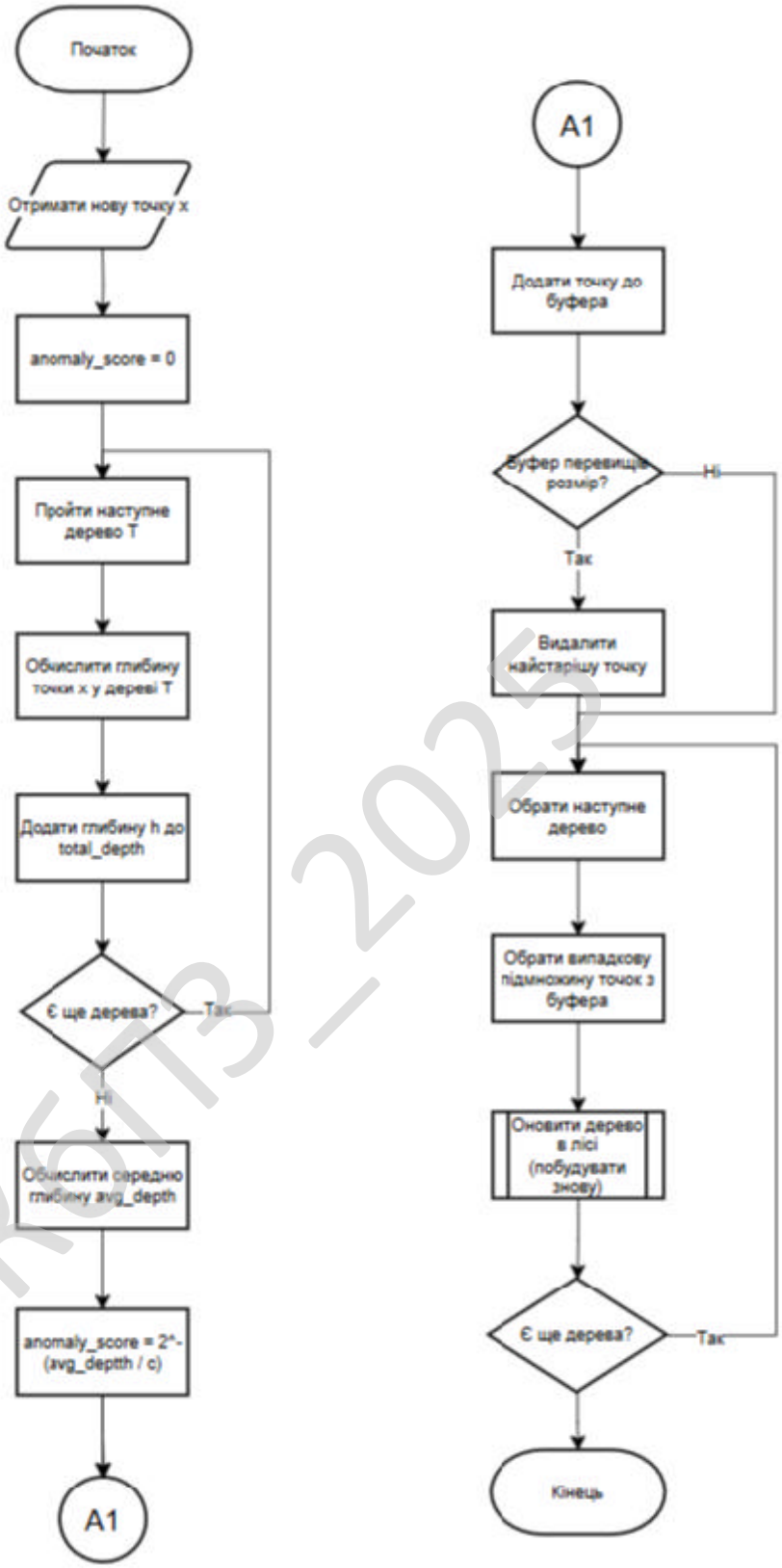

Рисунок 4.5 – Блок-схема оцінки аномалії за допомогою моделі RCF та донавчання моделі

На рисунку 4.5 зображено блок-схему процесу оцінки аномалії нової точки в моделі Random Cut Forest та подальшого донавчання. Робота підпрограми розпочинається з отримання нової точки з агрегованих даних корпоративнох мережі. Ця точка передається до моделі, яка містить попередньо побудований ансамбль дерев.

Кожне дерево окремо оцінює, наскільки ізольованою є нова точка у своїх структурі: для цього відбувається віртуальний прохід деревом, під час якого точка рекурсивно «розсікається» випадковими гіперплощинами, доки не досягає листа. Кількість таких розрізів до ізоляції – глибина – і є основою для розрахунку ізоляційної оцінки. Після отримання глибин від усіх дерев обчислюється середнє значення, і на його основі виводиться *anomaly score* – числовий показник, що відображає ступінь незвичності точки: чим менше потрібно розрізів для ізоляції, тим вища її аномальність.

Якщо оцінка перевищує заданий поріг, точка можн бути позначена як аномалія. Незалежно від цього, нова точка додається до буфера даних, який використовується для поступового оновлення моделі. Якщо буфер перевищує свій максимальний розмір, найстаріша точка з нього видаляється. Далі виконується оновлення дерев: кожне дерево випадково замінює одну зі своїх точок новою, і на основі нової підмножини точок відповідне дерево перебудовується. Таким чином, модель адаптується до змін у даних і зберігає актуальність навіть при довготривалому використанні.

Особливості безпекових подій і онлайн-аналіз

Інженерія ознак: Події в SIEM часто мають обмежену кількість числових полів і багато категоріальних (IP-адреси, користувачі, типи подій тощо). Тому замість прямих значень у RCF передають агреговані метрики. Наприклад, як ознаку можна обрати *count()* певних подій за останню хвилину або *average()* критичності за останню годину. Такий підхід враховує структуру журналів, але дає RCF на вхід лиш числа.

Часові вікна: Аналіз подій безпеки завжди прив'язаний до часу. У практиці застосовують ковзаючі вікна – тобто RCF-модель тренують і оновлюють на «історичній» вибірці останніх N подій або за фіксований інтервал часу. Це дозволяє фокусуватися на сучасних патернах поведінки та уникати застарілого контексту.

Онлайн-аналіз: RCF-підійде для потокових даних. Завдяки алгоритму **reservoir sampling** або підтримці фіксованого розміру вибірки RCF може «запам'ятовувати» лише нещодавні дані. Кожне нове спостереження одразу обробляється (вставляється в дерева), і старі дані «витісняються» з моделі. Це критично у SIEM для моніторингу в реальному часі – система в змозі оцінити аномалії майже миттєво при надходженні нових подій.

RCF як доповнення до сигнатурних методів

Random Cut Forest додає рівень **поведінкового аналізу**, доповнюючи традиційні сигнатурні методи в SIEM. На відміну від сигнатурних підходів, що орієнтуються на чітко відомі вказівники атаки (hash-функції файлів, IP-адреси, конкретні шаблони мережевої активності), аномалійна детекція шукає **невідомі** чи нетипові відхилення в поведінці системи. Наприклад, вона може спрацювати на ситуаціях, коли користувач раптом виконує вхід у незвичний час або починається раптовий потік нових IP-адрес.

Водночас слід враховувати, що аномальні алгоритми, включно з RCF, зазвичай мають вищий відсоток хибнопозитивних спрацьовувань. Тому їх використовують **не замість, а поряд із** сигнатурним детектуванням. Сигнатурні системи дають високоточні попередження щодо відомих загроз (низький рівень помилкових тривог), тоді як RCF підсвічує нові чи нетипові патерни, які згодом аналізують глибше. У практичних SIEM-комплексах це часто реалізують так: спершу фільтрують відомі IOC/сигнатури, а потім результати RCF-аналізу додають як контекст або фактор підтвердження наявності невідомої активності. Таке поєднання дозволяє отримати ширшу картину загроз і оперативно реагувати як на традиційні, так і на нетипові атаки.

4.2 Захист розробленого програмного забезпечення

Зважаючи на мережеву природу системи, що розробляється у даній роботі, для забезпечення захищеного обміну даними між компонентами системи використовується протокол HTTPS (HyperText Transfer Protocol Secure). Це поширений стандарт безпечної передачі інформації в Інтернеті та між веб-додатками, який базується на поєднанні протоколу HTTP та криптографічного протоколу TLS (Transport Layer Security).

Протокол HTTPS дозволяє шифрувати передані дані, щоб вони не могли бути отриманими або зміненинми у випадку перехоплення пакетів даних сторонніми особами. Це особливо важливо при обміні конфіденційною інформацією, зокрема логами, сповіщеннями про події і службовими даними між агентами на кінцевих точках та центральним сервером. Також шифрується трафік між веб-інтерфейсом системи та користувачами (адміністратором, аналітиками тощо).

Для забезпечення безпечної комунікації між сторонами використовуються набір криптографічних алгоритмів симетричного та асиметричного щифрування, а також алгортими цифрого підпису та обміну ключами. Спільна комбінація протоколів обирається сторонами автоматично в залежності від того, які алгоритми підтримуються кожною зі сторін.

Термінологія та позначення

Асиметричне шифрування – криптографічна операція, при якій для шифрування використовується відкритий ключ, а для розшифрування – відповідний закритий (наприклад, RSA).

Симетричне шифрування – криптографічна операція, при якій і шифрування, і розшифрування здійснюється оддним і тим самим ключем (наприклад, AES).

Обмін ключами – протокол, що дозволяє двом сторонам безпечно домовитися про спільний симетричний ключ (наприклад, DH – протокол Діффі-Геллмана – або його варіація ECDHE).

Цифровий підпис – криптографічна система, яка дозволяє перевірити, що сертифікат або повідомлення створено власником відповідного закритого ключа (наприклад, RSA, ECDSA).

Відкритий ключ – бітова послідовність фіксованої довжини, яка використовується для шифрування або перевірки цифрового підпису в рамках асиметричного шифрування (наприклад, RSA, ECDSA) та не є секретною.

Закритий ключ – бітова послідовність фіксованої довжини, яка зберігається в таємниці та використовується для розшифрування або створення цифрового підпису.

Цифровий сертифікат – структурований файл, що містить відкритий ключ, інформацію про його власника (наприклад, доменне ім’я або організацію), термін дії та підпис центру сертифікації.

Центр сертифікації (CA) – авторитетна організація або система, що випускає цифрові сертифікати, підтверджуючи справжність відкритих ключів шляхом підпису власним закритим ключем.

Ключ сесії – бітова послідовність, згенерована клієнтом і сервером під час обміну ключів (наприклад, за допомогою певної варіації протоколу Діффі-Геллмана), яка використовується для симетричного шифрування даних.

Ініціалізаційний вектор (IV) – випадково згенерована бітова послідовність, яка забезпечує унікальність кожного сеансу при використанні симетричних алгоритмів (наприклад, AES у режимі CBC або GCM).

Еліптична крива – це математична структура яка визначає набір точок, що задовільняють певному рівнянню та дозволяють виконувати криптографічні операції з меншою довжиною ключа порівняно з RSA, зберігаючи високий рівень безпеки.

Повідомлення – дані, які передаються між сторонами під час встановлення TLS-з’єднання або в процесі шифрування.

Загальні положення

Під час встановлення з’єднання через HTTPS клієнт перевіряє цифровий сертифікат сервера, щоб переконатися, що сервер справжній. Сертифікат містить відкритий ключ (наприклад, RSA) та засвідчується центром сертифікації (CA) шляхом накладення цифрового підпису. Клієнт перевіряє справжність сертифіката та вирішує чи продовжувати комунікацію.

В разі підтвердження справжності сертифіката відбувається обмін ключами. Для цього використовується асиметричне шифрування – наприклад, алгоритми RSA або ECDH (протокол Діффі-Геллмана на еліптичних кривих). На цьому етапі сторони погоджуть спільний секрет, який виконує роль ключа у симетричному шифруванні.

Коли спільний ключ обчислено, з’єднання переходить на симетричне шифрування – наприклад, з використанням AES (Advanced Encryption Standard) або ChaCha20. Симетричне шифрування дозволяє швидко та безпечно передавати дані між сторонами.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ ВИЯВЛЕННЯ ІНЦИДЕНТІВ БЕЗПЕКИ З ЕЛЕМЕНТАМИ ШТУЧНОГО ІНТЕЛЕКТУ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

У межах виконання бакалаврської дипломної роботи було реалізовано та впроваджено систему виявлення і реагування на інциденти безпеки (SIEM) та протидії кібератакам в комп'ютерних мережах із використанням технологій штучного інтелекту для підвищення ефективності обробки подій. Інтерфейс розробленої системи наведено на рисунку 5.1.

Розроблена SIEM-система складається з таких функціональних модулів:

– Модуль збору даних за допомогою агентів Wazuh із серверних та клієнтських систем.

– Система попередньої обробки та нормалізації подій безпеки.

– Механізм виявлення аномалій на основі алгоритму Random Cut Forest (RCF), інтегрованого через OpenSearch Anomaly Detection Plugin.

– Підсистема генерації оповіщень та аналітичних звітів.

– Інтерфейс управління інцидентами.

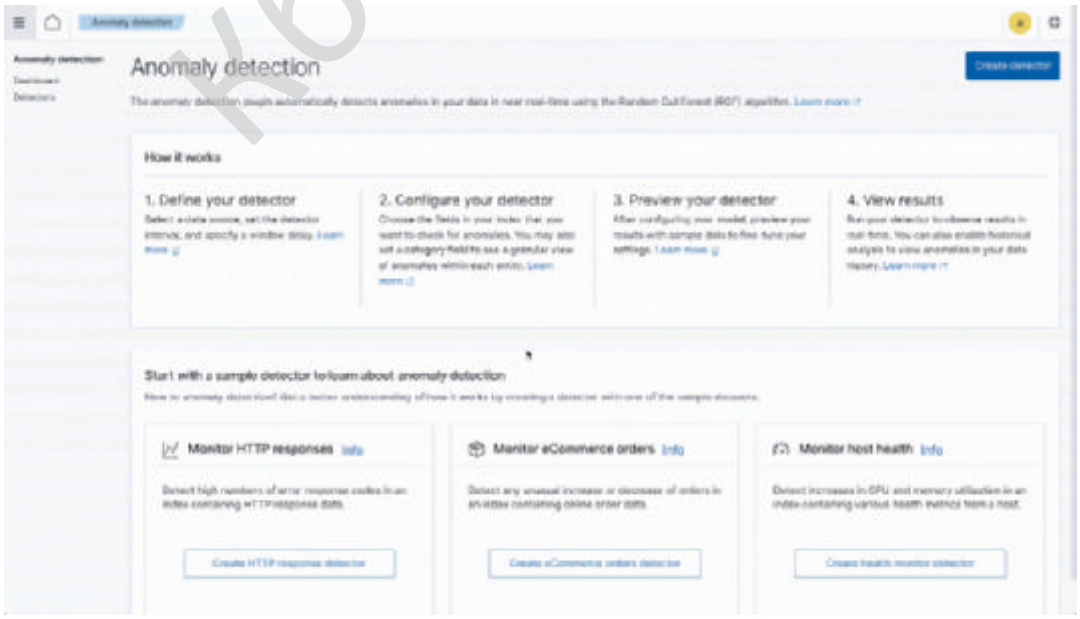

Рисунок 5.1 – Головне вікно розробленої SIEM-системи із ШІ

Основні етапи методики впровадження включають:

Підготовка середовища

На початковому етапі було розгорнуто базову інфраструктуру для роботи системи:

– Сервер збору даних і керування агентами Wazuh.

– Кластер OpenSearch для зберігання та аналізу подій.

– Додаткові компоненти для виявлення аномалій і візуалізації даних (Kibana/OpenSearch Dashboards).

Було проведено попередню оцінку апаратних ресурсів і обсягів даних для визначення оптимальних параметрів налаштування системи.

Налаштування збору та обробки даних

На цьому етапі були встановлені агенти Wazuh на сервери та робочі станції для збору системних логів, подій безпеки та журналів роботи мережевих пристроїв. Дані передавалися на центральний сервер, де виконувалася їх нормалізація та збагачення.

Впровадження модуля виявлення аномалій

Для виявлення аномальних подій у реальному часі було інтегровано плагін OpenSearch Anomaly Detection, що використовує модель Random Cut Forest (RCF).

Алгоритм RCF було обрано через такі переваги:

– Висока ефективність обробки потокових даних.

– Можливість працювати з неповними та зашумленими даними.

– Підтримка виявлення як одиничних, так і групових аномалій.

Процес налаштування включав:

– Вибір джерел даних для аналізу (наприклад, події входу в систему).

– Налаштування фільтрів для передобробки даних.

– Побудову детекторів аномалій для окремих полів або груп показників.

Приклади налаштування детекторів наведено на рисунку 5.2 та рисунку 5.3.

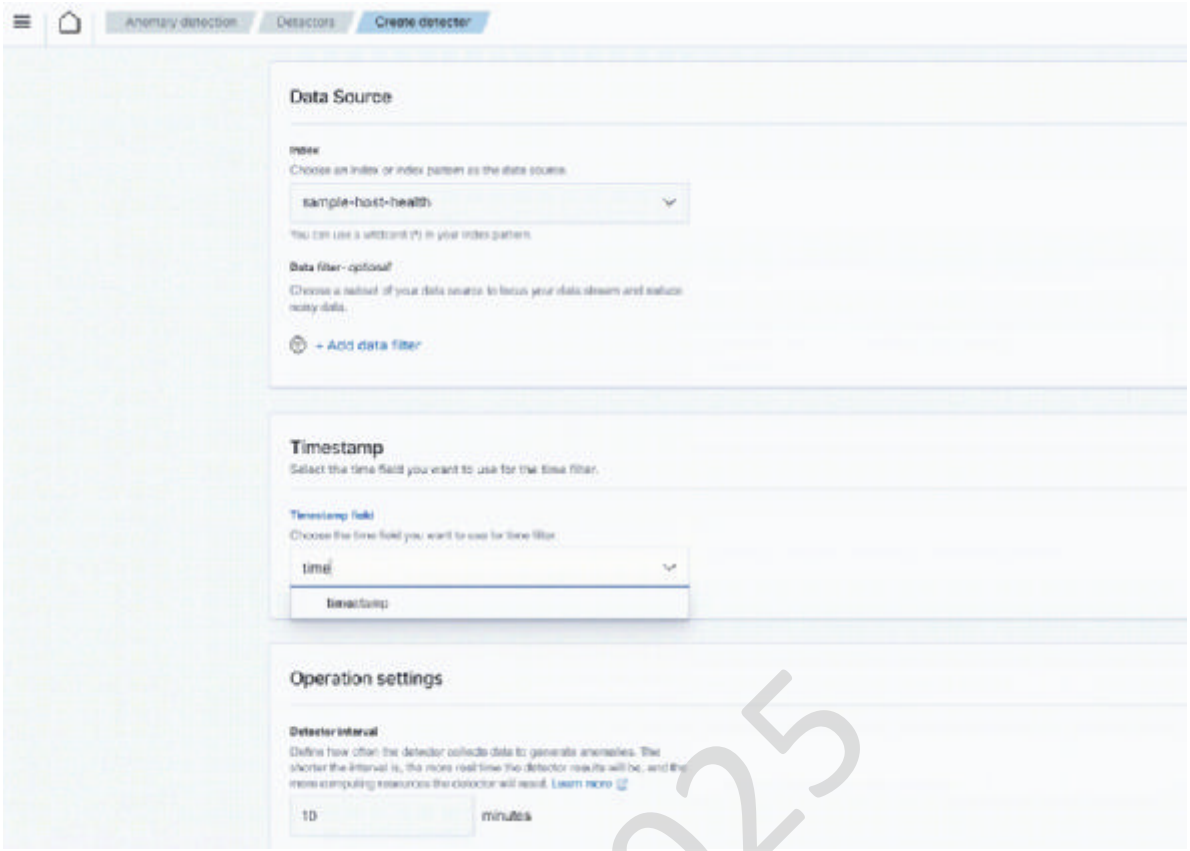

Рисунок 5.2 – Створення детектора аномалій

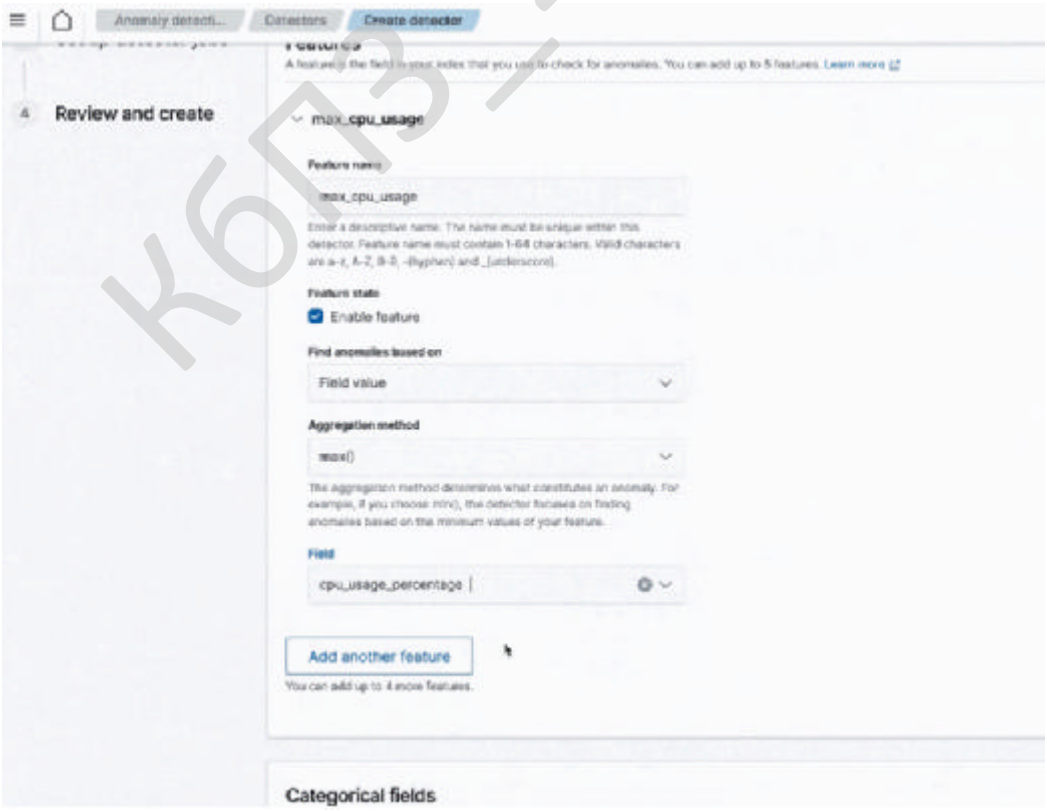

Рисунок 5.3 – Налаштування моделі детектора

Навчання моделі

Після налаштування детектори проходили етап початкового навчання (pre-training) на історичних даних, що дозволяло сформувати базові профілі нормальної активності.

Перевагою даного підходу є можливість адаптації моделі без явної потреби в ручному зазначенні правил і ознак для кожного типу подій.

Генерація інцидентів та реагування

Після виявлення аномальної події детектор автоматично створював інцидент безпеки, який відображався в інтерфейсі OpenSearch Dashboards та дублювався у Wazuh Manager для подальшої обробки і реагування. Алгоритм обробки інцидентів передбачав:

– Присвоєння рівня критичності інциденту на основі типу аномалії.

– Повідомлення відповідальних осіб через електронну пошту або месенджери.

– Створення автоматизованих відповідей за допомогою сценаріїв реагування (SOAR-механізмів).

Тестування і валідація

Було проведено комплексне тестування системи з акцентом на:

– Час виявлення інцидентів.

– Якість класифікації подій як нормальних або аномальних.

– Відсутність хибнопозитивних та хибнонегативних спрацьовувань у контрольованих сценаріях.

– Стійкість системи до високого навантаження на обробку подій.

Тестування проводилося як у форматі стандартних сценаріїв використання, так і через моделювання реальних атак.

Переваги використання штучного інтелекту в SIEM

Використання ШІ у складі SIEM-системи забезпечило:

– Зниження кількості помилкових сповіщень у порівнянні з класичними сигнатурними методами.

– Можливість виявлення раніше невідомих (zero-day) атак.

– Реальну адаптацію до змін у поведінці інформаційної системи без необхідності постійного ручного оновлення правил.

– Зниження навантаження на аналітиків безпеки завдяки автоматизованому аналізу аномалій.

На рисунку 5.4 зображено форму авторського права, що застовоіується до розробленої системи. Система побудована з використанням відкритого програмного забезпечення (Open Source), що поширюється на умовах вільних ліцензій. Компоненти системи базуються на загальнодоступному коді, що забезпечує можливість вивчення, модифікації та вільного розповсюдження з дотримання умов відповідних ліцензій (Apache License 2.0, GNU General Public License v3, Elastic License v2). Такий підхід сприяє прозорості, повторному використанні коду та розвитку спільнот розробників програмних рішень з відкритим кодом. Усі використані компоненти були ретельно перевірені на відповідність до вимог безпеки та стабільності.

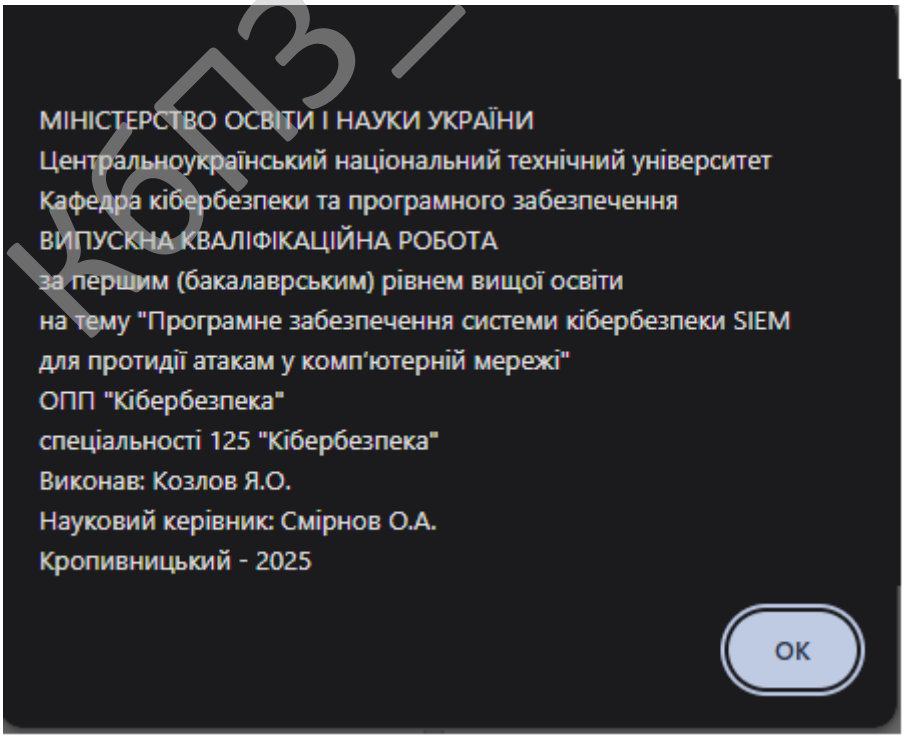

Рисунок 5.4 – Довідка

6 ОСНОВНІ ВИСНОВКИ

Програмний комплекс, створений в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначений для протидії атакам у комп'ютерних мережах. Ефективність системи кібербезпеки SIEM покращена за рахунок інтегрованих методів машинного навчання, зокрема виявлення аномалій.

Рішення поставденох задачі передбачало виконання наступних задач:

– Було проведено огляд існуючих SIEM-платформ з інтеграцією методів машинного навчання, зокрема Wazuh та OpenSearch з використанням алгоритму Random Cut Forest для виявлення аномалій.

– Досліджено архітектури та можливості налаштування підсистем збору, обробки й індексації подій із застосуванням Wazuh Manager, Logstash та конвеєрів OpenSearch Ingest.

– На основі отриманих результатів формалізовано блок-схеми алгоритмів виявлення загроз, сценарії запуску, конфігураційні шаблони та налаштування SIEM-системи на базі Wazuh з використанням OpenSearch та OpenSearch Anomaly Detection.

У результаті проведеного дослідження будо наочно продемонстровано, що інтеграція методів штучного інтелекту, зокрема моделі Random Cut Forest (RCF), у склад SIEM-систем суттєво підвищує ефективність виявлення інцидентів інформаційної безпеки. Застосування алгоритмів машинного навчання дозволяє не лише реагувати на події, які чітко підпадають задалегідь визначені правила або порогові значення, а й виявляти складніші птаерни нетипової поведінки користувачів, систем або сервісів, що не могли бути передбачені традиційними методами, які зазвичай базуються на наборі правил. Особливо важливою є здатність моделі RCF працювати в режимі реального часу та адаптуватися до

потокових даних без необхідності повного переобчислення моделі, що забезпечує оперативність у динамічному середовищі.

Розроблені в ході дипломної роботи конфігурації та сценарії забезпечують можливість централізованого та гнучкого керування потоками подій у корпоративній мережі, автоматичного виявлення нетипової поведінки без необхідності ручного втручання в правила кореляції. Інтерфейс OpenSearch Dashboards у поєднанні з плагіном Wazuh App та модулем Anomaly Detection UI забезпечує інтуїтивно зрозумілий та зручний доступ для SOC-аналітиків без глибоки знань у сфері ML.

При розробчі сценаріхв було застосовано модульний підхід із чітким поділом на компоненти: агенти Wazuh, менеджер, проміжний обробник Logstash, індексація та аналітика, підсистеми виявлення аномалій та сповіщень. Така арзітектура відповідає сучасним практикам побудови комерційних SIEM-рішень та дає змогу мінімізувати час розгортання, стандартизувати процедури, а також зменшити вартість підтримки.

Крім того, впровадження інтелектуальних методів дозволяє зменшити навантаження на аналітиків центрів моніторингу безпеки (SOC), особливо в частині первинного розбору інцидентів (triage). Це сприяє зниженню ефекту інформаційної втоми (alert fatigue), коли фахівці змушені обробляти велику кількість хибнопозитивних або нерелевантних сповіщень. Таким чином, система стає не лише точніщою, а й ефективнішою в плані використання людських ресурсів.

Загалом, створена SIEM-платформа підтверджує обґрунтованість вибору архітектури та алгоритмів, повністю відповідає технічному завданню і має значний потенціал до подальшого розвитку. Зокрема, передбачено розширення функціоналу за рахунок інтеграції прогнозного аналізу загроз та SOAR-оркестрації для повного циклу реагування на інциденти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SIEM: Security Information & Event Management Explained | Splunk [Електронний ресурс]. Режим доступу: https://www.splunk.com/en_us/blog/learn/siem-security-information-event-management.html

2. Gartner. Magic Quadrant for Security Information and Event Management [Електронний ресурс]. Режим доступу: https://www.gartner.com/en/conferences/emea/security-risk-management-uk/sessions/detail/3473973-Magic-Quadrant-and-Critical-Capabilities-for-SIEM

3. Смірнова, Т., Константинова, Л., Конопліцька-Слободенюк, О., Козлов, Я., Кравчук, О., Козірова, Н., & Смірнов, О. (2024). ДОСЛІДЖЕННЯ СУЧАСНОГО СТАНУ SIEM-СИСТЕМ [Електронний ресурс]. Режим доступу: https://doi.org/10.28925/2663-4023.2024.25.618

4. Смірнов О.А. Козлов Я.О., Смірнова Т.В. «Дослідження застосування SIEM-систем для забезпечення кібербезпеки та захисту інформації». II Міжнародна науково-практична Інтернет-конференція «Інновації та перспективні шляхи hозвитку інформаційних технологій (ІПШРІТ-2023)» м.Черкаси 6 грудня 2023 року – Черкаси: ЧДТУ.– 2023. – С.251-252. Режим доступу: https://itp.chdtu.edu.ua/ipshrit-2023/

5. Козлов Я.О., Смірнова Т.В., Смірнов О.А. «Дослідження SIEM-систем для забезпечення кібербезпеки». VII міжнародна науково-практична конференція “Інформаційна безпека та комп’ютерні технології” до 30-ти річчя кафедри кібербезпеки та програмного забезпечення, м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 26. Режим доступу: https://dspace.kntu.kr.ua/handle/123456789/13506

6. What is the MITRE ATT&CK Framework? | IBM [Електронний ресурс]. Режим доступу: https://www.ibm.com/topics/mitre-attack/

7. Election Security Spotlight – Signature-Based vs Anomaly-Based Detection | Center for Internet Security (CIS) [Електронний ресурс]. Режим доступу: https://www.cisecurity.org/insights/spotlight/cybersecurity-spotlight-signature-based-vs-anomaly-based-detection

8. Splunk — An Overview [Електронний ресурс]. Режим доступу: https://sarada-sastri.medium.com/splunk-an-overview-f5473441685

9. Webinar: Adaptive Thresholds and Anomaly Detection: Unleash the Power of Machine Learning for Improved Operations with Splunk ITSI | Splunk [Електронний ресурс]. Режим доступу: https://www.splunk.com/en_us/form/unleash-the-power-of-machine-learning.html

10. Козлов Я.О., Смірнов С.А., Кравчук О.В., Смірнов О.А. «Дослідження особливостей сучасних SIEM-систем». VII міжнародна науково-практична конференція “Інформаційна безпека та комп’ютерні технології” до 30-ти річчя кафедри кібербезпеки та програмного забезпечення, м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 54. Режим доступу: https://dspace.kntu.kr.ua/handle/123456789/13506

11. SIEM from Elastic [Електронний ресурс]. Режим доступу: https://www.elastic.co/security/siem

12. Elastic Security Solution [Електронний ресурс]. Режим доступу: https://www.elastic.co/security

13. Visual event analyzer | Elastic Security Solution [8.12] | Elastic [Електронний ресурс]. Режим доступу: https://www.elastic.co/guide/en/security/current/visual-event-analyzer.html

14. ELK Stack Architecture Deep-Dive [Електронний ресурс]. Режим доступу: https://medium.com/dataseries/elk-stack-architecture-deep-dive-41168732f0e3

15. Elasticsearch [Електронний ресурс]. Режим доступу: https://www.elastic.co/elasticsearch

16. Kibana [Електронний ресурс]. Режим доступу: https://www.elastic.co/guide/en/kibana/current/index.html

17. Logstash [Електронний ресурс]. Режим доступу: https://www.elastic.co/docs/logstash

18. Beats (Filebeat, Metricbeat, Packetbeat, Winlogbeat etc) [Електронний ресурс]. Режим доступу: https://www.elastic.co/beats

19. X-Pack Anomaly Detection: multi-metric equals multivariate? - Kibana - Discuss the Elastic Stack [Електронний ресурс]. Режим доступу: https://discuss.elastic.co/t/x-pack-anomaly-detection-multi-metric-equals-multivariate/245451

20. Implementing Anomaly Detection in the ELK Stack: A Practical Guide for Data Analysts [Електронний ресурс]. Режим доступу: https://eyer.ai/blog/implementing-anomaly-detection-in-the-elk-stack-a-practical-guide-for-data-analysts/

21. Оголошення про повернення Elasticsearch та Kibana до відкритого коду (додано AGPL поруч із ELv2 і SSPL) [Електронний ресурс]. Режим доступу: https://www.elastic.co/blog/elasticsearch-is-open-source-again

22. QRadar SIEM [Електронний ресурс]. Режим доступу: https://www.ibm.com/products/qradar-siem

23. QRadar | IBM [Електронний ресурс]. Режим доступу: https://www.ibm.com/qradar

24. Overview | Wazuh [Електронний ресурс]. Режим доступу: https://wazuh.com/platform/overview/

25. Wazuh - The Open Source Security Platform. Unified XDR and SIEM protection for endpoints and cloud workloads [Електронний ресурс]. Режим доступу: https://github.com/wazuh/wazuh?tab=readme-ov-file#wazuh

26. Wazuh License [Електронний ресурс]. Режим доступу: https://github.com/wazuh/wazuh/blob/master/LICENSE

27. Architecture - Getting started with Wazuh [Електронний ресурс]. Режим доступу: https://documentation.wazuh.com/current/getting-started/architecture.html

28. Getting started - Wazuh server API [Електронний ресурс]. Режим доступу: https://documentation.wazuh.com/current/user-manual/api/getting-started.html

29. OpenSearch integration. Wazuh documentation [Електронний ресурс]. Режим доступу: https://documentation.wazuh.com/current/integrations-guide/opensearch/index.html

30. Enhancing IT security with anomaly detection in Wazuh | Wazuh [Електронний ресурс]. Режим доступу: https://wazuh.com/blog/enhancing-it-security-with-anomaly-detection/

31. Microsoft Sentinel – хмарне SEIM-рішення [Електронний ресурс]. Режим доступу: https://www.microsoft.com/uk-ua/security/business/siem-and-xdr/microsoft-sentinel/

32. Moving to Next-Gen SIEM with Microsoft Sentinel [Електронний ресурс]. Режим доступу: https://www.microsoft.com/insidetrack/blog/moving-to-next-generation-siem-at-microsoft-with-microsoft-azure-sentinel/

33. What is Microsoft Sentinel? [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/en-us/azure/sentinel/overview?tabs=defender-portal

34. Anomalies detected by the Microsoft Sentinel machine learning engine | Microsoft Learn [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/en-us/azure/sentinel/anomalies-reference

35. OpenSearch. [Електронний ресурс]. Режим доступу: https://opensearch.org/

36. Козлов Я.О., Козірова Н.Л., Смірнов О.А. «Дослідження структури та принципу роботи SIEM-системи». VII міжнародна науково-практична конференція “Інформаційна безпека та комп’ютерні технології” до 30-ти річчя кафедри кібербезпеки та програмного забезпечення, м. Кропивницький. 1

листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 59 Режим доступу: Режим доступу: https://dspace.kntu.kr.ua/handle/123456789/13506

37. Supported algorithms – OpenSearch Documentation [Електронний ресурс]. Режим доступу: https://docs.opensearch.org/docs/2.6/ml-commons-plugin/algorithms/

38. About Security Analytics – OpenSearch Documentation [Електронний ресурс]. Режим доступу: https://docs.opensearch.org/docs/latest/security-analytics/

39. Machine Learning – OpenSearch Documentation [Електронний ресурс]. Режим доступу: https://docs.opensearch.org/docs/latest/ml-commons-plugin/

40. Curse of dimensionality in Machine Learning [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/curse-of-dimensionality-in-machine-learning/

41. 8 Anomaly Detection Algorithms to Know | Built In [Електронний ресурс]. Режим доступу: https://builtin.com/machine-learning/anomaly-detection-algorithms

42. Anomaly Detection | Graylog [Електронний ресурс]. Режим доступу: https://go2docs.graylog.org/current/what_more_can_graylog_do_for_me/anomaly_detection.html

43. Graylog Security Anomaly Detection: Metrics Ease the Workload [Електронний ресурс]. Режим доступу: https://graylog.org/post/graylog-security-anomaly-detection-metrics-ease-the-workload

44. How RCF Works - Amazon SageMaker AI [Електронний ресурс]. Режим доступу: https://docs.aws.amazon.com/sagemaker/latest/dg/rcf_how-it-works.html

45. Random Cut Forest (RCF) Algorithm - Amazon SageMaker AI [Електронний ресурс]. Режим доступу: https://docs.aws.amazon.com/sagemaker/latest/dg/randomcutforest.html

46. Using Random Cut Forests for real-time anomaly detection in Amazon OpenSearch Service | AWS Big Data Blog [Електронний ресурс]. Режим доступу:

https://aws.amazon.com/blogs/big-data/using-random-cut-forests-for-real-time-anomaly-detection-in-amazon-opensearch-service

47. Anomaly detection in Amazon OpenSearch Service - Amazon OpenSearch Service [Електронний ресурс]. Режим доступу: https://docs.aws.amazon.com/opensearch-service/latest/developerguide/ad.html

48. GitHub – aws/random-cut-forest-by-aws: An implementation of the Random Cut Forest data structure for sketching streaming data, with support for anomaly detection, density estimation, imputation, and more [Електронний ресурс]. Режим доступу: https://github.com/aws/random-cut-forest-by-aws

49. Enhancing Anomaly Detection in Linux Audit Logs with AI | NVIDIA Technical Blog [Електронний ресурс]. Режим доступу: https://developer.nvidia.com/blog/enhancing-anomaly-detection-in-linux-audit-logs-with-ai

50. ETL Process & Tools [Електронний ресурс]. Режим доступу: https://www.sas.com/en_us/insights/data-management/what-is-etl.html

51. What is a Rootkit? How to defend and stop them? [Електронний ресурс]. Режим доступу: https://www.fortinet.com/resources/cyberglossary/rootkit

52. What happens in a TLS handshake? [Електронний ресурс]. Режим доступу: https://www.cloudflare.com/en-gb/learning/ssl/what-happens-in-a-tls-handshake/

53. What is Diffie-Hellman Key Exchange? [Електронний ресурс]. Режим доступу: https://www.techtarget.com/searchsecurity/definition/Diffie-Hellman-key-exchange

54. What is a Certificate Authority (CA)? [Електронний ресурс]. Режим доступу: https://www.ssl.com/article/what-is-a-certificate-authority-ca/

55. Encryption choices: RSA vs AES explained [Електронний ресурс]. Режим доступу: https://preyproject.com/blog/types-of-encryption-symmetric-or-asymmetric-rsa-or-aes

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки SIEM для протидії атакам у комп’ютерній мережі.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 57-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки SIEM для протидії атакам у комп’ютерній мережі.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– систему керування кібербезпеки SIEM для протидії атакам у комп’ютерній мережі;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 та Linux, і з сумісними з цими платформами пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11 або Linux, та веб-інтерфейсу для взаємодії між системою та користувачами.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище JavaScript/TypeScript, Python та Java.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма. Сценарій запуску та налаштування програмного комплексу.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 3 аркуша.
– Пояснювальна записка – 85 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 02.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Смірнов О.А.

Програмне забезпечення системи кібербезпеки SIEM для протидії атакам у комп’ютерній мережі

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 42

Літера: РП

Кропивницький – 2025 року

Файл setup.sh – сценарій встановлення розробленого ПЗ на ОС Linux

```
#!/bin/bash

# Сценарій для встановлення SIEM системи на ОС Linux x64
# Компоненти: Wazuh Manager, OpenSearch, Logstash, Kibana, OpenSearch
Dashboards, RCF Anomaly Detection

set -e

INSTALL_DIR="$PWD/apps"
CONFIG_DIR="$PWD/config"

# Версії
OPENSEARCH_VERSION="2.14.0"
LOGSTASH_VERSION="8.13.4"
KIBANA_VERSION="8.13.4"
WAZUH_VERSION="4.7.4"

# URLs
OPENSEARCH_URL="https://artifacts.opensearch.org/releases/bundle/opensearch/${OP
ENSEARCH_VERSION}/opensearch-${OPENSEARCH_VERSION}-linux-x64.tar.gz"
DASHBOARDS_URL="https://artifacts.opensearch.org/releases/bundle/opensearch-
dashboards/${OPENSEARCH_VERSION}/opensearch-dashboards-${OPENSEARCH_VERSION}-
linux-x64.tar.gz"
LOGSTASH_URL="https://artifacts.elastic.co/downloads/logstash/logstash-
${LOGSTASH_VERSION}-linux-x86_64.tar.gz"
KIBANA_URL="https://artifacts.elastic.co/downloads/kibana/kibana-
${KIBANA_VERSION}-linux-x86_64.tar.gz"
WAZUH_URL="https://packages.wazuh.com/4.x/wazuh-install.sh"

# Створення директорій
mkdir -p "$INSTALL_DIR" "$CONFIG_DIR"/wazuh "$CONFIG_DIR"/opensearch
"$CONFIG_DIR"/logstash/pipeline "$CONFIG_DIR"/kibana "$CONFIG_DIR"/dashboards
cd "$INSTALL_DIR"

# Завантаження та розпакування
wget -q "$OPENSEARCH_URL" -O opensearch.tar.gz && tar -xzf opensearch.tar.gz &&
rm opensearch.tar.gz
wget -q "$DASHBOARDS_URL" -O dashboards.tar.gz && tar -xzf dashboards.tar.gz &&
rm dashboards.tar.gz
wget -q "$LOGSTASH_URL" -O logstash.tar.gz && tar -xzf logstash.tar.gz && rm
logstash.tar.gz
wget -q "$KIBANA_URL" -O kibana.tar.gz && tar -xzf kibana.tar.gz && rm
kibana.tar.gz

# Встановлення Wazuh
wget -q "$WAZUH_URL" -O wazuh-install.sh
chmod +x wazuh-install.sh
sudo ./wazuh-install.sh -i manager

# Переміщення до директорій
mv opensearch-* opensearch
mv opensearch-dashboards-* dashboards
mv logstash-* logstash
mv kibana-* kibana

# Налаштування конфігурацій
cat > "$CONFIG_DIR/opensearch/opensearch.yml" <<EOF
cluster.name: siem-cluster
network.host: 0.0.0.0
discovery.type: single-node
plugins.security.disabled: true
EOF

cat > "$CONFIG_DIR/kibana/kibana.yml" <<EOF
server.name: kibana
```

```
server.host: "0.0.0.0"
elasticsearch.hosts: ["http://localhost:9200"]
EOF

cat > "$CONFIG_DIR/logstash/pipeline/logstash.conf" <<EOF
input {
  beats {
    port => 5044
  }
}

output {
  opensearch {
    hosts => ["http://localhost:9200"]
    index => "wazuh-alerts-%{+YYYY.MM.dd}"
  }
}
EOF

# Конфігурації детекторів аномалій
cat > "$CONFIG_DIR/dashboards/rcf-detector.json" <<'JSON'
{
  "name": "wazuh-rcf-detector",
  "description": "Detect anomalies using RCF on Wazuh alerts",
  "time_field": "@timestamp",
  "indices": ["wazuh-alerts-*"],
  "feature_attributes": [
    {
      "feature_name": "count_alerts",
      "feature_enabled": true,
      "aggregation_query": {
        "count_alerts": {
          "value_count": {
            "field": "rule.level"
          }
        }
      }
    }
  ],
  "detection_interval": {
    "period": {
      "interval": 1,
      "unit": "minutes"
    }
  },
  "window_delay": {
    "period": {
      "interval": 1,
      "unit": "minutes"
    }
  },
  "shingle_size": 8,
  "category_field": [],
  "schema_version": 0
}
JSON

# Налаштування сервісів systemd для автозапуску та фонової роботи
sudo bash -c 'cat > /etc/systemd/system/opensearch.service' <<EOF
[Unit]
Description=OpenSearch
After=network.target

[Service]
Type=simple
ExecStart=$INSTALL_DIR/opensearch/bin/opensearch
WorkingDirectory=$INSTALL_DIR/opensearch
Restart=always
LimitNOFILE=65535
```

```
[Install]
WantedBy=multi-user.target
EOF

sudo bash -c 'cat > /etc/systemd/system/dashboards.service' <<EOF
[Unit]
Description=OpenSearch Dashboards
After=network.target

[Service]
Type=simple
ExecStart=$INSTALL_DIR/dashboards/bin/opensearch-dashboards
WorkingDirectory=$INSTALL_DIR/dashboards
Restart=always

[Install]
WantedBy=multi-user.target
EOF

sudo bash -c 'cat > /etc/systemd/system/logstash.service' <<EOF
[Unit]
Description=Logstash
After=network.target

[Service]
Type=simple
ExecStart=$INSTALL_DIR/logstash/bin/logstash -f
$CONFIG_DIR/logstash/pipeline/logstash.conf
WorkingDirectory=$INSTALL_DIR/logstash
Restart=always

[Install]
WantedBy=multi-user.target
EOF

sudo bash -c 'cat > /etc/systemd/system/kibana.service' <<EOF
[Unit]
Description=Kibana
After=network.target

[Service]
Type=simple
ExecStart=$INSTALL_DIR/kibana/bin/kibana
WorkingDirectory=$INSTALL_DIR/kibana
Restart=always

[Install]
WantedBy=multi-user.target
EOF

# Перезавантаження та підключення сервісів
sudo systemctl daemon-reload
sudo systemctl enable opensearch dashboards logstash kibana

echo "Встановлення завершено. Використовуйте 'sudo systemctl start <service>'
для запуску компонентів."
echo "Щоб додати детектори аномалій, запустіть:"
echo "curl -XPOST
\"http://localhost:9200/_plugins/_anomaly_detection/detectors\" -H 'Content-
Type: application/json' -d @$CONFIG_DIR/dashboards/rcf-detector.json"
```

Файл setup.ps1 – модуль сповіщення про довідкову інформацію

```
# Встановлення та налаштування SIEM системи на ОС Windows
# Компоненти: Wazuh Manager, OpenSearch, Logstash, Kibana, OpenSearch
Dashboards, RCF Anomaly Detection

$InstallDir = "$PSScriptRoot\apps"
$ConfigDir = "$PSScriptRoot\config"

# Версії
$OpenSearchVersion = "2.14.0"
$LogstashVersion = "8.13.4"
$KibanaVersion = "8.13.4"
$WazuhVersion = "4.7.4"

# URLs
$OpenSearchUrl =
"https://artifacts.opensearch.org/releases/bundle/opensearch/$OpenSearchVersion/
opensearch-$OpenSearchVersion-windows-x64.zip"
$DashboardsUrl = "https://artifacts.opensearch.org/releases/bundle/opensearch-
dashboards/$OpenSearchVersion/opensearch-dashboards-$OpenSearchVersion-windows-
x64.zip"
$LogstashUrl = "https://artifacts.elastic.co/downloads/logstash/logstash-
$LogstashVersion-windows-x86_64.zip"
$KibanaUrl = "https://artifacts.elastic.co/downloads/kibana/kibana-
$KibanaVersion-windows-x86_64.zip"
$WazuhUrl = "https://packages.wazuh.com/4.x/windows/wazuh-agent-
$WazuhVersion.msi"

# Створення директорій для конфігурацій
New-Item -ItemType Directory -Force -Path $InstallDir, "$ConfigDir\wazuh",
"$ConfigDir\opensearch", "$ConfigDir\logstash\pipeline", "$ConfigDir\kibana",
"$ConfigDir\dashboards"

# Завантаження функції
Function Download-And-Unzip ($url, $dest) {
    $zip = "$env:TEMP\temp.zip"
    Invoke-WebRequest -Uri $url -OutFile $zip
    Expand-Archive -Path $zip -DestinationPath $dest -Force
    Remove-Item $zip
}

# Завантаження та розпакування компонентів
Download-And-Unzip $OpenSearchUrl "$InstallDir\opensearch"
Download-And-Unzip $DashboardsUrl "$InstallDir\dashboards"
Download-And-Unzip $LogstashUrl "$InstallDir\logstash"
Download-And-Unzip $KibanaUrl "$InstallDir\kibana"

# Створення конфігураційних файлів
@"
cluster.name: siem-cluster
network.host: 0.0.0.0
discovery.type: single-node
plugins.security.disabled: true
"@ | Set-Content "$ConfigDir\opensearch\opensearch.yml"

@"
server.name: kibana
server.host: ""0.0.0.0""
elasticsearch.hosts: [""http://localhost:9200""]
"@ | Set-Content "$ConfigDir\kibana\kibana.yml"

@"
input {
  beats {
    port => 5044
  }
}
```

```
output {
  opensearch {
    hosts => [""http://localhost:9200""]
    index => ""wazuh-alerts-%{+YYYY.MM.dd}""
  }
}
"@ | Set-Content "$ConfigDir\logstash\pipeline\logstash.conf"

@'
{
  "name": "wazuh-rcf-detector",
  "description": "Detect anomalies using RCF on Wazuh alerts",
  "time_field": "@timestamp",
  "indices": ["wazuh-alerts-*"],
  "feature_attributes": [
    {
      "feature_name": "count_alerts",
      "feature_enabled": true,
      "aggregation_query": {
        "count_alerts": {
          "value_count": {
            "field": "rule.level"
          }
        }
      }
    }
  ],
  "detection_interval": {
    "period": {
      "interval": 1,
      "unit": "minutes"
    }
  },
  "window_delay": {
    "period": {
      "interval": 1,
      "unit": "minutes"
    }
  },
  "shingle_size": 8,
  "category_field": [],
  "schema_version": 0
}
'@ | Set-Content "$ConfigDir\dashboards\rcf-detector.json"

Write-Host "Встановлення завершено."
Write-Host "Для запуска компонентів, перейдість до відповідної директорії та
запустіть:"
Write-Host "Запустити OpenSearch: .\bin\opensearch.bat"
Write-Host "Запустити Dashboards: .\bin\opensearch-dashboards.bat"
Write-Host "Запустити Logstash: .\bin\logstash.bat -f
`"$ConfigDir\logstash\pipeline\logstash.conf`""
Write-Host "Запустити Kibana: .\bin\kibana.bat"
Write-Host ""
Write-Host "Для додавання детекторів аномалій виконайте:"
Write-Host "curl -XPOST
`"http://localhost:9200/_plugins/_anomaly_detection/detectors`" -H 'Content-
Type: application/json' -d @$ConfigDir\dashboards\rcf-detector.json"
```

Файл about.js – модуль сповіщення про довідкову інформацію

```
<script>
function about() {
  alert("МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ\n\
Центральноукраїнський національний технічний університет\n\
Кафедра кібербезпеки та програмного забезпечення\n\
ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА\n\
за першим (бакалаврським) рівнем вищої освіти\n\
на тему \"Програмне забезпечення системи кібербезпеки SIEM для протидії атакам у
комп’ютерній мережі\"\n\
ОПП \"Кібербезпека\"\n\
спеціальності 125 \"Кібербезпека\"\n\
Виконав: Козлов Я.О.\n\
Науковий керівник: Смірнов О.А.\n\
Кропивницький - 2025");
}

about();
</script>
```

Файл docker-compose.yml – конфігураційний файл для кросплатформеної системи Docker

```
version: '3.8'

services:
  opensearch:
    image: opensearchproject/opensearch:2.14.0
    container_name: opensearch
    environment:
      - discovery.type=single-node
      - bootstrap.memory_lock=true
      - plugins.security.disabled=true
      - cluster.name=siem-cluster
      - OPENSEARCH_JAVA_OPTS=-Xms1g -Xmx1g
    ulimits:
      memlock:
        soft: -1
        hard: -1
    configs:
      - source: opensearch_config
        target: /usr/share/opensearch/config/opensearch.yml
    ports:
      - 9200:9200
      - 9600:9600
    networks:
      - siem-net

  dashboards:
    image: opensearchproject/opensearch-dashboards:2.14.0
    container_name: dashboards
    depends_on:
      - opensearch
    environment:
      - OPENSEARCH_HOSTS=http://opensearch:9200
    configs:
      - source: dashboards_config
        target: /usr/share/opensearch-
dashboards/config/opensearch_dashboards.yml
    ports:
      - 5601:5601
    networks:
      - siem-net

  logstash:
    image: docker.elastic.co/logstash/logstash:8.13.4
    container_name: logstash
    depends_on:
      - opensearch
    configs:
      - source: logstash_pipeline
        target: /usr/share/logstash/pipeline/logstash.conf
    ports:
      - 5044:5044
    networks:
      - siem-net

  wazuh.manager:
    image: wazuh/wazuh-manager:4.7.4
    container_name: wazuh-manager
    ports:
      - "1514:1514/udp"
      - "1515:1515"
      - "55000:55000"
    networks:
      - siem-net

networks:
  siem-net:
```

```
    driver: bridge

configs:
  opensearch_config:
    name: opensearch_config
    file: /dev/null
    content: |
      cluster.name: siem-cluster
      network.host: 0.0.0.0
      discovery.type: single-node
      plugins.security.disabled: true

  dashboards_config:
    name: dashboards_config
    file: /dev/null
    content: |
      server.name: dashboards
      server.host: "0.0.0.0"
      opensearch.hosts: ["http://opensearch:9200"]

  logstash_pipeline:
    name: logstash_pipeline
    file: /dev/null
    content: |
      input {
        beats {
          port => 5044
        }
      }

      output {
        opensearch {
          hosts => ["http://opensearch:9200"]
          index => "wazuh-alerts-%{+YYYY.MM.dd}"
        }
      }
```

Файл detection_curl.sh – налаштування детекторів після запуску системи за допомогою Docker

```
#!/bin/bash

curl -XPOST "http://localhost:9200/_plugins/_anomaly_detection/detectors" \
     -H 'Content-Type: application/json' \
     -d '{
       "name": "wazuh-rcf-detector",
       "description": "Detect anomalies using RCF on Wazuh alerts",
       "time_field": "@timestamp",
       "indices": ["wazuh-alerts-*"],
       "feature_attributes": [
         {
           "feature_name": "count_alerts",
           "feature_enabled": true,
           "aggregation_query": {
             "count_alerts": {
               "value_count": {
                 "field": "rule.level"
               }
             }
           }
         }
       ],
       "detection_interval": {
         "period": {
           "interval": 1,
           "unit": "minutes"
         }
       },
       "window_delay": {
         "period": {
           "interval": 1,
           "unit": "minutes"
         }
       },
       "shingle_size": 8,
       "category_field": [],
       "schema_version": 0
     }'
```

Файл RandomCutForest.java – програмна реалізація алгоритмів виявлення аномалій

```
/*
 * Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

package com.amazon.randomcutforest;

import static com.amazon.randomcutforest.CommonUtils.checkArgument;
import static com.amazon.randomcutforest.CommonUtils.checkNotNull;
import static com.amazon.randomcutforest.CommonUtils.toDoubleArray;
import static com.amazon.randomcutforest.CommonUtils.toFloatArray;
import static
com.amazon.randomcutforest.summarization.Summarizer.DEFAULT_SEPARATION_RATIO_FOR
_MERGE;
import static java.lang.Math.max;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Optional;
import java.util.Random;
import java.util.function.BiFunction;
import java.util.function.BinaryOperator;
import java.util.function.Function;
import java.util.stream.Collector;

import com.amazon.randomcutforest.anomalydetection.AnomalyAttributionVisitor;
import com.amazon.randomcutforest.anomalydetection.AnomalyScoreVisitor;
import com.amazon.randomcutforest.anomalydetection.DynamicAttributionVisitor;
import com.amazon.randomcutforest.anomalydetection.DynamicScoreVisitor;
import
com.amazon.randomcutforest.anomalydetection.SimulatedTransductiveScalarScoreVisi
tor;
import com.amazon.randomcutforest.config.Config;
import com.amazon.randomcutforest.config.Precision;
import com.amazon.randomcutforest.executor.AbstractForestTraversalExecutor;
import com.amazon.randomcutforest.executor.AbstractForestUpdateExecutor;
import com.amazon.randomcutforest.executor.IStateCoordinator;
import com.amazon.randomcutforest.executor.ParallelForestTraversalExecutor;
import com.amazon.randomcutforest.executor.ParallelForestUpdateExecutor;
import com.amazon.randomcutforest.executor.PointStoreCoordinator;
import com.amazon.randomcutforest.executor.SamplerPlusTree;
import com.amazon.randomcutforest.executor.SequentialForestTraversalExecutor;
import com.amazon.randomcutforest.executor.SequentialForestUpdateExecutor;
import com.amazon.randomcutforest.imputation.ConditionalSampleSummarizer;
import com.amazon.randomcutforest.imputation.ImputeVisitor;
import com.amazon.randomcutforest.inspect.NearNeighborVisitor;
import com.amazon.randomcutforest.interpolation.SimpleInterpolationVisitor;
import com.amazon.randomcutforest.returntypes.ConditionalTreeSample;
import com.amazon.randomcutforest.returntypes.ConvergingAccumulator;
import com.amazon.randomcutforest.returntypes.DensityOutput;
import com.amazon.randomcutforest.returntypes.DiVector;
import com.amazon.randomcutforest.returntypes.InterpolationMeasure;
import com.amazon.randomcutforest.returntypes.Neighbor;
import
com.amazon.randomcutforest.returntypes.OneSidedConvergingDiVectorAccumulator;
```

```
import
com.amazon.randomcutforest.returntypes.OneSidedConvergingDoubleAccumulator;
import com.amazon.randomcutforest.returntypes.RangeVector;
import com.amazon.randomcutforest.returntypes.SampleSummary;
import com.amazon.randomcutforest.sampler.CompactSampler;
import com.amazon.randomcutforest.sampler.IStreamSampler;
import com.amazon.randomcutforest.store.IPointStore;
import com.amazon.randomcutforest.store.PointStore;
import com.amazon.randomcutforest.summarization.ICluster;
import com.amazon.randomcutforest.summarization.Summarizer;
import com.amazon.randomcutforest.tree.IBoundingBoxView;
import com.amazon.randomcutforest.tree.ITree;
import com.amazon.randomcutforest.tree.RandomCutTree;
import com.amazon.randomcutforest.util.ShingleBuilder;

/**
 * The RandomCutForest class is the interface to the algorithms in this package,
 * and includes methods for anomaly detection, anomaly detection with
 * attribution, density estimation, imputation, and forecasting. A Random Cut
 * Forest is a collection of Random Cut Trees and stream samplers. When an
 * update call is made to a Random Cut Forest, each sampler is independently
 * updated with the submitted (and if the point is accepted by the sampler, then
 * the corresponding Random Cut Tree is also updated. Similarly, when an
 * algorithm method is called, the Random Cut Forest proxies to the trees which
 * implement the actual scoring logic. The Random Cut Forest then combines
 * partial results into a final results.
 */
public class RandomCutForest {

    /**
     * Default sample size. This is the number of points retained by the stream
     * sampler.
     */
    public static final int DEFAULT_SAMPLE_SIZE = 256;

    /**
     * Default fraction used to compute the amount of points required by stream
     * samplers before results are returned.
     */
    public static final double DEFAULT_OUTPUT_AFTER_FRACTION = 0.25;

    /**
     * If the user doesn't specify an explicit time decay value, then we set it
to
     * the inverse of this coefficient times sample size.
     */
    public static final double DEFAULT_SAMPLE_SIZE_COEFFICIENT_IN_TIME_DECAY =
10.0;

    /**
     * Default number of trees to use in the forest.
     */
    public static final int DEFAULT_NUMBER_OF_TREES = 50;

    /**
     * By default, trees will not store sequence indexes.
     */
    public static final boolean DEFAULT_STORE_SEQUENCE_INDEXES_ENABLED = false;

    /**
     * By default, trees will accept every point until full.
     */
    public static final double DEFAULT_INITIAL_ACCEPT_FRACTION = 1.0;

    /**
     * By default, the collection of points stored in the forest will increase
from
     * a small size, as needed to maximum capacity
     */
```

```
    public static final boolean DEFAULT_DYNAMIC_RESIZING_ENABLED = true;

    /**
     * By default, shingling will be external
     */
    public static final boolean DEFAULT_INTERNAL_SHINGLING_ENABLED = false;

    /**
     * By default, shingles will be a sliding window and not a cyclic buffer
     */
    public static final boolean DEFAULT_INTERNAL_ROTATION_ENABLED = false;

    /**
     * By default, point stores will favor speed of size for larger shingle
sizes
     */
    public static final boolean DEFAULT_DIRECT_LOCATION_MAP = false;

    /**
     * Default floating-point precision for internal data structures.
     */
    public static final Precision DEFAULT_PRECISION = Precision.FLOAT_32;

    /**
     * fraction of bounding boxes maintained by each tree
     */
    public static final double DEFAULT_BOUNDING_BOX_CACHE_FRACTION = 1.0;

    /**
     * By default, nodes will not store center of mass.
     */
    public static final boolean DEFAULT_CENTER_OF_MASS_ENABLED = false;

    /**
     * By default RCF is unaware of shingle size
     */
    public static final int DEFAULT_SHINGLE_SIZE = 1;

    /**
     * Parallel execution is enabled by default.
     */
    public static final boolean DEFAULT_PARALLEL_EXECUTION_ENABLED = false;

    public static final boolean
DEFAULT_APPROXIMATE_ANOMALY_SCORE_HIGH_IS_CRITICAL = true;

    public static final double DEFAULT_APPROXIMATE_DYNAMIC_SCORE_PRECISION =
0.1;

    public static final int
DEFAULT_APPROXIMATE_DYNAMIC_SCORE_MIN_VALUES_ACCEPTED = 5;

    /**
     * Random number generator used by the forest.
     */
    protected Random random;
    /**
     * The number of dimensions in the input data.
     */
    protected final int dimensions;
    /**
     * The sample size used by stream samplers in this forest.
     */
    protected final int sampleSize;
    /**
     * The shingle size (if known)
     */
    protected final int shingleSize;
    /**
```

```
     * The input dimensions for known shingle size and internal shingling
     */
    protected final int inputDimensions;
    /**
     * The number of points required by stream samplers before results are
returned.
     */
    protected final int outputAfter;
    /**
     * The number of trees in this forest.
     */
    protected final int numberOfTrees;
    /**
     * The decay factor used by stream samplers in this forest.
     */
    protected double timeDecay;
    /**
     * Store the time information
     */
    protected final boolean storeSequenceIndexesEnabled;

    /**
     * enables internal shingling
     */
    protected final boolean internalShinglingEnabled;

    /**
     * The following can be set between 0 and 1 (inclusive) to achieve tradeoff
     * between smaller space, lower throughput and larger space, larger
throughput
     */
    protected final double boundingBoxCacheFraction;
    /**
     * Enable center of mass at internal nodes
     */
    protected final boolean centerOfMassEnabled;
    /**
     * Enable parallel execution.
     */
    protected final boolean parallelExecutionEnabled;
    /**
     * Number of threads to use in the thread pool if parallel execution is
enabled.
     */
    protected final int threadPoolSize;
    /**
     * A string to define an "execution mode" that can be used to set multiple
     * configuration options. This field is not currently in use.
     */
    protected String executionMode;

    protected IStateCoordinator<?, float[]> stateCoordinator;
    protected ComponentList<?, float[]> components;

    /**
     * This flag is initialized to false. It is set to true when all component
     * models are ready.
     */
    private boolean outputReady;

    /**
     * used for initializing the compact forests
     */
    private final int initialPointStoreSize;
    private final int pointStoreCapacity;

    /**
     * An implementation of forest traversal algorithms.
     */
```

```
    protected AbstractForestTraversalExecutor traversalExecutor;

    /**
     * An implementation of forest update algorithms.
     */
    protected AbstractForestUpdateExecutor<?, float[]> updateExecutor;

    public <P> RandomCutForest(Builder<?> builder, IStateCoordinator<P, float[]>
stateCoordinator,
            ComponentList<P, float[]> components, Random random) {
        this(builder, false);

        checkNotNull(stateCoordinator, "updateCoordinator must not be null");
        checkNotNull(components, "componentModels must not be null");
        checkNotNull(random, "random must not be null");

        this.stateCoordinator = stateCoordinator;
        this.components = components;
        this.random = random;
        initExecutors(stateCoordinator, components);
    }

    public RandomCutForest(Builder<?> builder) {
        this(builder, false);
        random = builder.getRandom();

        PointStore tempStore =
PointStore.builder().internalRotationEnabled(builder.internalRotationEnabled)
                .capacity(pointStoreCapacity).initialSize(initialPointStoreSize)

.internalShinglingEnabled(internalShinglingEnabled).shingleSize(shingleSize).dim
ensions(dimensions)
                .build();

        IStateCoordinator<Integer, float[]> stateCoordinator = new
PointStoreCoordinator<>(tempStore);
        ComponentList<Integer, float[]> components = new
ComponentList<>(numberOfTrees);
        for (int i = 0; i < numberOfTrees; i++) {
            ITree<Integer, float[]> tree = new
RandomCutTree.Builder().capacity(sampleSize)
                    .randomSeed(random.nextLong()).pointStoreView(tempStore)

.boundingBoxCacheFraction(boundingBoxCacheFraction).centerOfMassEnabled(centerOf
MassEnabled)

.storeSequenceIndexesEnabled(storeSequenceIndexesEnabled).outputAfter(1).build()
;

            IStreamSampler<Integer> sampler =
CompactSampler.builder().capacity(sampleSize).timeDecay(timeDecay)

.randomSeed(random.nextLong()).storeSequenceIndexesEnabled(storeSequenceIndexesE
nabled)

.initialAcceptFraction(builder.initialAcceptFraction).build();

            components.add(new SamplerPlusTree<>(sampler, tree));
        }
        this.stateCoordinator = stateCoordinator;
        this.components = components;
        initExecutors(stateCoordinator, components);
    }

    protected <PointReference> void
initExecutors(IStateCoordinator<PointReference, float[]> updateCoordinator,
            ComponentList<PointReference, float[]> components) {
        if (parallelExecutionEnabled) {
```

```
            traversalExecutor = new ParallelForestTraversalExecutor(components,
threadPoolSize);
            updateExecutor = new
ParallelForestUpdateExecutor<>(updateCoordinator, components, threadPoolSize);
        } else {
            traversalExecutor = new
SequentialForestTraversalExecutor(components);
            updateExecutor = new
SequentialForestUpdateExecutor<>(updateCoordinator, components);
        }
    }

    /**
     * This constructor is responsible for initializing a forest's configuration
     * variables from a builder. The method signature contains a boolean
argument
     * that isn't used. This argument exists only to create a distinct method
     * signature so that we can expose {@link #RandomCutForest(Builder)} as a
     * protected constructor.
     *
     * @param builder A Builder instance giving the desired random cut forest
     *                configuration.
     * @param notUsed This parameter is not used.
     */
    protected RandomCutForest(Builder<?> builder, boolean notUsed) {
        checkArgument(builder.numberOfTrees > 0, "numberOfTrees must be greater
than 0");
        checkArgument(builder.sampleSize > 0, "sampleSize must be greater than
0");
        builder.outputAfter.ifPresent(n -> checkArgument(n > 0, "outputAfter
must be greater than 0"));
        checkArgument(builder.dimensions > 0, "dimensions must be greater than
0");
        builder.timeDecay
                .ifPresent(timeDecay -> checkArgument(timeDecay >= 0, "timeDecay
must be greater than or equal to 0"));
        builder.threadPoolSize.ifPresent(n -> {
            checkArgument(n >= 0, "cannot be negative");
            checkArgument((n > 0) || (!builder.parallelExecutionEnabled),
                    "threadPoolSize must be greater/equal than 0. To disable
thread pool, set parallel execution to 'false'.");
        });
        checkArgument(builder.shingleSize == 1 || builder.dimensions %
builder.shingleSize == 0, "wrong shingle size");
        if (builder.internalRotationEnabled) {
            checkArgument(builder.internalShinglingEnabled, " enable internal
shingling");
        }
        builder.initialPointStoreSize
                .ifPresent(n -> checkArgument(n > 0, "initial point store must
be greater than 0"));
        checkArgument(builder.boundingBoxCacheFraction >= 0, "cache cannot be
negative");
        checkArgument(builder.boundingBoxCacheFraction <= 1, "incorrect cache
fraction range");
        numberOfTrees = builder.numberOfTrees;
        sampleSize = builder.sampleSize;
        outputAfter = builder.outputAfter.orElse(max(1, (int) (sampleSize *
DEFAULT_OUTPUT_AFTER_FRACTION)));
        internalShinglingEnabled = builder.internalShinglingEnabled;
        shingleSize = builder.shingleSize;
        dimensions = builder.dimensions;
        timeDecay = builder.timeDecay.orElse(1.0 /
(DEFAULT_SAMPLE_SIZE_COEFFICIENT_IN_TIME_DECAY * sampleSize));
        storeSequenceIndexesEnabled = builder.storeSequenceIndexesEnabled;
        centerOfMassEnabled = builder.centerOfMassEnabled;
        parallelExecutionEnabled = builder.parallelExecutionEnabled;
        boundingBoxCacheFraction = builder.boundingBoxCacheFraction;
```

```
        builder.directLocationMapEnabled = builder.directLocationMapEnabled ||
shingleSize == 1;
        inputDimensions = (internalShinglingEnabled) ? dimensions / shingleSize
: dimensions;
        pointStoreCapacity = max(sampleSize * numberOfTrees + 1, 2 *
sampleSize);
        initialPointStoreSize = builder.initialPointStoreSize.orElse(2 *
sampleSize);

        if (parallelExecutionEnabled) {
            threadPoolSize =
builder.threadPoolSize.orElse(Runtime.getRuntime().availableProcessors() - 1);
        } else {
            threadPoolSize = 0;
        }
    }

    /**
     * @return a new RandomCutForest builder.
     */
    public static Builder builder() {
        return new Builder();
    }

    /**
     * Create a new RandomCutForest with optional arguments set to default
values.
     *
     * @param dimensions The number of dimension in the input data.
     * @param randomSeed The random seed to use to create the forest random
number
     *                   generator
     * @return a new RandomCutForest with optional arguments set to default
values.
     */
    public static RandomCutForest defaultForest(int dimensions, long randomSeed)
{
        return builder().dimensions(dimensions).randomSeed(randomSeed).build();
    }

    /**
     * Create a new RandomCutForest with optional arguments set to default
values.
     *
     * @param dimensions The number of dimension in the input data.
     * @return a new RandomCutForest with optional arguments set to default
values.
     */
    public static RandomCutForest defaultForest(int dimensions) {
        return builder().dimensions(dimensions).build();
    }

    /**
     * @return the number of trees in the forest.
     */
    public int getNumberOfTrees() {
        return numberOfTrees;
    }

    /**
     * @return the sample size used by stream samplers in this forest.
     */
    public int getSampleSize() {
        return sampleSize;
    }

    /**
     * @return the shingle size used by the point store.
     */
```

```
    public int getShingleSize() {
        return shingleSize;
    }

    /**
     * @return the number of points required by stream samplers before results
are
     *         returned.
     */
    public int getOutputAfter() {
        return outputAfter;
    }

    /**
     * @return the number of dimensions in the data points accepted by this
forest.
     */
    public int getDimensions() {
        return dimensions;
    }

    /**
     * @return return the decay factor used by stream samplers in this forest.
     */
    public double getTimeDecay() {
        return timeDecay;
    }

    /**
     * @return true if points are saved with sequence indexes, false otherwise.
     */
    public boolean isStoreSequenceIndexesEnabled() {
        return storeSequenceIndexesEnabled;
    }

    /**
     * For compact forests, users can choose to specify the desired floating-
point
     * precision to use internally to store points. Choosing single-precision
will
     * reduce the memory size of the model at the cost of requiring double/float
     * conversions.
     *
     * @return the desired precision to use internally to store points.
     */
    public Precision getPrecision() {
        return Precision.FLOAT_32;
    }

    @Deprecated
    public boolean isCompact() {
        return true;
    }

    /**
     * @return true if internal shingling is performed, false otherwise.
     */
    public boolean isInternalShinglingEnabled() {
        return internalShinglingEnabled;
    }

    /**
     * @return true if tree nodes retain the center of mass, false otherwise.
     */
    public boolean isCenterOfMassEnabled() {
        return centerOfMassEnabled;
    }

    /**
```

```
     * @return true if parallel execution is enabled, false otherwise.
     */
    public boolean isParallelExecutionEnabled() {
        return parallelExecutionEnabled;
    }

    public double getBoundingBoxCacheFraction() {
        return boundingBoxCacheFraction;
    }

    /**
     * @return the number of threads in the thread pool if parallel execution is
     *         enabled, 0 otherwise.
     */
    public int getThreadPoolSize() {
        return threadPoolSize;
    }

    public IStateCoordinator<?, ?> getUpdateCoordinator() {
        return stateCoordinator;
    }

    public ComponentList<?, ?> getComponents() {
        return components;
    }

    /**
     * used for scoring and other function, expands to a shingled point in
either
     * case performs a clean copy
     *
     * @param point input point
     * @return a shingled copy or a clean copy
     */

    public float[] transformToShingledPoint(float[] point) {
        return stateCoordinator.getStore().transformToShingledPoint(point);
    }

    /**
     * does the pointstore use rotated shingles
     *
     * @return true/false based on pointstore
     */
    public boolean isRotationEnabled() {
        return stateCoordinator.getStore().isInternalRotationEnabled();
    }

    /**
     * transforms the missing indices on the input point to the corresponding
     * indices of a shingled point
     *
     * @param indexList input array of missing values
     * @param length    length of the input array
     * @return output array of missing values corresponding to shingle
     */
    protected int[] transformIndices(int[] indexList, int length) {
        return (internalShinglingEnabled && length == inputDimensions)
                ? stateCoordinator.getStore().transformIndices(indexList)
                : indexList;
    }

    /**
     *
     * @return the last known shingled point seen
     */
    public float[] lastShingledPoint() {
        checkArgument(internalShinglingEnabled, "incorrect use");
        return stateCoordinator.getStore().getInternalShingle();
```

```
    }

    /**
     *
     * @return the sequence index of the last known shingled point. If internal
     *         shingling is not enabled, then this would correspond to the
number of
     *         updates
     */
    public long nextSequenceIndex() {
        return stateCoordinator.getStore().getNextSequenceIndex();
    }

    /**
     * Update the forest with the given point. The point is submitted to each
     * sampler in the forest. If the sampler accepts the point, the point is
     * submitted to the update method in the corresponding Random Cut Tree.
     *
     * @param point             The point used to update the forest.
     * @param updateShingleOnly only update the shingle (true for internal
     *                          shingling)
     */

    public void update(float[] point, boolean updateShingleOnly) {
        checkNotNull(point, "point must not be null");
        checkArgument(internalShinglingEnabled || point.length == dimensions,
                String.format("point.length must equal %d", dimensions));
        checkArgument(!internalShinglingEnabled || point.length ==
inputDimensions,
                String.format("point.length must equal %d for internal
shingling", inputDimensions));
        checkArgument(!updateShingleOnly || internalShinglingEnabled,
                "update shingle setting is only valid for internal shingling");

        updateExecutor.update(point, updateShingleOnly);
    }

    @Deprecated
    public void update(double[] point) {
        update(toFloatArray(point), false);
    }

    public void update(float[] point) {
        update(point, false);
    }

    /**
     * Update the forest with the given point and a timestamp. The point is
     * submitted to each sampler in the forest as if that timestamp was the
correct
     * stamp. storeSequenceIndexes must be false since the algorithm will not
verify
     * the correctness of the timestamp.
     *
     * @param point       The point used to update the forest.
     * @param sequenceNum The timestamp of the corresponding point
     */
    public void update(double[] point, long sequenceNum) {
        checkNotNull(point, "point must not be null");
        update(toFloatArray(point), sequenceNum);
    }

    public void update(float[] point, long sequenceNum) {
        checkNotNull(point, "point must not be null");
        checkArgument(!internalShinglingEnabled, "cannot be applied with
internal shingling");
        checkArgument(point.length == dimensions, () -> "point.length must equal
to " + dimensions);
        updateExecutor.update(point, sequenceNum);
```

```
    }

    /**
     * Update the forest such that each tree caches a fraction of the bounding
     * boxes. This allows for a tradeoff between speed and storage.
     *
     * @param cacheFraction The (approximate) fraction of bounding boxes used in
     *                      caching.
     */
    public void setBoundingBoxCacheFraction(double cacheFraction) {
        checkArgument(0 <= cacheFraction, "cache cannot be negative");
        checkArgument(cacheFraction <= 1, "cacheFraction must be between 0 and 1
(inclusive)");
        updateExecutor.getComponents().forEach(c ->
c.setConfig(Config.BOUNDING_BOX_CACHE_FRACTION, cacheFraction));
    }

    /**
     * changes the setting of time dependent sampling on the fly
     *
     * @param timeDecay new value of sampling rate
     */
    public void setTimeDecay(double timeDecay) {
        checkArgument(0 <= timeDecay, "timeDecay must be greater than or equal
to 0");
        this.timeDecay = timeDecay;
        updateExecutor.getComponents().forEach(c ->
c.setConfig(Config.TIME_DECAY, timeDecay));
    }

    /**
     * Visit each of the trees in the forest and combine the individual results
into
     * an aggregate result. A visitor is constructed for each tree using the
visitor
     * factory, and then submitted to
     * {@link RandomCutTree#traverse(float[], IVisitorFactory)}. The results
from
     * all the trees are combined using the accumulator and then transformed
using
     * the finisher before being returned. Trees are visited in parallel using
     * {@link java.util.Collection#parallelStream()}.
     *
     * @param point         The point that defines the traversal path.
     * @param visitorFactory A factory method which is invoked for each tree to
     *                      construct a visitor.
     * @param accumulator    A function that combines the results from
individual
     *                      trees into an aggregate result.
     * @param finisher       A function called on the aggregate result in order
to
     *                      produce the final result.
     * @param <R>           The visitor result type. This is the type that will
be
     *                      returned after traversing each individual tree.
     * @param <S>           The final type, after any final normalization at
the
     *                      forest level.
     * @return The aggregated and finalized result after sending a visitor
through
     *         each tree in the forest.
     */
    public <R, S> S traverseForest(float[] point, IVisitorFactory<R>
visitorFactory, BinaryOperator<R> accumulator,
            Function<R, S> finisher) {

        checkNotNull(point, "point must not be null");
        checkArgument(point.length == dimensions, () -> "point.length must equal
to " + dimensions);
```

```
        checkNotNull(visitorFactory, "visitorFactory must not be null");
        checkNotNull(accumulator, "accumulator must not be null");
        checkNotNull(finisher, "finisher must not be null");

        return traversalExecutor.traverseForest(point, visitorFactory,
accumulator, finisher);
    }

    /**
     * Visit each of the trees in the forest and combine the individual results
into
     * an aggregate result. A visitor is constructed for each tree using the
visitor
     * factory, and then submitted to
     * {@link RandomCutTree#traverse(float[], IVisitorFactory)}. The results
from
     * individual trees are collected using the {@link
java.util.stream.Collector}
     * and returned. Trees are visited in parallel using
     * {@link java.util.Collection#parallelStream()}.
     *
     * @param point          The point that defines the traversal path.
     * @param visitorFactory A factory method which is invoked for each tree to
     *                       construct a visitor.
     * @param collector      A collector used to aggregate individual tree
results
     *                       into a final result.
     * @param <R>            The visitor result type. This is the type that will
be
     *                       returned after traversing each individual tree.
     * @param <S>            The final type, after any final normalization at
the
     *                       forest level.
     * @return The aggregated and finalized result after sending a visitor
through
     *         each tree in the forest.
     */
    public <R, S> S traverseForest(float[] point, IVisitorFactory<R>
visitorFactory, Collector<R, ?, S> collector) {

        checkNotNull(point, "point must not be null");
        checkArgument(point.length == dimensions, () -> "point.length must equal
to " + dimensions);
        checkNotNull(visitorFactory, "visitorFactory must not be null");
        checkNotNull(collector, "collector must not be null");

        return traversalExecutor.traverseForest(point, visitorFactory,
collector);
    }

    /**
     * Visit each of the trees in the forest sequentially and combine the
individual
     * results into an aggregate result. A visitor is constructed for each tree
     * using the visitor factory, and then submitted to
     * {@link RandomCutTree#traverse(float[], IVisitorFactory)}. The results
from
     * all the trees are combined using the {@link ConvergingAccumulator}, and
the
     * method stops visiting trees after convergence is reached. The result is
     * transformed using the finisher before being returned.
     *
     * @param point          The point that defines the traversal path.
     * @param visitorFactory A factory method which is invoked for each tree to
     *                       construct a visitor.
     * @param accumulator    An accumulator that combines the results from
     *                       individual trees into an aggregate result and
checks to
     *                       see if the result can be returned without further
```

```
     *                          processing.
     * @param finisher          A function called on the aggregate result in order
to
     *                          produce the final result.
     * @param <R>               The visitor result type. This is the type that will
be
     *                          returned after traversing each individual tree.
     * @param <S>               The final type, after any final normalization at
the
     *                          forest level.
     * @return The aggregated and finalized result after sending a visitor
through
     *         each tree in the forest.
     */
    public <R, S> S traverseForest(float[] point, IVisitorFactory<R>
visitorFactory,
            ConvergingAccumulator<R> accumulator, Function<R, S> finisher) {

        checkNotNull(point, "point must not be null");
        checkArgument(point.length == dimensions, () -> "point.length must equal
to " + dimensions);
        checkNotNull(visitorFactory, "visitorFactory must not be null");
        checkNotNull(accumulator, "accumulator must not be null");
        checkNotNull(finisher, "finisher must not be null");

        return traversalExecutor.traverseForest(point, visitorFactory,
accumulator, finisher);
    }

    /**
     * Visit each of the trees in the forest and combine the individual results
into
     * an aggregate result. A multi-visitor is constructed for each tree using
the
     * visitor factory, and then submitted to
     * {@link RandomCutTree#traverseMulti(float[], IMultiVisitorFactory)}. The
     * results from all the trees are combined using the accumulator and then
     * transformed using the finisher before being returned.
     *
     * @param point             The point that defines the traversal path.
     * @param visitorFactory A factory method which is invoked for each tree to
     *                          construct a multi-visitor.
     * @param accumulator       A function that combines the results from
individual
     *                          trees into an aggregate result.
     * @param finisher          A function called on the aggregate result in order
to
     *                          produce the final result.
     * @param <R>               The visitor result type. This is the type that will
be
     *                          returned after traversing each individual tree.
     * @param <S>               The final type, after any final normalization at
the
     *                          forest level.
     * @return The aggregated and finalized result after sending a visitor
through
     *         each tree in the forest.
     */
    public <R, S> S traverseForestMulti(float[] point, IMultiVisitorFactory<R>
visitorFactory,
            BinaryOperator<R> accumulator, Function<R, S> finisher) {

        checkNotNull(point, "point must not be null");
        checkArgument(point.length == dimensions, () -> "point.length must equal
to " + dimensions);
        checkNotNull(visitorFactory, "visitorFactory must not be null");
        checkNotNull(accumulator, "accumulator must not be null");
        checkNotNull(finisher, "finisher must not be null");
```

```
        return traversalExecutor.traverseForestMulti(point, visitorFactory,
accumulator, finisher);
    }

    /**
     * Visit each of the trees in the forest and combine the individual results
into
     * an aggregate result. A multi-visitor is constructed for each tree using
the
     * visitor factory, and then submitted to
     * {@link RandomCutTree#traverseMulti(float[], IMultiVisitorFactory)}. The
     * results from individual trees are collected using the
     * {@link java.util.stream.Collector} and returned. Trees are visited in
     * parallel using {@link java.util.Collection#parallelStream()}.
     *
     * @param point          The point that defines the traversal path.
     * @param visitorFactory A factory method which is invoked for each tree to
     *                       construct a visitor.
     * @param collector      A collector used to aggregate individual tree
results
     *                       into a final result.
     * @param <R>            The visitor result type. This is the type that will
be
     *                       returned after traversing each individual tree.
     * @param <S>            The final type, after any final normalization at
the
     *                       forest level.
     * @return The aggregated and finalized result after sending a visitor
through
     *         each tree in the forest.
     */
    public <R, S> S traverseForestMulti(float[] point, IMultiVisitorFactory<R>
visitorFactory,
            Collector<R, ?, S> collector) {

        checkNotNull(point, "point must not be null");
        checkArgument(point.length == dimensions, () -> "point.length must equal
to " + dimensions);
        checkNotNull(visitorFactory, "visitorFactory must not be null");
        checkNotNull(collector, "collector must not be null");

        return traversalExecutor.traverseForestMulti(point, visitorFactory,
collector);
    }

    /**
     * Compute an anomaly score for the given point. The point being scored is
     * compared with the points in the sample to compute a measure of how
anomalous
     * it is. Scores are greater than 0, with higher scores corresponding to
bing
     * more anomalous. A threshold of 1.0 is commonly used to distinguish
anomalous
     * points from non-anomalous ones.
     * <p>
     * See {@link AnomalyScoreVisitor} for more details about the anomaly score
     * algorithm.
     *
     * @param point The point being scored.
     * @return an anomaly score for the given point.
     */
    @Deprecated
    public double getAnomalyScore(double[] point) {
        return getAnomalyScore(toFloatArray(point));
    }

    public double getAnomalyScore(float[] point) {
        if (!isOutputReady()) {
            return 0.0;
```

```
        }

        IVisitorFactory<Double> visitorFactory = (tree, x) -> new
AnomalyScoreVisitor(tree.projectToTree(x),
                tree.getMass());
        BinaryOperator<Double> accumulator = Double::sum;
        Function<Double, Double> finisher = x -> x / numberOfTrees;

        return traverseForest(transformToShingledPoint(point), visitorFactory,
accumulator, finisher);
    }

    /**
     * Anomaly score evaluated sequentially with option of early stopping the
early
     * stopping parameter precision gives an approximate solution in the range
     * (1-precision)*score(q)- precision, (1+precision)*score(q) + precision for
the
     * score of a point q. In this function z is hardcoded to 0.1. If this
function
     * is used, then not all the trees will be used in evaluation (but they have
to
     * be updated anyways, because they may be used for the next q). The
advantage
     * is that "almost certainly" anomalies/non-anomalies can be detected easily
     * with few trees.
     *
     * @param point input point q
     * @return anomaly score with early stopping with z=0.1
     */
    @Deprecated
    public double getApproximateAnomalyScore(double[] point) {
        return getApproximateAnomalyScore(toFloatArray(point));
    }

    public double getApproximateAnomalyScore(float[] point) {
        if (!isOutputReady()) {
            return 0.0;
        }

        IVisitorFactory<Double> visitorFactory = (tree, x) -> new
AnomalyScoreVisitor(tree.projectToTree(x),
                tree.getMass());

        ConvergingAccumulator<Double> accumulator = new
OneSidedConvergingDoubleAccumulator(
                DEFAULT_APPROXIMATE_ANOMALY_SCORE_HIGH_IS_CRITICAL,
DEFAULT_APPROXIMATE_DYNAMIC_SCORE_PRECISION,
                DEFAULT_APPROXIMATE_DYNAMIC_SCORE_MIN_VALUES_ACCEPTED,
numberOfTrees);

        Function<Double, Double> finisher = x -> x /
accumulator.getValuesAccepted();

        return traverseForest(transformToShingledPoint(point), visitorFactory,
accumulator, finisher);
    }

    /**
     * Compute an anomaly score attribution DiVector for the given point. The
point
     * being scored is compared with the points in the sample to compute a
measure
     * of how anomalous it is. The result DiVector will contain an anomaly score
in
     * both the positive and negative directions for each dimension of the data.
     * <p>
     * See {@link AnomalyAttributionVisitor} for more details about the anomaly
     * score algorithm.
```

```
     *
     * @param point The point being scored.
     * @return an anomaly score for the given point.
     */
    public DiVector getAnomalyAttribution(double[] point) {
        return getAnomalyAttribution(toFloatArray(point));
    }

    public DiVector getAnomalyAttribution(float[] point) {
        // this will return the same (modulo floating point summation) L1Norm as
        // getAnomalyScore
        if (!isOutputReady()) {
            return new DiVector(dimensions);
        }

        IVisitorFactory<DiVector> visitorFactory = new VisitorFactory<>(
                (tree, y) -> new
AnomalyAttributionVisitor(tree.projectToTree(y), tree.getMass()),
                (tree, x) -> x.lift(tree::liftFromTree));
        BinaryOperator<DiVector> accumulator = DiVector::addToLeft;
        Function<DiVector, DiVector> finisher = x -> x.scale(1.0 /
numberOfTrees);

        return traverseForest(transformToShingledPoint(point), visitorFactory,
accumulator, finisher);
    }

    /**
     * Sequential version of attribution corresponding to
getAnomalyScoreSequential;
     * The high-low sum in the result should be the same as the scalar score
     * computed by {@link #getAnomalyScore(double[])}.
     *
     * @param point The point being scored.
     * @return anomaly attribution for the given point.
     */
    public DiVector getApproximateAnomalyAttribution(double[] point) {
        return getApproximateAnomalyAttribution(toFloatArray(point));
    }

    public DiVector getApproximateAnomalyAttribution(float[] point) {
        if (!isOutputReady()) {
            return new DiVector(dimensions);
        }

        IVisitorFactory<DiVector> visitorFactory = new VisitorFactory<>(
                (tree, y) -> new
AnomalyAttributionVisitor(tree.projectToTree(y), tree.getMass()),
                (tree, x) -> x.lift(tree::liftFromTree));

        ConvergingAccumulator<DiVector> accumulator = new
OneSidedConvergingDiVectorAccumulator(dimensions,
                DEFAULT_APPROXIMATE_ANOMALY_SCORE_HIGH_IS_CRITICAL,
DEFAULT_APPROXIMATE_DYNAMIC_SCORE_PRECISION,
                DEFAULT_APPROXIMATE_DYNAMIC_SCORE_MIN_VALUES_ACCEPTED,
numberOfTrees);

        Function<DiVector, DiVector> finisher = x -> x.scale(1.0 /
accumulator.getValuesAccepted());

        return traverseForest(transformToShingledPoint(point), visitorFactory,
accumulator, finisher);
    }

    /**
     * Compute a density estimate at the given point.
     * <p>
     * See {@link SimpleInterpolationVisitor} and {@link DensityOutput} for more
     * details about the density computation.
```

```
     *
     * @param point The point where the density estimate is made.
     * @return A density estimate.
     */
    @Deprecated
    public DensityOutput getSimpleDensity(double[] point) {
        return getSimpleDensity(toFloatArray(point));
    }

    public DensityOutput getSimpleDensity(float[] point) {

        // density estimation should use sufficiently larger number of samples
        // and only return answers when full

        if (!isOutputReady()) {
            return new DensityOutput(dimensions, sampleSize);
        }

        IVisitorFactory<InterpolationMeasure> visitorFactory = new
VisitorFactory<>((tree,
                y) -> new SimpleInterpolationVisitor(tree.projectToTree(y),
tree.getMass(), 1.0, centerOfMassEnabled),
                (tree, x) -> x.lift(tree::liftFromTree));
        Collector<InterpolationMeasure, ?, InterpolationMeasure> collector =
InterpolationMeasure.collector(dimensions,
                0, numberOfTrees);
        DensityOutput a = new
DensityOutput(traverseForest(transformToShingledPoint(point), visitorFactory,
collector));
        return new DensityOutput(traverseForest(transformToShingledPoint(point),
visitorFactory, collector));
    }

    /**
     * Given a point with missing values, return a collection of treesamples.
These
     * tree samples can be postprocessed in a variety of ways -- primarily to
     * produce summaries and imputation. The treesamples correspond to
pointstore
     * index, distances to tree points (excluding the missing values) the actual
     * point at the leaf and the tree sample is 1 for each tree.
     *
     *
     * @param point          A point with missing values.
     * @param missingIndexes An array containing the indexes of the missing
values
     *                       in the point. The length of the array should be
greater
     *                       than or equal to the number of missing values.
     * @param centrality     a parameter that provides a central estimation
versus a
     *                       more random estimation
     * @return A collection of tree samples
     */
    protected List<ConditionalTreeSample> getConditionalField(float[] point,
int[] missingIndexes, double centrality) {

        // missing indexes can be null -- but then getNearNeighborsInSample may
be more
        // efficient
        checkArgument(centrality >= 0, " cannot be negative");
        checkArgument(centrality <= 1, "centrality needs to be in range [0,1]");
        checkArgument(point != null, " cannot be null");
        if (!isOutputReady()) {
            return new ArrayList<>();
        }

        int[] liftedIndices = transformIndices(missingIndexes, point.length);
```

```
        IMultiVisitorFactory<ConditionalTreeSample> visitorFactory = (tree, y) -
> new ImputeVisitor(y,
                tree.projectToTree(y), liftedIndices,
tree.projectMissingIndices(liftedIndices), centrality,
                tree.getRandomSeed());
        return traverseForestMulti(transformToShingledPoint(point),
visitorFactory, ConditionalTreeSample.collector);
    }

    /**
     * The function returns summary statistics of points close to a query point
     * (with possible missing values). The statics can perform an optional
     * multicentroid clustering
     *
     * @param point                   the query point
     * @param missingIndexes          the list of positions which are missing
     * @param numberOfRepresentatives number of representatives in a cluster
     * @param shrinkage               controls the shape of clusters (=0
corresponds
     *                                to spanning trees, and =1 corresponds to
     *                                centroidal clustering)
     * @param addtypical              an option to perform the clustering/not
     * @param project                 should the clustring/statistics be
computed
     *                                only on the data projected to the entries
in
     *                                missingIndexes
     * @param centrality              how closely should each tree try to
predict
     *                                the missing values =0 implies loosely, =1
     *                                implies closely
     * @param shingleSize             the effective shingleSize -- the
     *                                clustering/statistics would be projected
to
     *                                the last dimension/shinglesize values
     * @return a summary of the predictions returned by each tree
     */
    public SampleSummary getConditionalFieldSummary(float[] point, int[]
missingIndexes, int numberOfRepresentatives,
            double shrinkage, boolean addtypical, boolean project, double
centrality, int shingleSize) {
        // missing indexes can be null -- but then getNearNeighborsInSample may
be more
        // efficient
        checkArgument(centrality >= 0, " cannot be negative");
        checkArgument(centrality <= 1, "centrality needs to be in range [0,1]");
        checkArgument(point != null, " cannot be null");
        if (!isOutputReady()) {
            return new SampleSummary(dimensions);
        }

        int[] liftedIndices = transformIndices(missingIndexes, point.length);
        ConditionalSampleSummarizer summarizer = new
ConditionalSampleSummarizer(liftedIndices,
                transformToShingledPoint(point), centrality, project,
numberOfRepresentatives, shrinkage, shingleSize);
        return summarizer.summarize(getConditionalField(point, missingIndexes,
centrality), addtypical);
    }

    /**
     * Given a point with missing values, return a new point with the missing
values
     * imputed. Each tree in the forest individual produces an imputed value.
The
     * median imputed value is returned. This can be improved using
     * getConditionalSummary or getConditionalField
     *
     * @param point          A point with missing values.
```

```
     * @param missingIndexes An array containing the indexes of the missing
values
     *                       in the point. The length of the array should be
greater
     *                       than or equal to the number of missing values.
     * @return A point with the missing values imputed.
     */

    public float[] imputeMissingValues(float[] point, int[] missingIndexes) {
        return getConditionalFieldSummary(point, missingIndexes, 1, 0, false,
false, 1.0, 1).median;
    }

    // number of missing values is redundant
    @Deprecated
    public float[] imputeMissingValues(float[] point, int numberOfMissingValues,
int[] missingIndexes) {
        return imputeMissingValues(point, missingIndexes);
    }

    @Deprecated
    public double[] imputeMissingValues(double[] point, int
numberOfMissingValues, int[] missingIndexes) {
        return toDoubleArray(imputeMissingValues(toFloatArray(point),
numberOfMissingValues, missingIndexes));
    }

    /**
     * Given an initial shingled point, extrapolate the stream into the future
to
     * produce a forecast. This method is intended to be called when the input
data
     * is being shingled, and it works by imputing forward one shingle block at
a
     * time.
     *
     * @param point        The starting point for extrapolation.
     * @param horizon      The number of blocks to forecast.
     * @param blockSize    The number of entries in a block. This should be the
same
     *                     as the size of a single input to the shingle.
     * @param cyclic       If true then the shingling is cyclic, otherwise it's
a
     *                     sliding shingle.
     * @param shingleIndex If cyclic is true, then this should be the current
index
     *                     in the shingle. That is, the index where the next
point
     *                     added to the shingle would be written. If cyclic is
false
     *                     then this value is not used.
     * @return a forecasted time series.
     */
    @Deprecated
    double[] extrapolateBasic(double[] point, int horizon, int blockSize,
boolean cyclic, int shingleIndex) {
        return toDoubleArray(extrapolateBasic(toFloatArray(point), horizon,
blockSize, cyclic, shingleIndex));
    }

    @Deprecated
    float[] extrapolateBasic(float[] point, int horizon, int blockSize, boolean
cyclic, int shingleIndex) {
        return extrapolateWithRanges(point, horizon, blockSize, cyclic,
shingleIndex, 1.0).values;
    }

    // the following is provided for maximum flexibilty from the calling entity;
    // but likely use is extrapolateFromShingle(), which abstracts away rotation
```

```
    // etc.
    public RangeVector extrapolateWithRanges(float[] point, int horizon, int
blockSize, boolean cyclic,
            int shingleIndex, double centrality) {
        checkArgument(0 < blockSize && blockSize < dimensions,
                "blockSize must be between 0 and dimensions (exclusive)");
        checkArgument(dimensions % blockSize == 0, "dimensions must be evenly
divisible by blockSize");
        checkArgument(0 <= shingleIndex && shingleIndex < dimensions /
blockSize,
                "shingleIndex must be between 0 (inclusive) and dimensions /
blockSize");

        RangeVector result = new RangeVector(blockSize * horizon);
        int[] missingIndexes = new int[blockSize];
        float[] queryPoint = Arrays.copyOf(point, dimensions);

        if (cyclic) {
            extrapolateBasicCyclic(result, horizon, blockSize, shingleIndex,
queryPoint, missingIndexes, centrality);
        } else {
            extrapolateBasicSliding(result, horizon, blockSize, queryPoint,
missingIndexes, centrality);
        }

        return result;
    }

    // external management of shingle; can function for both internal and
external
    // shingling
    // however blocksize has to be externally managed

    @Deprecated
    RangeVector extrapolateFromShingle(float[] shingle, int horizon, int
blockSize, double centrality) {
        return extrapolateWithRanges(shingle, horizon, blockSize,
isRotationEnabled(),
                ((int) nextSequenceIndex()) % shingleSize, centrality);
    }

    /**
     * Given an initial shingled point, extrapolate the stream into the future
to
     * produce a forecast. This method is intended to be called when the input
data
     * is being shingled, and it works by imputing forward one shingle block at
a
     * time. If the shingle is cyclic, then this method uses 0 as the shingle
index.
     *
     * @param point     The starting point for extrapolation.
     * @param horizon   The number of blocks to forecast.
     * @param blockSize The number of entries in a block. This should be the
same as
     *                  the size of a single input to the shingle.
     * @param cyclic    If true then the shingling is cyclic, otherwise it's a
     *                  sliding shingle.
     * @return a forecasted time series.
     */
    @Deprecated
    double[] extrapolateBasic(double[] point, int horizon, int blockSize,
boolean cyclic) {
        return toDoubleArray(extrapolateBasic(toFloatArray(point), horizon,
blockSize, cyclic, 0));
    }

    protected float[] extrapolateBasic(float[] point, int horizon, int
blockSize, boolean cyclic) {
```

```
        return extrapolateBasic(point, horizon, blockSize, cyclic, 0);
    }

    /**
     * Given a shingle builder, extrapolate the stream into the future to
produce a
     * forecast. This method assumes you are passing in the shingle builder used
to
     * preprocess points before adding them to this forest.
     *
     * @param builder The shingle builder used to process points before adding
them
     *                to the forest.
     * @param horizon The number of blocks to forecast.
     * @return a forecasted time series.
     */
    @Deprecated
    public double[] extrapolateBasic(ShingleBuilder builder, int horizon) {
        return
toDoubleArray(extrapolateBasic(toFloatArray(builder.getShingle()), horizon,
builder.getInputPointSize(),
                builder.isCyclic(), builder.getShingleIndex()));
    }

    void extrapolateBasicSliding(RangeVector result, int horizon, int blockSize,
float[] queryPoint,
            int[] missingIndexes, double centrality) {
        int resultIndex = 0;

        Arrays.fill(missingIndexes, 0);
        for (int y = 0; y < blockSize; y++) {
            missingIndexes[y] = dimensions - blockSize + y;
        }

        for (int k = 0; k < horizon; k++) {
            // shift all entries in the query point left by 1 block
            System.arraycopy(queryPoint, blockSize, queryPoint, 0, dimensions -
blockSize);

            SampleSummary imputedSummary =
getConditionalFieldSummary(queryPoint, missingIndexes, 1, 0, false, false,
                    centrality, dimensions / blockSize);
            for (int y = 0; y < blockSize; y++) {
                result.values[resultIndex] = queryPoint[dimensions - blockSize +
y] = imputedSummary.median[y];
                result.lower[resultIndex] = imputedSummary.lower[y];
                result.upper[resultIndex] = imputedSummary.upper[y];
                resultIndex++;
            }
        }
    }

    void extrapolateBasicCyclic(RangeVector result, int horizon, int blockSize,
int shingleIndex, float[] queryPoint,
            int[] missingIndexes, double centrality) {

        int resultIndex = 0;
        int currentPosition = shingleIndex;
        Arrays.fill(missingIndexes, 0);

        for (int k = 0; k < horizon; k++) {
            for (int y = 0; y < blockSize; y++) {
                missingIndexes[y] = (currentPosition + y) % dimensions;
            }

            SampleSummary imputedSummary =
getConditionalFieldSummary(queryPoint, missingIndexes, 1, 0, false, false,
                    centrality, 1);
```

```
            for (int y = 0; y < blockSize; y++) {
                result.values[resultIndex] = queryPoint[(currentPosition + y)
                        % dimensions] = imputedSummary.median[(currentPosition +
y) % dimensions];
                result.lower[resultIndex] =
imputedSummary.lower[(currentPosition + y) % dimensions];
                result.upper[resultIndex] =
imputedSummary.upper[(currentPosition + y) % dimensions];
                resultIndex++;
            }

            currentPosition = (currentPosition + blockSize) % dimensions;
        }
    }

    /**
     * Extrapolate the stream into the future to produce a forecast. This method
is
     * intended to be called when the input data is being shingled internally,
and
     * it works by imputing forward one shingle block at a time.
     *
     * @param horizon The number of blocks to forecast.
     * @return a forecasted time series.
     */
    public double[] extrapolate(int horizon) {
        return toDoubleArray(extrapolateFromCurrentTime(horizon));
    }

    public float[] extrapolateFromCurrentTime(int horizon) {
        checkArgument(internalShinglingEnabled, "incorrect use");
        IPointStore<?, ?> store = stateCoordinator.getStore();
        return extrapolateBasic(lastShingledPoint(), horizon, inputDimensions,
store.isInternalRotationEnabled(),
                ((int) nextSequenceIndex()) % shingleSize);
    }

    /**
     * For each tree in the forest, follow the tree traversal path and return
the
     * leaf node if the standard Euclidean distance between the query point and
the
     * leaf point is smaller than the given threshold. Note that this will not
     * necessarily be the nearest point in the tree, because the traversal path
is
     * determined by the random cuts in the tree. If the same leaf point is
found in
     * multiple trees, those results will be combined into a single Neighbor in
the
     * result.
     *
     * If sequence indexes are disabled for this forest, then the list of
sequence
     * indexes will be empty in returned Neighbors.
     *
     * @param point             A point whose neighbors we want to find.
     * @param distanceThreshold The maximum Euclidean distance for a point to be
     *                          considered a neighbor.
     * @return a list of Neighbors, ordered from closest to furthest.
     */
    @Deprecated
    public List<Neighbor> getNearNeighborsInSample(double[] point, double
distanceThreshold) {
        return getNearNeighborsInSample(toFloatArray(point), distanceThreshold);
    }

    public List<Neighbor> getNearNeighborsInSample(float[] point, double
distanceThreshold) {
        checkNotNull(point, "point must not be null");
```

```
        checkArgument(distanceThreshold > 0, "distanceThreshold must be greater
than 0");

        if (!isOutputReady()) {
            return Collections.emptyList();
        }

        IVisitorFactory<Optional<Neighbor>> visitorFactory = (tree, x) -> new
NearNeighborVisitor(x, distanceThreshold);

        return traverseForest(transformToShingledPoint(point), visitorFactory,
Neighbor.collector());
    }

    /**
     * For each tree in the forest, follow the tree traversal path and return
the
     * leaf node. Note that this will not necessarily be the nearest point in
the
     * tree, because the traversal path is determined by the random cuts in the
     * tree. If the same leaf point is found in multiple trees, those results
will
     * be combined into a single Neighbor in the result.
     *
     * If sequence indexes are disabled for this forest, then sequenceIndexes
will
     * be empty in the returned Neighbors.
     *
     * @param point A point whose neighbors we want to find.
     * @return a list of Neighbors, ordered from closest to furthest.
     */
    @Deprecated
    public List<Neighbor> getNearNeighborsInSample(double[] point) {
        return getNearNeighborsInSample(toFloatArray(point));
    }

    public List<Neighbor> getNearNeighborsInSample(float[] point) {
        return getNearNeighborsInSample(point, Double.POSITIVE_INFINITY);
    }

    /**
     * @return true if all samplers are ready to output results.
     */
    public boolean isOutputReady() {
        return outputReady || (outputReady = stateCoordinator.getTotalUpdates()
>= outputAfter
                &&
components.stream().allMatch(IComponentModel::isOutputReady));
    }

    /**
     * @return true if all samplers in the forest are full.
     */
    public boolean samplersFull() {
        return stateCoordinator.getTotalUpdates() >= sampleSize;
    }

    /**
     * Returns the total number updates to the forest.
     *
     * The count of updates is represented with long type and may overflow.
     *
     * @return the total number of updates to the forest.
     */
    public long getTotalUpdates() {
        return stateCoordinator.getTotalUpdates();
    }

    public void pauseSampling() {
```

```
        updateExecutor.setCurrentlySampling(false);
    }

    public void resumeSampling() {
        updateExecutor.setCurrentlySampling(true);
    }

    public boolean isCurrentlySampling() {
        return updateExecutor.isCurrentlySampling();
    }

    /**
     * an L1 clustering primitive that shows the aggregation of the points
stored in
     * RCF the clustering uses multi-centroid clustering introduced in CURE
     * https://en.wikipedia.org/wiki/CURE_algorithm However CURE also shrunk the
     * well scattered points by a fraction alpha (there by creating new points);
     * while that concept is used herein, the (multi) summarization algorithm
     * changes the distance metric as opposed to creating new points since
     * continuity of values is not an useful assumption in context of RCFs. The
     * usage of distance metric is similar to the discussion in
     * https://en.wikipedia.org/wiki/Data_stream_clustering See the examples
package
     * for an example of dynamic summarization. /
     *
     * @param maxAllowed              maximum number of clusters one is willing
to
     *                                see
     * @param shrinkage               a parameter that controls between
spherical
     *                                nature (=1) and MST (=0), this corresponds
to
     *                                the parameter alpha in the description
above
     * @param numberOfRepresentatives number of centroids used to represent a
     *                                cluster, this is the parameter c in the
     *                                description of CURE
     * @param separationRatio         a parameter in [0,1] that controls how
     *                                zealously should the algorithm reduce the
     *                                number of clusters a default value of 0.8
is a
     *                                reasonable value for many settings. A
value
     *                                close to 0 would tend to merge eveything
into
     *                                a single cluster. The option is provided
since
     *                                it can be of use in the future to produce
     *                                dendograms and similar information.
     * @param distance                a distance function for points
     * @param previous                a (possibly null) list of previous
clustering
     *                                obtained. If the list is non-null then the
     *                                representatives of the previous cluster
would
     *                                be added as zero weight points, ensuring
that
     *                                the summarization is more smooth (in
contrast
     *                                to two independent summarizations). The
zero
     *                                weight points of the past can serve as
     *                                representatives of the current clustering.
     * @return a list of clusters
     */
    public List<ICluster<float[]>> summarize(int maxAllowed, double shrinkage,
int numberOfRepresentatives,
            double separationRatio, BiFunction<float[], float[], Double>
distance, List<ICluster<float[]>> previous) {
```

```
        return stateCoordinator.getStore().summarize(maxAllowed, shrinkage,
numberOfRepresentatives, separationRatio,
                distance, previous);
    }

    // same as above with default filled in
    public List<ICluster<float[]>> summarize(int maxAllowed, double shrinkage,
int numberOfRepresentatives,
            List<ICluster<float[]>> previous) {
        return summarize(maxAllowed, shrinkage, numberOfRepresentatives,
DEFAULT_SEPARATION_RATIO_FOR_MERGE,
                Summarizer::L1distance, previous);
    }

    public static class Builder<T extends Builder<T>> {

        // We use Optional types for optional primitive fields when it doesn't
make
        // sense to use a constant default.

        private int dimensions;
        private int sampleSize = DEFAULT_SAMPLE_SIZE;
        private Optional<Integer> outputAfter = Optional.empty();
        private int numberOfTrees = DEFAULT_NUMBER_OF_TREES;
        private Optional<Double> timeDecay = Optional.empty();
        private Optional<Long> randomSeed = Optional.empty();
        private boolean storeSequenceIndexesEnabled =
DEFAULT_STORE_SEQUENCE_INDEXES_ENABLED;
        private boolean centerOfMassEnabled = DEFAULT_CENTER_OF_MASS_ENABLED;
        private boolean parallelExecutionEnabled =
DEFAULT_PARALLEL_EXECUTION_ENABLED;
        private Optional<Integer> threadPoolSize = Optional.empty();
        private boolean directLocationMapEnabled = DEFAULT_DIRECT_LOCATION_MAP;
        private double boundingBoxCacheFraction =
DEFAULT_BOUNDING_BOX_CACHE_FRACTION;
        private int shingleSize = DEFAULT_SHINGLE_SIZE;

        private boolean internalShinglingEnabled =
DEFAULT_INTERNAL_SHINGLING_ENABLED;
        protected boolean internalRotationEnabled =
DEFAULT_INTERNAL_ROTATION_ENABLED;
        protected Optional<Integer> initialPointStoreSize = Optional.empty();
        protected double initialAcceptFraction =
DEFAULT_INITIAL_ACCEPT_FRACTION;

        public T dimensions(int dimensions) {
            this.dimensions = dimensions;
            return (T) this;
        }

        public T sampleSize(int sampleSize) {
            this.sampleSize = sampleSize;
            return (T) this;
        }

        public T outputAfter(int outputAfter) {
            this.outputAfter = Optional.of(outputAfter);
            return (T) this;
        }

        public T numberOfTrees(int numberOfTrees) {
            this.numberOfTrees = numberOfTrees;
            return (T) this;
        }

        public T shingleSize(int shingleSize) {
            this.shingleSize = shingleSize;
            return (T) this;
        }
```

```
        public T timeDecay(double timeDecay) {
            this.timeDecay = Optional.of(timeDecay);
            return (T) this;
        }

        public T randomSeed(long randomSeed) {
            this.randomSeed = Optional.of(randomSeed);
            return (T) this;
        }

        public T centerOfMassEnabled(boolean centerOfMassEnabled) {
            this.centerOfMassEnabled = centerOfMassEnabled;
            return (T) this;
        }

        public T parallelExecutionEnabled(boolean parallelExecutionEnabled) {
            this.parallelExecutionEnabled = parallelExecutionEnabled;
            return (T) this;
        }

        public T threadPoolSize(int threadPoolSize) {
            this.threadPoolSize = Optional.of(threadPoolSize);
            return (T) this;
        }

        public T initialPointStoreSize(int initialPointStoreSize) {
            this.initialPointStoreSize = Optional.of(initialPointStoreSize);
            return (T) this;
        }

        public T storeSequenceIndexesEnabled(boolean
storeSequenceIndexesEnabled) {
            this.storeSequenceIndexesEnabled = storeSequenceIndexesEnabled;
            return (T) this;
        }

        @Deprecated
        public T compact(boolean compact) {
            return (T) this;
        }

        public T internalShinglingEnabled(boolean internalShinglingEnabled) {
            this.internalShinglingEnabled = internalShinglingEnabled;
            return (T) this;
        }

        public T internalRotationEnabled(boolean internalRotationEnabled) {
            this.internalRotationEnabled = internalRotationEnabled;
            return (T) this;
        }

        @Deprecated
        public T dynamicResizingEnabled(boolean dynamicResizingEnabled) {
            return (T) this;
        }

        @Deprecated
        public T precision(Precision precision) {
            return (T) this;
        }

        public T boundingBoxCacheFraction(double boundingBoxCacheFraction) {
            this.boundingBoxCacheFraction = boundingBoxCacheFraction;
            return (T) this;
        }

        public T initialAcceptFraction(double initialAcceptFraction) {
            this.initialAcceptFraction = initialAcceptFraction;
```

```
            return (T) this;
        }

        public RandomCutForest build() {
            return new RandomCutForest(this);
        }

        public Random getRandom() {
            // If a random seed was given, use it to create a new Random.
Otherwise, call
            // the 0-argument constructor
            return randomSeed.map(Random::new).orElseGet(Random::new);
        }
    }

    /**
     * Score a point using the given scoring functions.
     *
     * @param point                    input point being scored
     * @param ignoreLeafMassThreshold said threshold
     * @param seen                     the function that applies if input is
equal to
     *                                 a previously seen sample in a leaf
     * @param unseen                   if the input does not have a match in the
     *                                 leaves
     * @param damp                     damping function based on the duplicity of
the
     *                                 previously seen samples
     * @return anomaly score
     */
    public double getDynamicScore(float[] point, int ignoreLeafMassThreshold,
BiFunction<Double, Double, Double> seen,
            BiFunction<Double, Double, Double> unseen, BiFunction<Double,
Double, Double> damp) {

        checkArgument(ignoreLeafMassThreshold >= 0, "ignoreLeafMassThreshold
should be greater than or equal to 0");

        if (!isOutputReady()) {
            return 0.0;
        }

        VisitorFactory<Double> visitorFactory = new VisitorFactory<>((tree, y) -
> new DynamicScoreVisitor(
                tree.projectToTree(y), tree.getMass(), ignoreLeafMassThreshold,
seen, unseen, damp));
        BinaryOperator<Double> accumulator = Double::sum;

        Function<Double, Double> finisher = sum -> sum / numberOfTrees;

        return traverseForest(transformToShingledPoint(point), visitorFactory,
accumulator, finisher);
    }

    /**
     * Similar to above but now the scoring takes in a function of Bounding Box
to
     * probabilities (vector over the dimensions); and produces a score af-if
the
     * tree were built using that function (when in reality the tree is an RCF).
     * Changing the defaultRCFgVec function to some other function f() will
provide
     * a mechanism of dynamic scoring for trees that are built using f() which
is
     * the purpose of TransductiveScalarScore visitor. Note that the answer is
an
     * MCMC simulation and is not normalized (because the scoring functions are
     * flexible and unknown) and over a small number of trees the errors can be
     * large specially if vecSep is very far from defaultRCFgVec
```

```
     *
     * Given the large number of possible sources of distortion,
ignoreLeafThreshold
     * is not supported.
     *
     * @param point  point to be scored
     * @param seen   the score function for seen point
     * @param unseen score function for unseen points
     * @param damp   dampening the score for duplicates
     * @param vecSep the function of (BoundingBox) -&gt; array of probabilities
     * @return the simuated score
     */

    public double getDynamicSimulatedScore(float[] point, BiFunction<Double,
Double, Double> seen,
            BiFunction<Double, Double, Double> unseen, BiFunction<Double,
Double, Double> damp,
            Function<IBoundingBoxView, double[]> vecSep) {

        if (!isOutputReady()) {
            return 0.0;
        }

        VisitorFactory<Double> visitorFactory = new VisitorFactory<>(
                (tree, y) -> new
SimulatedTransductiveScalarScoreVisitor(tree.projectToTree(y), tree.getMass(),
seen,
                        unseen, damp, CommonUtils::defaultRCFgVecFunction,
vecSep));
        BinaryOperator<Double> accumulator = Double::sum;

        Function<Double, Double> finisher = sum -> sum / numberOfTrees;

        return traverseForest(transformToShingledPoint(point), visitorFactory,
accumulator, finisher);
    }

    /**
     * Score a point using the given scoring functions. This method will
     * short-circuit before visiting all trees if the scores that are returned
from
     * a subset of trees appears to be converging to a given value. See
     * {@link OneSidedConvergingDoubleAccumulator} for more about convergence.
     *
     * @param point                 input point
     * @param precision             controls early convergence
     * @param highIsCritical        this is true for the default scoring
function.
     *                              If the user wishes to use a different
scoring
     *                              function where anomaly scores are low
values
     *                              (for example, height in tree) then this
should
     *                              be set to false.
     * @param ignoreLeafMassThreshold said threshold
     * @param seen                  scoring function when the input matches
some
     *                              tuple in the leaves
     * @param unseen                scoring function when the input is not
found
     * @param damp                  dampening function for duplicates which
are
     *                              same as input (applies with seen)
     * @return the dynamic score under sequential early stopping
     */
    public double getApproximateDynamicScore(float[] point, double precision,
boolean highIsCritical,
```

```
            int ignoreLeafMassThreshold, BiFunction<Double, Double, Double>
seen,
            BiFunction<Double, Double, Double> unseen, BiFunction<Double,
Double, Double> damp) {

        checkArgument(ignoreLeafMassThreshold >= 0, "ignoreLeafMassThreshold
should be greater than or equal to 0");

        if (!isOutputReady()) {
            return 0.0;
        }

        VisitorFactory<Double> visitorFactory = new VisitorFactory<>((tree, y) -
> new DynamicScoreVisitor(
                tree.projectToTree(y), tree.getMass(), ignoreLeafMassThreshold,
seen, unseen, damp));

        ConvergingAccumulator<Double> accumulator = new
OneSidedConvergingDoubleAccumulator(highIsCritical, precision,
                DEFAULT_APPROXIMATE_DYNAMIC_SCORE_MIN_VALUES_ACCEPTED,
numberOfTrees);

        Function<Double, Double> finisher = x -> x /
accumulator.getValuesAccepted();

        return traverseForest(transformToShingledPoint(point), visitorFactory,
accumulator, finisher);
    }

    /**
     * Same as above, but for dynamic scoring. See the params of
     * getDynamicScoreParallel
     *
     * @param point                   point to be scored
     * @param ignoreLeafMassThreshold said threshold
     * @param seen                    score function for seen points
     * @param unseen                  score function for unseen points
     * @param newDamp                 dampening function for duplicates in the
seen
     *                                function
     * @return dynamic scoring attribution DiVector
     */
    public DiVector getDynamicAttribution(float[] point, int
ignoreLeafMassThreshold,
            BiFunction<Double, Double, Double> seen, BiFunction<Double, Double,
Double> unseen,
            BiFunction<Double, Double, Double> newDamp) {

        if (!isOutputReady()) {
            return new DiVector(dimensions);
        }

        VisitorFactory<DiVector> visitorFactory = new VisitorFactory<>(
                (tree, y) -> new
DynamicAttributionVisitor(tree.projectToTree(y), tree.getMass(),
                        ignoreLeafMassThreshold, seen, unseen, newDamp),
                (tree, x) -> x.lift(tree::liftFromTree));
        BinaryOperator<DiVector> accumulator = DiVector::addToLeft;
        Function<DiVector, DiVector> finisher = x -> x.scale(1.0 /
numberOfTrees);

        return traverseForest(transformToShingledPoint(point), visitorFactory,
accumulator, finisher);
    }

    /**
     * Atrribution for dynamic sequential scoring; getL1Norm() should agree with
     * getDynamicScoringSequential
     *
```

```
     * @param point                   input
     * @param precision               parameter to stop early stopping
     * @param highIsCritical          are high values anomalous (otherwise low
     *                                values are anomalous)
     * @param ignoreLeafMassThreshold we ignore leaves with mass equal/below *
     *                                threshold
     * @param seen                    function for scoring points that have been
     *                                seen before
     * @param unseen                  function for scoring points not seen in
tree
     * @param newDamp                 dampening function based on duplicates
     * @return attribution DiVector of the score
     */
    public DiVector getApproximateDynamicAttribution(float[] point, double
precision, boolean highIsCritical,
            int ignoreLeafMassThreshold, BiFunction<Double, Double, Double>
seen,
            BiFunction<Double, Double, Double> unseen, BiFunction<Double,
Double, Double> newDamp) {

        if (!isOutputReady()) {
            return new DiVector(dimensions);
        }

        VisitorFactory<DiVector> visitorFactory = new VisitorFactory<>((tree, y)
-> new DynamicAttributionVisitor(y,
                tree.getMass(), ignoreLeafMassThreshold, seen, unseen, newDamp),
                (tree, x) -> x.lift(tree::liftFromTree));

        ConvergingAccumulator<DiVector> accumulator = new
OneSidedConvergingDiVectorAccumulator(dimensions,
                highIsCritical, precision,
DEFAULT_APPROXIMATE_DYNAMIC_SCORE_MIN_VALUES_ACCEPTED, numberOfTrees);

        Function<DiVector, DiVector> finisher = vector -> vector.scale(1.0 /
accumulator.getValuesAccepted());

        return traverseForest(transformToShingledPoint(point), visitorFactory,
accumulator, finisher);
    }

}
```

Файл rcf_model.py – обгортка навколо реалізації алгоритма RCF на Java

```
from typing import List, Optional, Tuple, Any

import numpy as np
import logging
from com.amazon.randomcutforest import RandomCutForest
import jpype

class RandomCutForestModel:
    """
    Random Cut Forest Python Binding around the AWS Random Cut Forest Official
Java version:
    https://github.com/aws/random-cut-forest-by-aws
    """

    def __init__(self, forest: RandomCutForest = None, shingle_size: int = 8,
                 num_trees: int = 100, random_seed: int = None,
                 sample_size: int = 256, parallel_execution_enabled: bool =
True,
                 thread_pool_size: Optional[int] = None, lam: float=0.0001,
                 output_after: int=256):
        if forest is not None:
            self.forest = forest
        else:
            builder = RandomCutForest.builder().numberOfTrees(num_trees). \
                sampleSize(sample_size). \
                dimensions(shingle_size). \
                storeSequenceIndexesEnabled(True). \
                centerOfMassEnabled(True). \
                parallelExecutionEnabled(parallel_execution_enabled). \
                timeDecay(lam). \
                outputAfter(output_after)
            if thread_pool_size is not None:
                builder.threadPoolSize(thread_pool_size)

            if random_seed is not None:
                builder = builder.randomSeed(random_seed)

            self.forest = builder.build()

    def score(self, point: List[float]) -> float:
        """
        Compute an anomaly score for the given point.

        Parameters
        ----------
        point: List[float]
            A data point with shingle size

        Returns
        -------
        float
            The anomaly score for the given point

        """
        return self.forest.getAnomalyScore(point)

    def update(self, point: List[float]):
        """
        Update the model with the data point.

        Parameters
        ----------
        point: List[float]
            Point with shingle size
        """
        self.forest.update(point)
```

```
    def impute(self, point: List[float]) -> List[float]:
        """
        Given a point with missing values, return a new point with the missing
values imputed. Each tree in the forest
        individual produces an imputed value. For 1-dimensional points, the
median imputed value is returned. For
        points with more than 1 dimension, the imputed point with the 25th
percentile anomaly score is returned.

        Parameters
        ----------
        point: List[float]
            The point with shingle size

        Returns
        -------
        List[float]
            The imputed point.
        """
        num_missing = np.isnan(point).sum()
        if num_missing == 0:
            return point
        missing_index = np.argwhere(np.isnan(point)).flatten()
        imputed_shingle = list(self.forest.imputeMissingValues(point,
num_missing, missing_index))
        return imputed_shingle

    def forecast(self, point: List[float]) -> float:
        """
        Given one shingled data point, return one step forecast containing the
next value.

        Parameters
        ----------
        point: List[float]
            The point with shingle size

        Returns
        -------
        float
            Forecast value of next timestamp.

        """
        val = list(self.forest.extrapolateBasic(point, 1, 1, False, 0))[0]
        return val

    @property
    def shingle_size(self) -> int:
        """
        Returns
        -------
        int
            Shingle size of random cut trees.
        """
        return self.forest.getDimensions()

    def get_attribution(self, point: List[float]) -> Tuple[List[float],
List[float]]:
        try:
            attribution_di_vec: Any = self.forest.getAnomalyAttribution(point)
            low: List[float] = list(attribution_di_vec.low)
            high: List[float] = list(attribution_di_vec.high)
            return low, high
        except jpype.JException as exception:
            logging.info("Error when loading the model: %s",
exception.message())
            logging.info("Stack track: %s", exception.stacktrace())
            raise exception
```