Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:
«СИСТЕМА АВТОМАТИЗАЦІЇ РЕАГУВАННЯ НА ІНЦИДЕНТИ БЕЗПЕКИ»

Виконав: студент 2 курсу, групи 1БС-23м
спеціальності 125 Кібербезпека та захист інформації
_____ Владислав БУГАЄЦЬ

Керівник: к. т. н., доцент каф. ЗІ
_____ Леонід КУПЕРШТЕЙН
«13» 12 2024 р.

Опонент: к. т. н., доцент каф. ПЗ
_____ Олена КОВАЛЕНКО
«13» 12 2024 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«13» 12 2024 р.

Вінниця ВНТУ – 2024 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 125 «Кібербезпека та захист інформації»
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., проф.

Володимир ЛУЖЕЦЬКИЙ

(підпис)

«18» 09 2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бугайцю Владиславу Сергійовичу

1. Тема роботи: «Система автоматизації реагування на інциденти безпеки» керівник роботи: Куперштейн Леонід Михайлович, к. т. н., доцент кафедри ЗІ,затверджені наказом ректора ВНТУ № 310 від 17.09.2024
2. Строк подання студентом роботи 16.12.2024
3. Вихідні дані до роботи:
   - збір подій безпеки;
   - інтеграція з засобами безпеки;
   - аналіз подій;
   - реагування на події.
4. Зміст текстової частини: Вступ. 1. Аналіз предметної області. 2. Розробка архітектурних рішень. 3. Розробка та тестування рішення. 4. Економічний розділ. Висновки. Список використаних джерел. Додатки.
6. Перелік ілюстративного матеріалу: Структурна схема системи автоматизації реагування на інциденти (плакат, А4). Схема роботи черг (плакат, А4). Схема роботи механізму автентифікації (плакат, А4). Схема зв`язків з системами безпеки (плакат, А4). Структура бази даних (плакат, А4). Життєвий цикл розробки плейбуків (плакат, А4). Схема автоматизованого реагування на інциденти (плакат, А4). Схема плейбуку захисту облікових записів (плакат, А4). Схема роботи плейбуку безпеки

електронної пошти (плакат, А4). Схема роботи плейбуку захисту хмарного середовища (плакат, А4). Схема роботи плейбуку мережевої безпеки (плакат, А4). Схема роботи плейбуку протидії шкідливому програмному забезпеченню (плакат, А4).

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Куперштейн Л. М., к.т.н., доц. каф. ЗІ	11.09.24	22.09
2	Куперштейн Л. М., к.т.н., доц. каф. ЗІ	23.09.24	12.10.
3	Куперштейн Л. М., к.т.н., доц. каф. ЗІ	13.10.2024	10.11.24
4	Ратушняк О. Г., доц. кафедри ЕПВМ	01.12.24	04.12.24

8. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	01.09.2024 – 10.09.2024	
2	Аналіз літературних джерел за напрямком бакалаврської кваліфікаційної роботи	11.09.2024 – 22.09.2024	
3	Розробка рішень	23.09.2024 – 12.10.2024	
4	Практична реалізація, моделювання, експериментування, результати	13.10.2024 – 10.11.2024	
5	Розробка розділу тестування і обґрунтування доцільності розробки	11.11.2024 – 25.11.2024	
9	Попередній захист та доопрацювання МКР	28.11.2024 – 01.12.2024	
6	Розробка економічного розділу	01.12.2024 – 04.12.2024	
7	Аналіз виконання ТЗ, висновки	05.12.2024 – 06.12.2024	
8	Оформлення пояснювальної записки	07.12.2024 – 12.12.2024	
10	Представлення МКР до захисту	13.12.2024 – 16.12.2024	
11	Захист МКР	17.12.2024 – 20.12.2024	

Студент _____ Владислав БУГАЄЦЬ

Керівник роботи _____ Леонід КУПЕРШТЕЙН
(підпис)

АНОТАЦІЯ

УДК 004.056

Бугаєць В. Система автоматизації реагування на інциденти безпеки. Магістерська кваліфікаційна робота зі спеціальності 125 – Кібербезпека, освітня програма – Безпека інформаційних і комунікаційних систем. Вінниця: ВНТУ, 2024. 100 с.

Укр. мовою. Бібліогр.: 32 назв; рис.: 35; табл.: 10.

Магістерська кваліфікаційна робота присвячена розробці системи автоматизації реагування на інциденти безпеки (SOAR–системи), яка дозволяє ефективно і оперативно виявляти, аналізувати та нейтралізувати загрози в інформаційній інфраструктурі організації. У роботі проаналізовано сучасні кіберзагрози, традиційні підходи до реагування та існуючі SOAR-рішення, виявлено їхні недоліки, що ускладнюють впровадження, особливо для організацій з обмеженими ресурсами.

У роботі описано архітектуру системи, методи інтеграції з різними джерелами даних і сервісами безпеки, механізми авторизації та аутентифікації, а також підходи до створення і використання плейбуків для автоматизованого реагування. Продемонстровано практичну реалізацію окремих компонентів, проілюстровано взаємодію фронтенду та бекенду, налаштування контейнеризації та тестування рішення.

Результати роботи можуть бути впроваджені в реальні інформаційні системи з метою підвищення кібербезпеки, скорочення часу реагування на інциденти та оптимізації використання ресурсів команд безпеки.

Ключові слова: SOAR, автоматизація реагування, оркестрація, моніторинг, реагування на інциденти, плейбуки, кібербезпека.

ABSTRACT

Buhaiets V. Information technology for software data security monitoring. Master's thesis of the specialty 125 – Cybersecurity, educational program – Security of information and communication systems. Vinnytsia: VNTU, 2023. ?? p.

In Ukrainian. Bibliography: 32 titles; Figures: 35; Tables: 10.

The master's thesis is dedicated to the development of an automated security incident response system (SOAR system), which enables efficient and prompt detection, analysis, and neutralization of threats in the organization's information infrastructure. The work analyzes current cyber threats, traditional response approaches, and existing SOAR solutions, identifying their shortcomings that complicate implementation, especially for organizations with limited resources.

The thesis describes the system architecture, methods of integration with various data sources and security services, authorization and authentication mechanisms, as well as approaches to creating and using playbooks for automated response. It demonstrates the practical implementation of individual components, illustrates the interaction between the frontend and backend, containerization setup, and solution testing.

The results of the work can be implemented in real information systems to enhance cybersecurity, reduce incident response time, and optimize the use of security team resources.

Keywords: SOAR, response automation, orchestration, monitoring, incident response, playbooks, cybersecurity.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології стали невід'ємною частиною життя суспільства, проникаючи в усі сфери діяльності, від економіки та промисловості до освіти та охорони здоров'я. Зі зростанням обсягу даних і складності інформаційних систем підвищується й рівень загроз, пов'язаних із кібербезпекою. Кількість кібератак та інцидентів безпеки невпинно зростає, що призводить до значних фінансових втрат, компрометації конфіденційної інформації та підриву репутації організацій.

Своєчасне та ефективне реагування на інциденти безпеки є критичним фактором у мінімізації негативних наслідків кібератак. Традиційні методи реагування, які базуються на ручному аналізі та втручанні фахівців, часто виявляються недостатньо швидкими та ефективними. Це створює потребу в автоматизованих системах, здатних швидко виявляти та реагувати на загрози в режимі реального часу.

Актуальність. У зв'язку з динамічним розвитком кіберзагроз та збільшенням кількості інцидентів безпеки, автоматизація процесів реагування стає не просто бажаною, а необхідною. Сучасні організації потребують інструментів, які дозволяють швидко ідентифікувати загрози, аналізувати їх та вживати відповідних заходів для їх нейтралізації. Автоматизовані системи реагування на інциденти безпеки здатні значно знизити час реакції, мінімізувати людський фактор та підвищити загальний рівень захищеності інформаційних систем.

Об'єктом дослідження є процес реагування на інциденти інформаційної безпеки в організаціях.

Предметом дослідження є методи та засоби автоматизації реагування на кіберінциденти в інформаційних системах.

Метою магістерської кваліфікаційної роботи є підвищення ефективності реагування на інциденти безпеки шляхом розробки системи автоматизації, яка дозволяє оперативно виявляти та нейтралізувати кіберзагрози.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– провести аналіз сучасних кіберзагроз та методів реагування на них;
– дослідити існуючі системи автоматизації реагування на інциденти безпеки;
– розробити вимоги та концепцію системи автоматизації реагування;
– створити архітектуру та алгоритми функціонування системи;
– реалізувати прототип системи та провести його тестування;
– оцінити результати впровадження та визначити напрями подальшого розвитку;

Наукова новизна роботи полягає у покращенні технології реагування на інциденти безпеки за рахунок автоматизації процесів з використанням структурованих сценаріїв реагування, що дозволяє скоротити час реагування в 2 рази.

Практична цінність роботи полягає у створенні прототипу системи автоматизації реагування на інциденти безпеки, який може бути інтегрований у існуючі інфраструктури організацій. Це сприятиме підвищенню рівня кібербезпеки, зниженню ризиків та мінімізації потенційних збитків від кібератак.

Окремі результати даної роботи були впроваджені в роботі інформаційної системи підприємства ТОВ "НП Діджитал".

Результати здійснених досліджень під час виконання магістерської кваліфікаційної роботи доповідались на Міжнародній науково-практична конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)».

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасні кіберзагрози та інциденти безпеки

У сучасному цифровому середовищі кіберзагрози постійно еволюціонують, стаючи все більш складними. Це зумовлено стрімким розвитком технологій, зростанням кількості підключених пристроїв та збільшенням обсягу даних. Кіберзлочинці активно використовують ці тенденції для розробки нових методів атак та експлуатації вразливостей у системах безпеки [1].

Хмарні технології стали привабливою мішенню для зловмисників [2]. Перенесення даних та сервісів у хмару підвищує ефективність бізнес-процесів, але водночас створює нові ризики. Недостатньо налаштовані хмарні середовища можуть призвести до витоку конфіденційної інформації або несанкціонованого доступу до критичних систем. Зловмисники можуть експлуатувати слабкі місця в конфігураціях, використовувати фішингові атаки для отримання облікових даних або здійснювати атаки на рівні хмарних API.

Соціальні мережі стали платформою для поширення фішингових атак та дезінформації [1]. Зловмисники використовують персональні дані, розміщені в соціальних профілях, для створення цільових атак, що робить їх більш ефективними. Крім того, соціальні мережі можуть бути використані для поширення шкідливих посилань або файлів серед широкої аудиторії.

Програми-вимагачі (ransomware) продовжують бути однією з найбільш серйозних загроз для організацій та окремих користувачів [2]. Ці програми шифрують дані на пристроях жертв та вимагають викуп за їх відновлення. Злочинці стали більш організованими, створюючи моделі "вимагач як послуга" (RaaS), де інші зловмисники можуть купувати або орендувати інструменти для проведення атак. Це призводить до збільшення кількості інцидентів та підвищення розміру викупів.

Атаки типу "нульового дня" представляють особливу небезпеку, оскільки вони експлуатують вразливості, про які ще не відомо виробникам програмного

забезпечення та засобам захисту. Це дає зловмисникам можливість проникати в системи без виявлення протягом тривалого часу. Виявлення та захист від таких атак вимагає проактивного підходу та використання передових технологій аналізу загроз.

Цільові атаки (APT – Advanced Persistent Threats), які часто спонсоруються державами або великими організаціями, спрямовані на конкретні компанії або галузі з метою шпигунства, крадіжки інтелектуальної власності або саботажу [2]. Ці атаки характеризуються високим рівнем підготовки, використанням складних методів проникнення та маскування, а також тривалим перебуванням у системах жертви. Діаграма, яка зображена на рисунку 1.1 відображає розподіл кібератак за 2024 рік в світі.

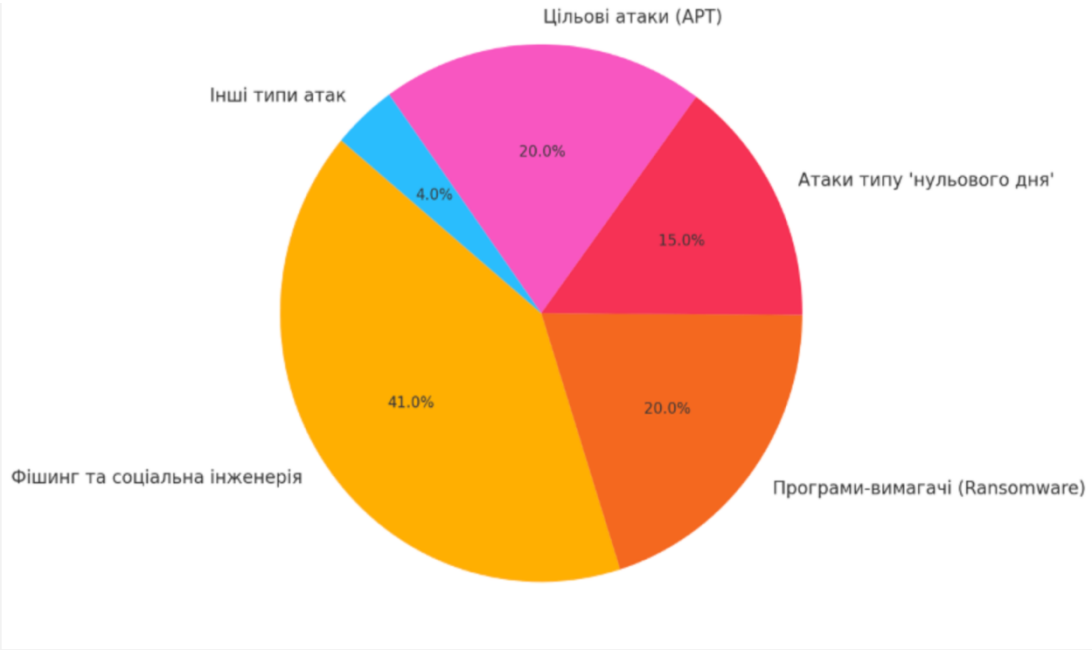


Рисунок 1.1 – Розподіл кібератак за 2024 рік

В даній діаграмі де найбільшу частку займають фішинг та соціальна інженерія (41%), що свідчить про високий рівень маніпуляцій для отримання доступу до даних користувачів. Програми-вимагачі, які становлять 20%, залишаються значною загрозою, спричиняючи блокування даних із подальшим вимаганням викупу. Атаки типу "нульового дня" охоплюють 15% і використовують невідомі вразливості, що робить їх особливо небезпечними.

Цільові атаки (APT) також займають 20%, відображаючи загрози від високорозвинених і довготривалих кампаній, зазвичай підтримуваних державами. Інші типи атак становлять 4%, включаючи менш поширені методи, такі як DDoS та атаки на ланцюги постачання, що свідчить про багатогранність можливих небезпек.

У відповідь на зростаючу кількість та складність кіберзагроз, організації змушені переглядати свої стратегії кібербезпеки. Традиційні методи, засновані на статичних правилах та сигнатурах, більше не можуть ефективно захищати від сучасних атак. Необхідність швидкого реагування на інциденти та адаптації до нових загроз привела до розвитку систем автоматизованого реагування на інциденти безпеки [4].

Автоматизація стала ключовим елементом у боротьбі з кіберзагрозами. Системи автоматизованого реагування використовують штучний інтелект та машинне навчання для аналізу великої кількості даних, виявлення аномалій та прийняття рішень у режимі реального часу [4]. Це дозволяє швидко реагувати на інциденти, знижуючи ризик компрометації систем та мінімізуючи потенційні збитки.

Наприклад, системи SOAR (Security Orchestration, Automation, and Response) інтегрують різні інструменти та процеси кібербезпеки, автоматизуючи рутинні завдання та забезпечуючи узгоджене реагування на загрози [4]. Вони можуть автоматично збирати інформацію про інцидент, аналізувати його вплив, приймати рішення щодо реагування та документувати всі дії для подальшого аналізу.

Однак автоматизація процесів реагування на інциденти безпеки мають свої недоліки. Існують певні виклики, пов'язані з її впровадженням та використанням. Для ефективної роботи автоматизованих систем необхідні великі обсяги якісних даних, оскільки недостатня або неточна інформація може призвести до помилкових спрацювань або пропуску реальних загроз. Автоматизовані рішення повинні безшовно взаємодіяти з наявними інструментами безпеки, що може вимагати значних ресурсів та часу на налаштування та тестування. Крім того,

автоматизація не може повністю замінити експертів з кібербезпеки. Необхідні кваліфіковані фахівці, які можуть інтерпретувати результати роботи систем, приймати стратегічні рішення та реагувати на нестандартні ситуації.

Проте автоматичне прийняття рішень, особливо якщо воно включає блокування користувачів або доступу до ресурсів, може викликати питання щодо безперебійності бізнес процесів. Також самі системи автоматизації можуть стати цілями кібератак. Зловмисники можуть намагатися атакувати або обійти автоматизовані системи реагування, тому вони повинні бути захищені та регулярно оновлюватися.

Для ефективного впровадження автоматизації в процес реагування на сповіщення систем безпеки необхідний комплексний підхід, який включає оцінку ризиків та потреб, розуміння специфіки організації, її вразливостей та потенційних загроз. Розробка стратегії автоматизації передбачає визначення цілей, обсягу автоматизації, вибір інструментів та технологій. Пілотне впровадження та тестування системи в контрольованому середовищі дозволяє виявити та усунути недоліки. Важливим є навчання персоналу, підготовка співробітників до роботи з автоматизованими системами, розвиток навичок аналізу та реагування [4].

1.2 Підходи до реагування на інциденти безпеки

Традиційно, реагування на інциденти базувалося на ручному аналізі та втручанні фахівців з кібербезпеки. Цей підхід передбачає, що спеціалісти моніторять системи та мережі, виявляючи аномалії та підозрілу активність шляхом аналізу логів та журналів подій [5]. Використання систем управління подіями та інформацією безпеки (SIEM) стало стандартом для збору та аналізу великого обсягу даних [5]. Після виявлення потенційної загрози аналітики виконують необхідні дії для її усунення, такі як ізоляція уражених систем, видалення шкідливого програмного забезпечення та відновлення даних з

резервних копій. Проте, такий метод має свої недоліки. Загальний вигляд процесу реагування на інцидент можна побачити на рисунку 1.2.

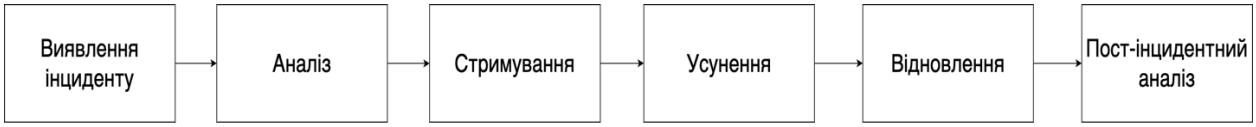


Рисунок 1.2 – Схема процесів реагування на інциденти

Виявлення інциденту — це перший крок, де система безпеки фіксує можливу загрозу. Інструменти на кшталт SIEM чи антивірусів моніторять активність в мережі, аналізують журнали подій та інші дані, щоб знайти підозрілі дії. Якщо така активність помічається, створюється сповіщення, яке запускає процес реагування [6].

Далі йде аналіз, де інформація про інцидент ретельно обробляється, щоб визначити його джерело та серйозність. Спеціалісти або автоматизовані системи перевіряють, чи є інцидент справжньою загрозою, оцінюють її вплив та задають пріоритет, щоб критичні випадки були оброблені першочергово.

Після цього відбувається стримування загрози — етап, де вживаються заходи для обмеження її поширення. Це може включати ізоляцію уражених систем, блокування шкідливого трафіку або відключення заражених пристроїв від мережі, що допомагає мінімізувати можливі збитки.

Наступний крок — усунення загрози. На цьому етапі команда безпеки або автоматизовані інструменти видаляють шкідливі файли, виправляють вразливості, закривають недоліки в конфігурації та вживають інші заходи, щоб усунути причини інциденту та зменшити ризик його повторення.

Відновлення — це процес повернення систем до нормальної роботи. Після усунення загрози системи знову підключаються до мережі та перевіряються на наявність залишкових шкідливих компонентів. На цьому етапі можуть бути проведені додаткові тести, щоб переконатися, що загроза повністю усунута.

Завершує процес пост-інцидентний аналіз, де команда безпеки документує всі дії, пов’язані з інцидентом, та робить висновки. Цей аналіз включає

підготовку звітів, рекомендацій для покращення системи безпеки та оновлення сценаріїв реагування. Навчання персоналу на основі цих уроків допомагає покращити готовність організації до майбутніх загроз.

Ручна обробка інцидентів займає значний час, що може дозволити загрозі поширитися глибше в системі. Крім того, людський фактор може призвести до помилок, особливо в умовах високого стресу або перевантаження інформацією.

У відповідь на ці виклики, сучасні організації все частіше звертаються до автоматизації процесів реагування на інциденти. Використання систем оркестрації, автоматизації та реагування на безпеку (SOAR) дозволяє автоматизувати рутинні завдання та швидко реагувати на загрози. SOAR-системи можуть автоматично виявляти та класифікувати інциденти на основі попередньо визначених правил та алгоритмів, а також виконувати необхідні дії без втручання людини. Наприклад, при виявленні підозрілої активності система може автоматично блокувати IP-адресу зловмисника або ізолювати уражений пристрій від мережі. Практичний приклад ефективності SOAR можна побачити у випадку однієї фінансової установи, яка зіткнулася з масованою фішинговою атакою. Завдяки впровадженню SOAR-системи, банк автоматично ідентифікував підозрілі повідомлення та запобіг їхньому відкриттю співробітниками, що суттєво зменшило час реакції та уникнуло потенційних втрат.

Плейбуки та ранбуки є важливими інструментами в управлінні реагуванням на інциденти [7]. Плейбук (playbook) — це набір стандартних процедур та кроків, які команда безпеки повинна виконати при виникненні певного інциденту. Він містить детальні інструкції щодо виявлення, аналізу та реагування на конкретні типи загроз. Ранбук (runbook) — це більш технічний документ, який описує покрокові дії для виконання специфічних завдань або процесів, часто використовується системними адміністраторами для рутинних операцій. Використання SOAR дозволяє автоматизувати ці плейбуки та ранбуки, що прискорює обробку сповіщень та зменшує час реакції на інциденти. SOAR-системи можуть автоматично запускати відповідні плейбуки при виявленні певних подій, виконуючи необхідні дії без участі людини, що підвищує

ефективність та точність реагування [5]. Наприклад, велика технологічна компанія виявила шкідливе програмне забезпечення на одному з комп'ютерів у мережі. SOAR-система автоматично ізолювала уражений пристрій, запустила процес видалення шкідливого ПЗ та відновила системні конфігурації, що дозволило мінімізувати вплив інциденту та запобігти його поширенню.

Проактивні підходи до безпеки також набувають все більшого значення. Вони включають регулярне проведення аудиту безпеки та оцінок вразливостей, що дозволяє виявляти та усувати слабкі місця в системах до того, як ними скористаються зловмисники. Використання методологій Red Teaming та Blue Teaming допомагає організаціям оцінити свою готовність до реальних атак [9]. Обмін інформацією про загрози (Threat Intelligence) між організаціями та використання зовнішніх джерел даних допомагає бути в курсі останніх тенденцій та методів атак, що підвищує готовність до потенційних інцидентів. Використання платформ Threat Intelligence дозволяє автоматично оновлювати захисні механізми на основі нових відомостей про загрози [9].

Реактивні методи реагування залишаються важливими, особливо у випадках складних або невідомих загроз. Інцидент-менеджмент включає детальне розслідування інцидентів, збір доказів для подальшого аналізу, а також відновлення систем до нормальної роботи. Важливо мати чіткий план реагування на інциденти, який визначає ролі та відповідальності команди, процедури повідомлення та комунікації, а також кроки для відновлення.

Пост-інцидентний аналіз дозволяє організаціям навчатися з минулих подій, визначати причини інцидентів та вносити зміни до політик та процедур для покращення кібербезпеки.

Комбіновані підходи, що поєднують різні методи та технології, забезпечують глибокий та багаторівневий захист. Стратегії глибокого захисту (Defense in Depth) передбачають використання кількох шарів безпеки, таких як мережеві фільтри, антивірусні програми, системи виявлення та запобігання вторгнень (IDS/IPS), а також засоби контролю доступу та шифрування даних. Адаптивні системи безпеки можуть динамічно змінювати свої параметри в

реальному часі, реагуючи на зміни в середовищі та характері загроз. Наприклад, медична установа, яка стала ціллю атаки типу "вимагач", змогла завдяки автоматизованим системам швидко ізолювати уражені сервери, відновити дані з резервних копій та запобігти поширенню шкідливого ПЗ на інші системи. Це забезпечило безперервність надання медичних послуг та захист конфіденційної інформації пацієнтів.

Впровадження автоматизованих систем та комбінованих підходів до реагування на інциденти дозволяє значно знизити ризики, підвищити швидкість реагування та забезпечити стійкість до сучасних кіберзагроз. Такий підхід створює надійний фундамент для забезпечення кібербезпеки організацій у динамічному цифровому середовищі.

1.3 Огляд систем автоматизації реагування на інциденти

Системи SOAR (Security Orchestration, Automation, and Response) призначені для автоматизації, оркестрації та оптимізації процесів реагування на інциденти у сфері кібербезпеки. Ці рішення виконують автоматизацію рутинних завдань, аналізують великі обсяги даних та інтегрують різні інструменти, що дозволяє оперативно виявляти й нейтралізувати загрози. Головною метою SOAR-систем є зменшення часу реагування на інциденти, підвищення точності обробки та зниження навантаження на фахівців з безпеки, щоб вони могли зосередитись на вирішенні більш критичних завдань.

Серед ключових особливостей SOAR-систем, що виділяють їх на фоні традиційних рішень безпеки, є автоматизація, оркестрація та інтеграція з різними системами. Автоматизація дозволяє системі виконувати стандартні процеси обробки інцидентів, зокрема, збирати та аналізувати дані з різних джерел, приймати попередньо визначені рішення або запускати процес блокування загрози, що значно скорочує час на виконання рутинних завдань [4]. Оркестрація, у свою чергу, забезпечує злагоджену роботу різних інструментів та систем безпеки, включаючи SIEM, антивіруси та фаєрволи. Це сприяє

ефективній обробці інцидентів і зменшує ризик помилок через людський фактор, полегшуючи координацію між командами та підвищуючи загальну ефективність управління безпекою. Інтеграція з різними системами дозволяє SOAR об’єднуватися з безліччю сторонніх інструментів і сервісів, таких як інструменти Threat Intelligence, платформи управління вразливостями та системи захисту від DDoS [4]. Це дає можливість збирати й аналізувати дані з різних джерел, що робить реагування на загрози більш точним та повним.

IBM QRadar SOAR є потужною системою, що забезпечує комплексний підхід до автоматизації та оркестрації процесів реагування на кіберінциденти [10]. Його архітектура орієнтована на централізоване управління інцидентами, де основний акцент зроблено на інтеграцію з різноманітними інструментами та джерелами даних для всебічного аналізу загроз.

Основні можливості QRadar SOAR спрямовані на ефективне виявлення загроз за допомогою аналізу подій та застосування інтелектуальних алгоритмів, які дозволяють автоматично класифікувати та оцінювати ризик інцидентів. Система має широкий набір функцій для розслідування інцидентів, надаючи фахівцям з безпеки засоби для управління життєвим циклом інциденту, від його виявлення до остаточного вирішення.

QRadar SOAR інтегрується з іншими продуктами IBM, такими як QRadar SIEM, що дозволяє використовувати об’єднані дані та підвищувати ефективність моніторингу безпеки. Крім того, QRadar SOAR може працювати з зовнішніми інструментами, такими як антивірусні рішення, фаєрволи та платформи Threat Intelligence.

IBM QRadar SOAR вирізняється низкою значних переваг. Серед них — висока ефективність в управлінні інцидентами та здатність автоматично проводити комплексний аналіз загроз у реальному часі, що дозволяє суттєво знизити час реагування на інциденти. Його інтеграція з іншими продуктами IBM, такими як QRadar SIEM, а також з різноманітними зовнішніми інструментами, сприяє створенню єдиної інформаційної платформи безпеки.

Однак QRadar SOAR також має свої недоліки. Для ефективного налаштування та використання система вимагає значних технічних знань і ресурсів, що може ускладнити її впровадження в організаціях із обмеженими кадрами або бюджетом. Крім того, налаштування інтеграцій із зовнішніми системами безпеки може потребувати додаткових часових і фінансових витрат, особливо якщо організація використовує специфічні або нестандартні інструменти безпеки.

Splunk SOAR — це система оркестрації, автоматизації та реагування на інциденти кібербезпеки, що пропонує комплексний підхід до автоматизації управління загрозами [11]. Цей інструмент дозволяє організаціям автоматизувати рутинні процеси, знижуючи навантаження на аналітиків та забезпечуючи швидке реагування на потенційні загрози. Splunk SOAR інтегрується з широким спектром сторонніх інструментів, включаючи антивіруси, SIEM-системи, фаєрволи та сервіси Threat Intelligence, що дозволяє створювати єдиний центр управління загрозами. Крім того, система пропонує інструменти для розробки плейбуків та запуску автоматизованих сценаріїв обробки інцидентів, що забезпечує швидку реакцію на загрози та точне виконання встановлених процедур без участі людини.

Основні переваги Splunk SOAR полягають у його гнучкості, здатності інтегруватися з багатьма іншими системами та підтримці широкого спектру автоматизованих дій. Це робить систему дуже привабливою для великих організацій, де необхідно обробляти значний обсяг інцидентів. Платформа також дозволяє кастомізувати плейбуки для конкретних вимог бізнесу, що підвищує адаптивність та зручність у використанні.

Серед недоліків Splunk SOAR слід зазначити його високу вартість та складність у налаштуванні, що може бути проблемою для менших компаній або організацій з обмеженими ресурсами. Крім того, для повноцінного використання всіх функцій потрібні технічні знання та досвід, оскільки система досить складна у впровадженні й потребує ретельного налаштування інтеграцій.

Cortex XSOAR - це потужна платформа автоматизації, оркестрації та реагування на кіберінциденти, розроблена Palo Alto Networks [12]. Cortex XSOAR надає розширені можливості для автоматизації кібербезпеки, орієнтуючись на швидку і скоординовану реакцію на загрози за допомогою плейбуків та ранбуків, які визначають покрокові процедури обробки інцидентів. Перевагою Cortex XSOAR є її гнучкість і масштабованість. Система підтримує широкий спектр інтеграцій з інструментами безпеки, дозволяючи використовувати різні сторонні рішення в єдиному інтерфейсі. Вона також пропонує інструменти для командної співпраці, що сприяє швидкому обміну інформацією та координації дій, а завдяки гнучким плейбукам,

Серед недоліків Cortex XSOAR можна відзначити високі вимоги до ресурсів і технічної підтримки. Впровадження системи потребує часу та значних зусиль для налаштування інтеграцій та плейбуків, а для її повноцінного використання потрібні досвідчені спеціалісти.

Попри численні переваги, існуючі SOAR-системи, такі як IBM QRadar SOAR, Splunk SOAR та Cortex XSOAR, мають низку обмежень, які можуть ускладнювати їх впровадження в багатьох організаціях, особливо з обмеженими ресурсами. Перш за все, усі ці рішення вимагають значних технічних знань та ресурсів для налаштування та підтримки, що може бути непосильним для малих і середніх компаній. Висока вартість ліцензій та підтримки є додатковою перепоною, особливо для компаній з обмеженим бюджетом.

Крім того, складність інтеграції з нестандартними інструментами або специфічними рішеннями для забезпечення безпеки може призводити до затримок у налаштуванні системи та зменшення ефективності роботи. Необхідність у постійній технічній підтримці для забезпечення коректної роботи інтеграцій та плейбуків є значним навантаженням на ресурси організацій.

Результати порівняння підкреслюють, що на ринку SOAR-рішень все ще залишається попит на більш універсальні платформи, здатні адаптуватися до потреб організацій з різними бюджетами та ресурсами. Хоча кожна з платформ має свої сильні сторони, високі витрати на впровадження та технічну підтримку

є суттєвою перешкодою для багатьох компаній. З огляду на це, майбутні рішення SOAR повинні враховувати спрощення процесів налаштування та інтеграції, забезпечуючи при цьому гнучкість і надійність, щоб забезпечити доступ до потужних інструментів кібербезпеки для ширшого кола користувачів і зменшити бар'єри для швидкого реагування на інциденти.

1.4 Формалізація вимог та постановка задачі

Проведений аналіз сучасних кіберзагроз та існуючих методів реагування на інциденти безпеки свідчить про необхідність удосконалення підходів до автоматизації процесів реагування. Огляд наявних SOAR-систем, таких як IBM QRadar SOAR, Splunk SOAR та Cortex XSOAR, виявив низку обмежень, що ускладнюють їх впровадження в організаціях з обмеженими ресурсами. До основних проблем належать:

- Висока вартість ліцензій та підтримки, що робить ці рішення недоступними для малих та середніх підприємств.
- Складність налаштування та інтеграції, яка вимагає значних технічних знань та ресурсів.
- Відсутність гнучкості при адаптації до специфічних потреб організації або інтеграції з нестандартними інструментами безпеки.
- Враховуючи ці обмеження, виникає потреба у розробці більш доступної та адаптивної SOAR-системи, яка б відповідала потребам організацій різного масштабу та рівня технічної компетенції.

Метою даної роботи є підвищення ефективності процесів реагування на інциденти безпеки шляхом розробки адаптивної SOAR-системи, яка забезпечить автоматизацію та оркестрацію реагування на загрози, будучи при цьому доступною для організацій з обмеженими ресурсами.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- Спроєктувати модульну архітектуру SOAR-системи, яка дозволить гнучко налаштовувати систему.

- Визначити інтерфейси та протоколи для інтеграції з зовнішніми системами та сервісами.
- Створити набір плейбуків, які можуть бути легко адаптовані під специфічні потреби організації.
- Реалізувати основні компоненти системи на базі відкритих технологій та фреймворків.
- Забезпечити можливість кастомізації та розширення функціональності.
- Провести тестування системи в реальних або наближених до реальних умовах.
- Оцінити ефективність автоматизації процесів реагування та вплив на швидкість та якість обробки інцидентів.

Основні вимоги до розроблюваної SOAR-системи:

- Низькі вимоги до апаратних ресурсів та можливість розгортання на існуючій інфраструктурі.
- Інтуїтивно зрозумілий інтерфейс користувача для налаштування та управління системою.
- Модульна архітектура, що дозволяє додавати або видаляти функціональні компоненти за потребою.
- Підтримка інтеграції з різними інструментами та сервісами через стандартизовані API.
- Захист самої системи від потенційних кібератак, включаючи механізми аутентифікації та авторизації.
- Швидке виявлення та реагування на інциденти в режимі реального часу.
- Зниження навантаження на команди безпеки шляхом автоматизації рутинних завдань.

Відповідно до поставлених задач та вимог, у подальших розділах буде детально розглянуто процес розробки адаптивної SOAR-системи, описано її архітектуру, функціональні можливості та результати тестування в практичних умовах.

2 РОЗРОБКА АРХІТЕКТУРНИХ РІШЕНЬ

2.1 Архітектура системи

Інструмент автоматизованої обробки сповіщень систем безпеки складається з кількох основних компонентів, кожен з яких виконує певну роль у процесі автоматизованого реагування.

Інструмент автоматизованої обробки сповіщень у системах безпеки складається з кількох основних компонентів, кожен з яких відіграє певну роль у процесі автоматизованого реагування та забезпечення кібербезпеки.

Загалом структуру самого рішення можна представити у вигляді теоретико- множинної моделі, загальна структурна схема схему наведено на рисунку 2.1 :

$$\mathrm{SOAR} = \{\mathrm{Src}, \mathrm{IngtLayer}, \mathrm{AutoLayer}, \mathrm{PlaybookLayer}, \mathrm{OrchLayer}, \mathrm{MonLayer}, \mathrm{PostIncLayer}\}$$

де:

- Src - шар «Джерела даних» - збирає та передає дані про потенційні загрози;
- IngtLayer - інтеграційний шар – перетворює дані у стандартизований формат, придатний для подальшої обробки;
- AutoLayer - шар автоматизації – відповідає за роботу автоматизованих сценаріїв;
- PlaybookLayer – шар плейбуків – слугує місцем зберігання сценаріїв реагування для різних типів загроз;
- OrchLayer – оркестраційний шар – забезпечує узгоджене виконання дій плейбуків на рівні реальних систем
- MonLayer – шар моніторингу – займається відслідковуванням статусу виконання задач.
- PostIncLayer – шар пост-інцидентного аналізу – займається

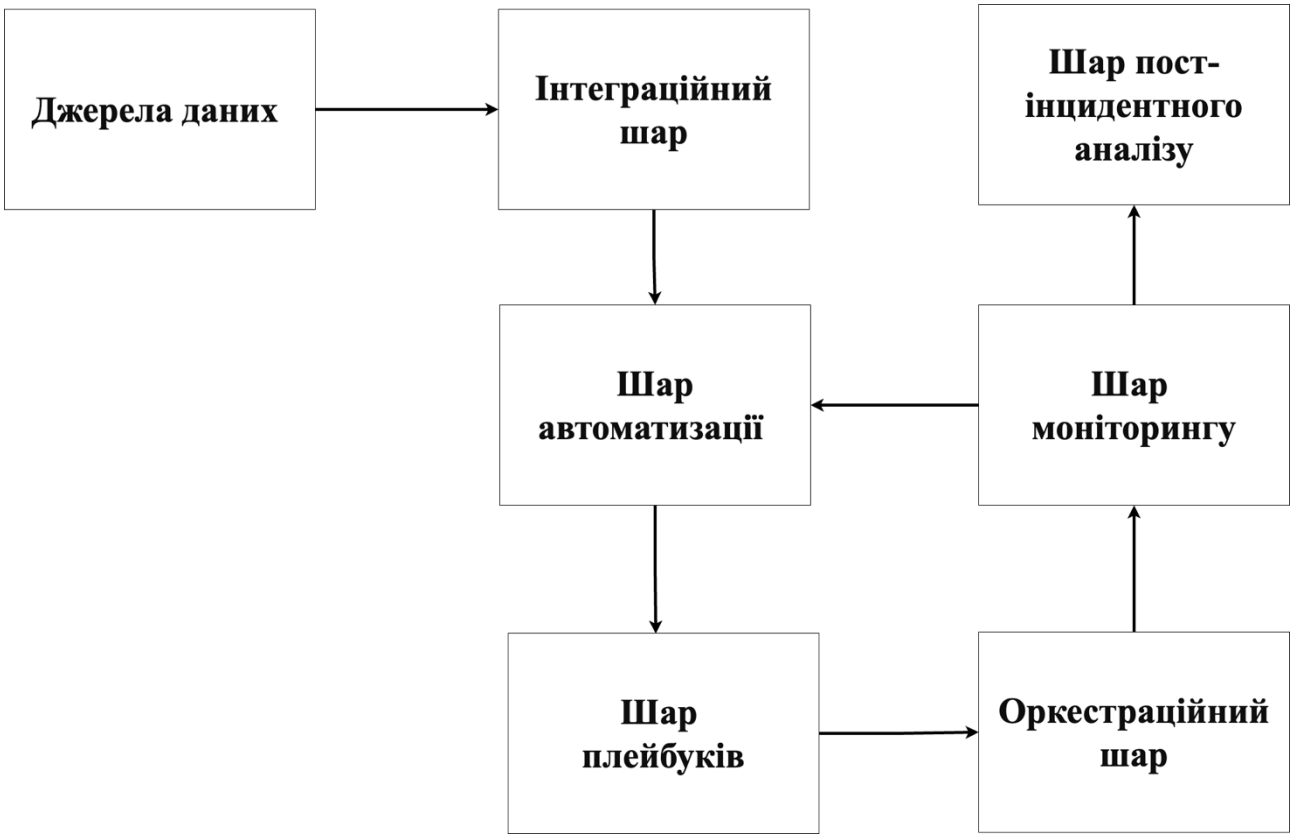


Рисунок 2.1 – Загальна структурна схема системи автоматизації

Шар «Джерела даних» включає системи збору інформації про загрози, такі як SIEM, антивірусні програми, фаєрволи та служби Threat Intelligence. Цей шар збирає та передає дані про потенційні загрози для подальшого аналізу, створюючи основу для реагування на інциденти.

Інтеграційний шар уніфікує вхідні дані, перетворюючи їх у стандартизований формат, придатний для обробки в системі. Тут дані нормалізуються та фільтруються, що дозволяє зосередити ресурси на критичних інцидентах, зменшуючи вплив незначних подій на загальну ефективність реагування.

Шар автоматизації є центральним елементом системи, який обирає відповідні сценарії реагування, виходячи з аналізу отриманих даних. Завдяки налаштованим плейбукам цей шар ініціює автоматичні дії для нейтралізації загроз і зменшення часу реагування.

Шар плейбуків слугує місцем зберігання сценаріїв реагування для різних типів загроз. Кожен плейбук описує послідовність дій для вирішення певного

інциденту, автоматизуючи рутинні процеси та дозволяючи аналітикам зосередитися на складніших задачах.

Оркестраційний шар забезпечує узгоджене виконання дій плейбуків на рівні реальних систем, інтегруючи SOAR з іншими інструментами безпеки. Цей компонент координує всі дії, необхідні для усунення загрози, забезпечуючи цілісність та ефективність виконання заходів реагування.

Шар моніторингу відстежує виконання дій у реальному часі та виявляє нові аномалії, що можуть вказувати на нові загрози або проблеми під час реагування. Він слугує механізмом зворотного зв'язку, що дозволяє оперативно вносити корективи.

Шар пост-інцидентного аналізу завершує процес обробки, оцінюючи ефективність реагування, генеруючи звіти та вдосконалюючи плейбуки на основі отриманого досвіду. Це дозволяє постійно підвищувати якість реагування на майбутні інциденти.

Кожен з цих компонентів тісно взаємодіє з іншими, створюючи інтегровану структуру SOAR-системи, яка забезпечує швидке та узгоджене реагування на інциденти в режимі реального часу.

Шар «Джерела даних» отримує інформацію з EDR, NDR, Threat Intelligence та SIEM, дані зв'язки можна представити в вигляді теоретико множинної моделі та додатково схему наведено на рисунку 2.2:

$$Src = \{EDR, NDR, SIEM, TI\},$$

де:

- EDR - Endpoint detection and response - антивірусний засіб;
- NDR - Network Detection And Response - засіб для виявлення загроз в мережі;
- SIEM - Security information and event management - засіб для централізованого збору та управління подіями безпеки;
- TI - Threat intelligence - сервіси для збору інформації про загрози з зовнішніх ресурсів;

Рисунок 2.2 – Структура шару «Джерела даних»

EDR сканує кінцеві пристрої, виявляє шкідливі файли та процеси, аналізуючи їх на наявність загроз. У разі виявлення підозрілих об'єктів або процесів система передає дані до SOAR для швидкої реакції, такої як ізоляція пристрою чи видалення шкідливого ПЗ.

NDR та фаєрволи захищають мережу, контролюючи та фільтруючи трафік і блокуючи підозрілі з'єднання. IDS/IPS додатково аналізують трафік і сповіщають SOAR про підозрілі дії або несанкціонований доступ, що дозволяє оперативно приймати рішення.

Threat Intelligence постачає зовнішні дані про загрози (IP-адреси, домени, файли, пов’язані з атаками), збагачуючи SOAR актуальною інформацією про методи атак та вразливості, що дозволяє оцінити ризик і пріоритетність кожного інциденту.

Інтеграційний шар відповідає за обробку та підготовку даних для SOAR-системи. Структуру даного шару наведено нижче в вигляді теоретико множинної моделі, також наведено структурну схему даного шару на рисунку 2.3.

$$IntLayer = \{DC, Normalize, Filter, Corre, Rout\},$$

де:

– DC – Data Collection Module - приймає інформацію від усіх джерел, об'єднуючи її;
– Normalize – Data Normalization Module – нормалізує інформацію під один формат;

– Filter – Data Filtering Module – фільтрує інформацію відповідно до її серйозності;
– Corre – Data Correlation Module – об'єднує пов’язані події в єдиний інцидент;
– Rout – Data Routing Module – оцінює критичність інциденту і передає його далі;

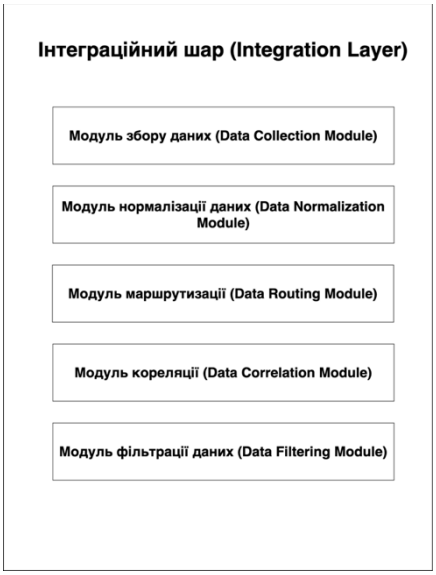


Рисунок 2.3 – Структурна схема інтеграційного шару

Прикладом роботи даного шару може служити спроба несанкціонованого входу з підозрілої IP-адреси до корпоративної мережі. Ця подія фіксується системами SIEM та EDR, і додаткова інформація про IP-адресу надходить із зовнішнього джерела Threat Intelligence, яке повідомляє про її зв’язок із відомою ботнет-мережею.

На першому етапі модуль збору даних (Data Collection Module) приймає цю інформацію від усіх джерел, об'єднуючи її в єдиний вхідний потік для SOAR-системи. Використовуючи API та конектори, модуль збирає параметри інциденту, такі як IP-адреса, час події, тип загрози, та уніфікує ці дані, щоб стандартизувати формат для подальшої обробки.

Далі, модуль нормалізації даних (Data Normalization Module) стандартизує ці дані. Наприклад, час події, зафіксований у SIEM та Threat Intelligence у різних

часових поясах, перетворюється до єдиного стандарту UTC. Також IP-адреса і дані про ботнет структуруються в узгоджений формат, що спрощує їхнє використання на наступних етапах.

Після нормалізації інцидент проходить через модуль фільтрації даних (Data Filtering Module), який аналізує важливість події. Оскільки інформація свідчить про зв’язок IP-адреси з ботнет-мережею, цей інцидент вважається високопріоритетним. Натомість інші незначні події (наприклад, загальні попередження системи про низький рівень загрози) можуть бути відсіяні, залишаючи в обробці лише суттєві дані.

Далі інцидент обробляється модулем кореляції (Data Correlation Module), який об'єднує пов’язані події в єдиний інцидент. Наприклад, спроби входу з однієї й тієї ж IP-адреси, що повторювалися кілька разів, та отримані попередження від Threat Intelligence про цю IP-адресу формують один цілісний інцидент. Ця кореляція дозволяє системі визначити, що всі ці події є частиною комплексної загрози, пов’язаної з ботнет-активністю.

На завершення, модуль маршрутизації (Data Routing Module) оцінює критичність інциденту, який отримав високий пріоритет через зв’язок із ботнетом, і передає його до модуля автоматизації в SOAR для негайного реагування.

Шар автоматизації забезпечує управління та виконання сценаріїв реагування на інциденти в SOAR-системі. Структуру даного шару наведено нижче в вигляді теоретико множинної моделі, також наведено структурну схему даного шару на рисунку 2.4.

$$AutoLayer = \{IncAnalyzer, PBPicker, PBxecuter, RuleEdditor, Loger\},$$

де:

- IncAnalyzer – Incident Analysis Module – отримує інформацію про інцидент і оцінює його харакетиристики;
- PBPicker – Playbook Selection Module – обирає сценарій реагування відповідно до налаштувань;
- PBxecuter – Playbook Execution Module – передає інформацію ;

– RuleEdittor – Rule Management Module – містить правила для застосування обраного плейбука;
– Loger – Data Routing Module – логує події;

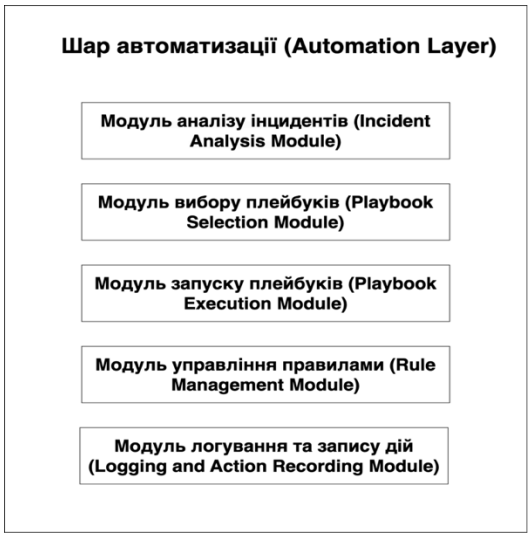


Рисунок 2.4 – Структурна схема інтеграційного шару

Прикладом роботи даного шару може бути реагування на спробу несанкціонованого входу з IP-адреси, яка була визначена як загроза ботнету.

На першому етапі модуль аналізу інцидентів Incident Analysis Module отримує інформацію про інцидент з інтеграційного шару, оцінюючи характеристики загрози, такі як тип інциденту, рівень пріоритетності та потенційний вплив на систему. Згідно з попередньо встановленими правилами, цей інцидент визначається як критичний, що потребує негайного реагування.

Після початкового аналізу, модуль вибору плейбуків підбирає відповідний сценарій реагування для даного інциденту. Наприклад, для спроби входу з підозрілої IP-адреси, пов'язаної з ботнетом, обирається плейбук, який включає інструкції для блокування IP-адреси у фаєрволі та інформування команди безпеки про спробу проникнення.

Далі, модуль управління правилами дозволяє налаштувати правила для застосування обраного плейбука.

Обраний сценарій передається в модуль запуску плейбуків (Playbook Execution Module), який виконує всі команди у визначеній послідовності. У випадку з ботнет-інцидентом, модуль ініціює блокування IP-адреси у фаєрволі, ізолює заражений пристрій (якщо він був ідентифікований), і надсилає повідомлення команді безпеки.

Далі модуль логування та запису дій (Logging and Action Recording Module) зберігає всі виконані дії в детальному журналі. У випадку даного інциденту зберігається запис про кожен етап реагування — від аналізу інциденту до блокування IP-адреси та сповіщення команди безпеки.

Шар плейбуків у SOAR-системі відповідає за управління сценаріями реагування, які автоматизують дії під час інцидентів. Структуру даного шару наведено нижче в вигляді теоретико множинної моделі, також наведено структурну схему даного шару на рисунку 2.5.

$$PlaybookLayer = \{PBLibr, PBEditor, CondExec, PBIntLayer, Loger\},$$

де:

- PBLibr – Playbook Library – централізоване сховище усіх сценаріїв реагування;
- PBEditor – Playbook Editor – модуль для створення та редагування сценаріїв;
- CondExec – Conditional Execution Module – модуль умов виконання;
- PBIntLayer – Playbook Integration Layer – забезпечує взаємодію плейбуків із компонентами інтеграційного шару;
- Loger – Execution Logging module – логує події;

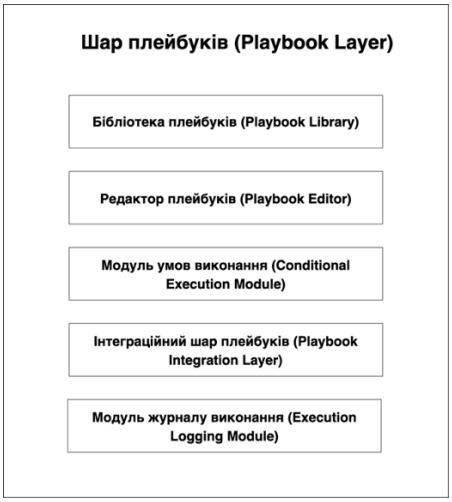


Рисунок 2.5 – Структурна схема інтеграційного шару

Бібліотека плейбуків (Playbook Library) слугує централізованим сховищем усіх сценаріїв реагування, які використовуються в системі. У цій бібліотеці зберігаються плейбуки для різних типів загроз, таких як фішингові атаки, DDoS-атаки та витоки даних. Кожен плейбук містить чітко визначену послідовність дій для усунення конкретної загрози, що дозволяє швидко отримати доступ до відповідного сценарію на основі характеристик інциденту.

Редактор плейбуків (Playbook Editor) дозволяє адміністраторам створювати, змінювати та налаштовувати плейбуки відповідно до потреб організації. За допомогою редактора можна визначати конкретні дії та умови їх виконання, а також оновлювати сценарії відповідно до змін у ландшафті загроз або появи нових методів захисту.

Модуль умов виконання (Conditional Execution Module) додає гнучкості під час виконання плейбуків, реагуючи на результати попередніх дій.

Інтеграційний модуль плейбуків (Playbook Integration) забезпечує взаємодію плейбуків із компонентами інтеграційного шару SOAR.

Модуль журналу виконання (Execution Logging Module) веде детальний запис усіх дій, виконаних у рамках сценарію реагування. Він фіксує інформацію про час і статус кожного кроку, надаючи аналітичні дані для подальшого аналізу.

Шар оркестрації забезпечує узгоджену роботу всіх компонентів SOAR-системи, координуючи виконання завдань для ефективного реагування на

інциденти. Структуру даного шару наведено нижче в вигляді теоретико множинної моделі, також наведено структурну схему даного шару на рисунку 2.6.

$$OrchLayer = \{TMan, ToolMaster, CoordinationMod, ErrorHandler, Loger\},$$

де:

- TMan – Task Management Module – розподіляє завдання між різними інструментами безпеки;
- ToolMaster – Security Tools Integration Module – забезпечує безпосередню взаємодію SOAR-системи з зовнішніми інструментами;
- CoordinationMod – Conditional Execution Module – координує виконання завдань;
- ErrorHandler– Error Handling Module – відстежує можливі помилки під час виконання завдань;
- Loger – Execution Reporting module – логує події;

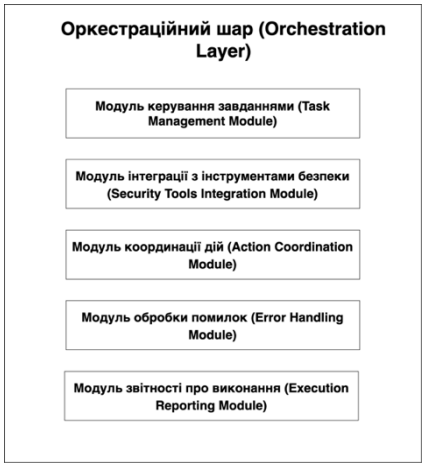


Рисунок 2.6 – Структурна схема оркестраційного шару

У контексті інциденту, пов’язаного з підозрілою IP-адресою, що намагається отримати доступ до корпоративної мережі, кластер оркестрації забезпечує узгоджену роботу всіх компонентів SOAR-системи для ефективного реагування.

На першому етапі модуль керування завданнями (Task Management Module) розподіляє завдання між різними інструментами безпеки, забезпечуючи правильну послідовність виконання дій для реагування на загрозу.

Модуль інтеграції з інструментами безпеки (Security Tools Integration Module) забезпечує безпосередню взаємодію SOAR-системи з такими зовнішніми інструментами, як фаєрволи, IDS/IPS, антивірусні рішення та системи SIEM.

Модуль обробки помилок (Error Handling Module) відстежує можливі помилки під час виконання завдань.

Модуль координації дій (Action Coordination Module) синхронізує виконання всіх дій, забезпечуючи їхню правильну послідовність.

Модуль звітності про виконання (Execution Reporting Module) збирає та зберігає інформацію про статус виконання кожного завдання, включаючи як завершені, так і невдалі дії.

Шар моніторингу забезпечує контроль за кожним етапом реагування, відстежуючи статус і продуктивність виконання завдань у реальному часі. Структуру даного шару наведено нижче в вигляді теоретико множинної моделі, також наведено структурну схему даного шару на рисунку 2.7.

$$MonLayer = \{IncTracker, AlertMod, PerfMod, AnomDetector, Loger\},$$

де:

- IncTracker – Incident Tracking Module – контролює поточний статус інциденту;
- AlertMod – Alert Notification Module – надсилає повідомлення про важливі події під час реагування;
- PerfMod – Performance Monitoring Module – збирає статистичні дані про продуктивність;
- AnomDetector – Anomaly Detection Module – аналізує потоки даних у пошуках незвичних патернів;
- Loger – Event Loging module – логує події;

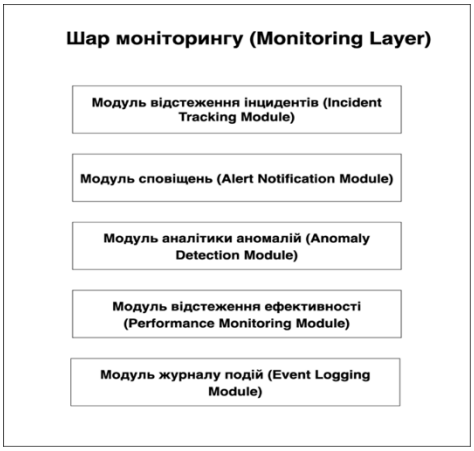


Рисунок 2.7 – Структурна схема моніторингового шару

У контексті інциденту з підозрілою IP-адресою модуль буде працювати таким чином. На першому етапі модуль відстеження інцидентів (Incident Tracking Module) контролює поточний статус інциденту, надаючи команді безпеки актуальну інформацію про його обробку.

У випадку даного інциденту модуль дозволяє команді бачити, на якому етапі знаходиться процес блокування IP-адреси, чи було виконано додаткові перевірки і чи завершено інші дії, пов’язані з інцидентом.

Модуль сповіщень (Alert Notification Module) надсилає повідомлення про важливі події під час реагування. Наприклад, якщо виникає проблема під час спроби заблокувати IP-адресу або з’являється додаткова загроза, пов’язана з цим інцидентом, модуль автоматично сповіщає команду безпеки, дозволяючи їй швидко відреагувати на зміну ситуації.

Модуль відстеження ефективності (Performance Monitoring Module) збирає статистичні дані про продуктивність виконання завдань у процесі обробки інциденту.

Модуль аналітики аномалій (Anomaly Detection Module) аналізує потоки даних у пошуках незвичних патернів, які можуть свідчити про нові.

Модуль журналу подій (Event Logging Module) зберігає повну історію всіх подій, що сталися під час обробки інциденту.

Шар пост-інцидентного аналізу завершує процес обробки інцидентів, надаючи детальний аналіз та рекомендації для вдосконалення системи кібербезпеки. Структуру даного шару можна представити у вигляді теоретико-множинної моделі, а також схематично зобразити на рисунку 2.8:

$$PostIncLayer = \{Report, Effanalysis, PBimprovenetn, Train, Archiv\},$$

де:

- Report – Reporting Module – генерує детальні звіти про кожен інцидент;
- EffAnalysis – Effectiveness Analysis Module – оцінює загальну ефективність дій, здійснених під час інциденту;
- PBmprovement – Performance Monitoring Module – використовує результати аналізу інцидентів для оптимізації сценаріїв реагування;
- Train – Anomaly Detection Module – створює навчальні матеріали для команди безпеки на основі даних, зібраних під час аналізу інцидентів;
- Archiv – Archiving Module – зберігає всю інформацію про інциденти для довгострокового використання та аналізу;

Рисунок 2.8 – Структурна схема шару пост-інцидентного аналізу

Модуль звітності (Reporting Module) генерує детальні звіти про кожен інцидент, охоплюючи всі етапи реагування — від виявлення до завершення. Звіти містять інформацію про час і результат кожної дії, надаючи повну картину розвитку інциденту та заходів, вжитих для його усунення.

Модуль аналізу ефективності (Effectiveness Analysis Module) оцінює загальну ефективність дій, здійснених під час інциденту. Він аналізує швидкість реагування, якість виконаних дій та їх вплив на усунення загрози.

Модуль вдосконалення плейбуків (Playbook Improvement Module) використовує результати аналізу інцидентів для оптимізації сценаріїв реагування. На основі отриманих даних про те, як було усунено інцидент, модуль дозволяє вдосконалювати плейбуки, вносячи корективи для підвищення їх ефективності під час наступних подібних інцидентів.

Модуль навчання (Training Module) створює навчальні матеріали для команди безпеки на основі даних, зібраних під час аналізу інцидентів. Він допомагає співробітникам розвивати навички та знання, необхідні для реагування на подібні загрози у майбутньому.

Модуль архівування (Archiving Module) зберігає всю інформацію про інциденти для довгострокового використання та аналізу. Він створює історію інцидентів, яка може використовуватися для повторного аналізу, відповідності вимогам регуляторних органів чи вивчення тенденцій загроз.

Після успішного реагування на інцидент, пов'язаний із спробою несанкціонованого доступу з підозрілої IP-адреси, шар пост-інцидентного аналізу починає свою роботу.

Модуль звітності генерує детальний звіт про інцидент, включаючи часові мітки кожної дії (виявлення, блокування IP-адреси, сповіщення команди безпеки), результативність цих дій та загальний вплив на систему.

Модуль аналізу ефективності оцінює, наскільки швидко було виявлено загрозу та виконано необхідні дії. Він може виявити, що, наприклад, процес блокування IP-адреси зайняв більше часу, ніж очікувалося, і запропонувати оптимізації.

Модуль вдосконалення плейбуків на основі аналізу може внести корективи в існуючий плейбук. Наприклад, додати автоматичне сповіщення до додаткових членів команди або інтегрувати нові дії для швидшого реагування.

Модуль навчання використовує отримані дані для створення навчальних матеріалів або проведення тренінгів для команди безпеки, фокусуючись на виявлених слабких місцях або нових методах реагування.

Модуль архівування зберігає всю інформацію про інцидент у захищеному сховищі, що дозволяє в майбутньому аналізувати тенденції, відповідати на запити регуляторів або використовувати дані для навчання нових співробітників.

Шар пост-інцидентного аналізу тісно інтегрований з іншими компонентами системи:

- Отримує дані від шару моніторингу (MonLayer) про виконані дії та їх результати.
- Передає рекомендації до шару плейбуків (PlaybookLayer) для оновлення сценаріїв реагування.
- Співпрацює з шаром автоматизації (AutoLayer) для впровадження змін та оптимізацій у процесі реагування.
- Забезпечує команду безпеки аналітичними звітами та навчальними матеріалами для підвищення кваліфікації.

Загалом, структура SOAR-системи побудована таким чином, щоб забезпечити комплексний, автоматизований підхід до реагування на інциденти кібербезпеки, що охоплює всі етапи – від збору й інтеграції даних до моніторингу та пост-інцидентного аналізу.

SOAR-система взаємодіє з іншими системами безпеки, такими як фаєрволи, антивірусні програми, IDS/IPS, передаючи їм команди для виконання певних дій. Наприклад, у випадку виявлення загрози може надіслати команду фаєрволу для блокування підозрілого IP-адресу або EDR для ізоляції зараженого пристрою. Це дозволяє плейбукам виконувати автоматизовані дії з мінімальним втручанням людини.

Джерела загроз передають інформацію про інциденти потрапляє в інтеграційний шар, де відбувається нормалізація, фільтрація та кореляція даних. Цей процес стандартизує та фільтрує дані, залишаючи лише критичні події, що потребують подальшої обробки.

Після фільтрації дані передаються до шару автоматизації, який аналізує інциденти та обирає відповідні плейбуки для реагування. Далі шар автоматизації передає команди для запуску сценаріїв реагування, обираючи оптимальні дії для кожного конкретного інциденту.

Вибраний плейбук активується передаючи послідовність дій до оркестраційного модуля, який керує виконанням цих дій. Оркестраційний шар також використовує зв'язки з іншими інструментами безпеки, гарантуючи, що всі кроки плейбуку виконуються належним чином.

Шар моніторингу отримує інформацію про статус виконання кожної дії та відстежує будь-які аномалії або помилки. У разі виявлення проблем, надсилається сповіщення команді безпеки, що дозволяє оперативно втручатися у процес, якщо автоматизоване реагування не справляється із загрозою.

Основні компоненти SOAR-системи, такі як база даних, модулі автоматизації та інтеграційні шари, повинні мати резервні копії. Це забезпечує відмовостійкість, що дозволяє системі продовжувати роботу навіть при збоях окремих елементів. Наприклад, при виході з ладу одного вузла система автоматично переключається на резервний вузол, що дозволяє уникнути простоїв та забезпечити безперервне реагування на загрози.

Для забезпечення стабільної роботи під час збільшення навантаження використовується горизонтальне масштабування, що передбачає додавання нових серверів або вузлів, які обробляють інциденти паралельно. Вертикальне масштабування також може бути корисним, коли ресурси існуючих серверів збільшуються шляхом додавання пам'яті або процесорної потужності.

З метою підвищення продуктивності SOAR-системи застосовуються такі підходи, як кешування даних, оптимізація запитів до бази даних та використання асинхронної обробки задач. Кешування дозволяє зменшити кількість повторних запитів до бази даних, прискорюючи доступ до часто використовуваних даних.

Безпека SOAR-системи є критично важливою, оскільки система обробляє конфіденційні дані та координує автоматизовані заходи з кібербезпеки. Для

цього використовуються механізми контролю доступу, шифрування даних та інструменти захисту від загроз.

У системі реалізовано механізми багаторівневого контролю доступу, що включають аутентифікацію та авторизацію. Для доступу до різних модулів системи застосовуються ролі та привілеї, що обмежують доступ тільки до авторизованих користувачів.

Для захисту даних, які передаються між компонентами системи та зовнішніми інструментами безпеки, використовується шифрування як на рівні передачі, так і на рівні зберігання. Протоколи шифрування, такі як TLS (Transport Layer Security), забезпечують захищену передачу даних, запобігаючи їх перехопленню сторонніми особами.

Крім того, конфіденційні дані в базах даних шифруються, що забезпечує їх безпеку навіть у разі компрометації серверів.

2.2 Розробка алгоритмів роботи системи

Система SOAR отримує вхідні сповіщення з різних джерел безпеки, таких як SIEM, IDS/IPS чи антивірусні програми. Кожне сповіщення аналізується за ключовими параметрами, включаючи тип загрози, рівень критичності та джерело.

Алгоритм обробки починається з підрахунку кількості схожих сповіщень, які надходять за певний проміжок часу. Якщо система виявляє, наприклад, 10 сповіщень, що мають спільні параметри, вони автоматично об’єднуються в один інцидент. Відповідно, замість розгляду окремих подій система працює з єдиним інцидентом, що представляє собою сукупність пов’язаних подій.

Як тільки інцидент сформовано, він передається на обробку згідно з відповідним сценарієм реагування (плейбуком). У цьому процесі система автоматично виконує необхідні дії, такі як блокування джерела атаки, повідомлення аналітиків чи ізоляція уражених пристроїв. Якщо інцидент

стосується кількох систем чи служб, плейбук може включати паралельні чи послідовні дії для забезпечення максимально ефективної реакції.

У разі виникнення помилок під час обробки система автоматично генерує сповіщення та намагається повторити дію. Якщо кількість спроб досягає встановленого ліміту, команда безпеки отримує повідомлення про необхідність втручання.

Передача даних між шарами реалізована за допомогою REST API для забезпечення синхронної та стандартизованої взаємодії між компонентами системи. REST API використовується для ініціалізації критичних запитів, таких як створення, оновлення та вилучення інцидентів у модулі моніторингу та аналітики.

Асинхронний обмін даними між модулями досягається за допомогою черг, які організовують передачу даних у фоні, незалежно від основних процесів. Використання черг сприяє зниженню навантаження на модулі, оскільки кожен модуль може обробляти запити та реагувати на інциденти в зручний для нього час, забезпечуючи високу продуктивність системи навіть при збільшенні навантаження. Схему роботи черг наведено на рисунку 2.9.

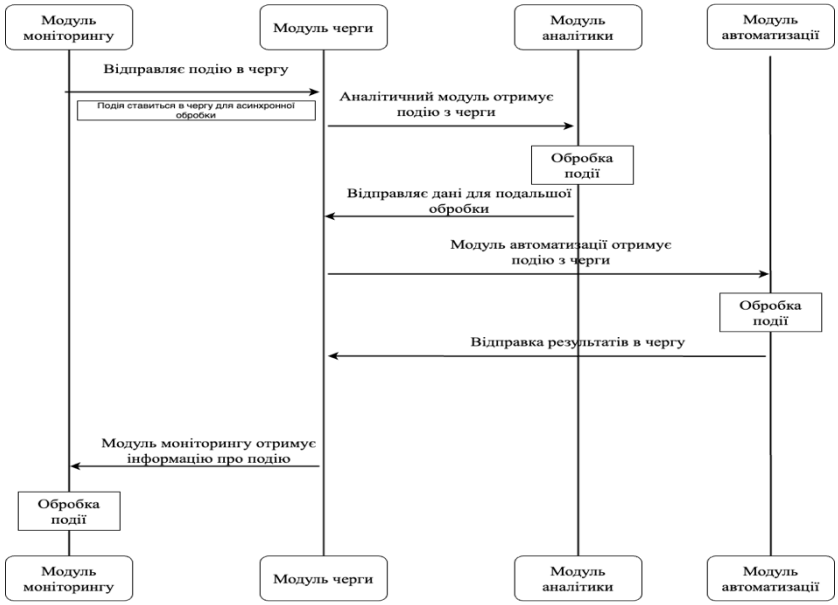


Рисунок 2.9 – Механізм роботи черг

Для безпеки міжмодульної взаємодії використовується механізм автентифікації та авторизації, який базується на токенах. При кожному запиті, REST API або вебсокети перевіряють наявність дійсного токена для дозволу доступу до захищених ресурсів, схему наведено на рисунку 2.10.

Токени надають кожному модулю право обробки лише тих запитів, які йому дозволені, забезпечуючи розмежування прав доступу до різних функцій системи.

Також реалізований захист від базових атак на веб-додатки, таких як ін'єкції SQL, NoSQL і XSS, а також CSRF-атак, що запобігає несанкціонованому доступу або маніпуляціям з боку зловмисників. Для захисту від ін'єкційних атак система фільтрує та перевіряє введені дані, не дозволяючи виконувати небезпечні запити.

Рисунок 2.10 – Схема роботи авторизації та аутентифікації

Окрім цього, система має засоби запобігання атакам типу brute force, шляхом обмеження кількості запитів від одного джерела та тимчасового блокування IP-адреси при виявленні підозрілої активності.

Засоби захисту даних забезпечують цілісність, конфіденційність та доступність інформації в системі, що особливо важливо в умовах кіберзагроз. Для захисту даних у процесі передачі та зберігання використовуються сучасні криптографічні протоколи, які гарантують високий рівень безпеки, роботу даних засобів в інфраструктурі SOAR системи наведено на рисунку 2.11.

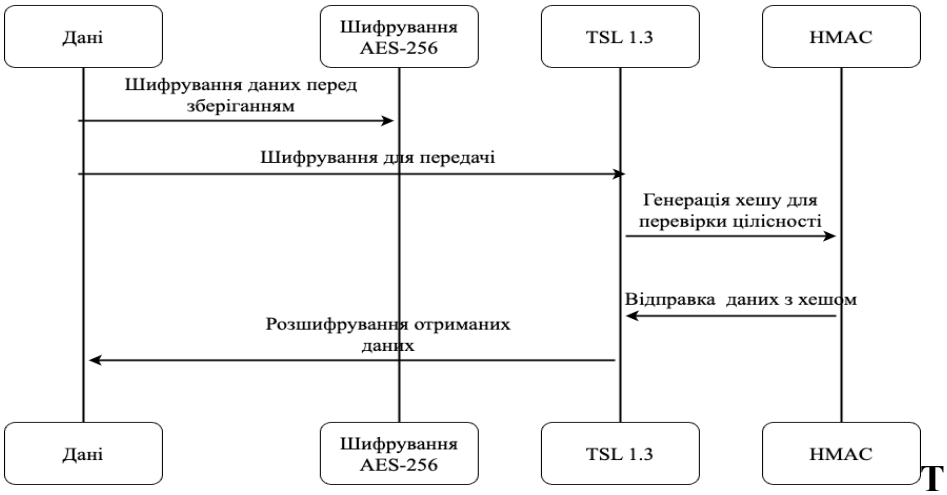


Рисунок 2.11 – Архітектура безпеки даних

Шифрування AES-256 застосовується для зберігання даних, оскільки він забезпечує надійний захист завдяки своїй стійкості до атак. Цей алгоритм створює шифр, що дозволяє зберігати конфіденційні дані в захищеному вигляді, знижуючи ризики несанкціонованого доступу. У процесі передачі даних між модулями використовується протокол TLS 1.3, який забезпечує шифрування в реальному часі, захищаючи інформацію від перехоплення та модифікацій під час передачі через мережу. TLS 1.3 є вдосконаленим протоколом передачі, який не тільки шифрує інформацію, але і захищає її від загроз, пов’язаних із можливими атаками посередника.

Для перевірки цілісності повідомлень у міжмодульній комунікації використовується HMAC (Хеш-код повідомлення із ключем). Використання HMAC дозволяє перевіряти, чи не було дані змінені під час передачі, оскільки хеш генерується на основі секретного ключа та переданого повідомлення. Це дозволяє модулям системи перевіряти, чи отримане повідомлення є

автентичним, і чи не було змінено його зміст. Додатково, всі резервні копії даних шифруються за допомогою AES-256, щоб запобігти можливим ризикам під час зберігання резервних даних.

Інтеграція з іншими системами є критичною складовою, яка дозволяє системі обмінюватися інформацією з зовнішніми сервісами, такими як SIEM (система управління інформацією та подіями безпеки), EDR (інструмент захисту кінцевих точок), та фаєрволами, схему звязків наведено на рисунку 2.13.

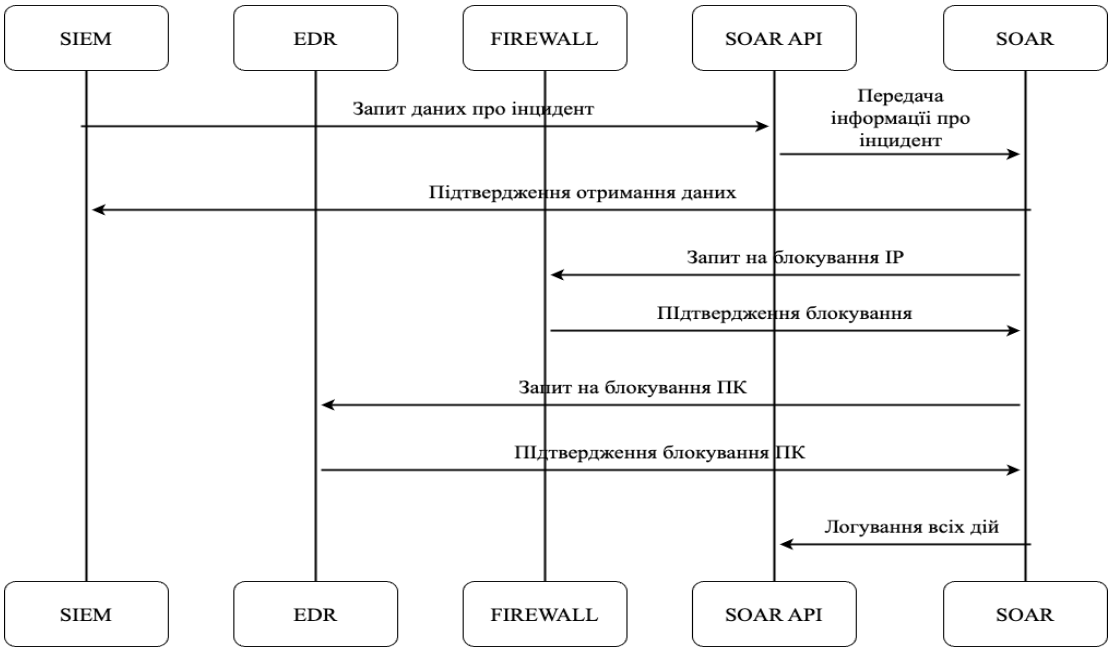


Рисунок 2.13 – Зв'язок з системами безпеки

Для інтеграції з цими системами використовується архітектура, яка забезпечує уніфікований підхід до обміну даними. Зв'язки з системами безпеки дозволяють взаємодіяти з різними інструментами кібербезпеки, надсилаючи запити для отримання даних про інциденти або відправки команд на виконання дій у відповідь на загрози. Це включає можливість ініціювати блокування IP-адреси, ізоляцію зараженого пристрою або збір даних з інших систем для подальшого аналізу. Безпека передачі даних реалізується за допомогою протоколу TLS, який забезпечує захищене з’єднання. Також для автентифікації запитів використовується токени, які дозволяють ідентифікувати, що запит надходить від авторизованої системи.

Обмін даними з зовнішніми системами відбувається в уніфікованих форматах, таких як JSON або XML, що забезпечує сумісність з більшістю сучасних кібербезпекових рішень.

Для ефективної взаємодії з іншими системами впроваджено механізми зворотного зв’язку, що дозволяють отримувати підтвердження успішного виконання команд або змін статусу завдань у сторонніх системах. Наприклад, якщо команда на блокування IP успішно виконана, система отримує підтвердження цього від фаєрволу. Це дозволяє перевіряти статуси всіх ініційованих дій і оперативно вносити зміни в план обробки інциденту.

Для автоматизації процесів було представлено модель використання сценаріїв реагування (плейбуки). Детальна схема структури плейбука наведена на рисунку 2.14.

Рисунок 2.14 – Структурна схема плейбуку

У верхній частині схеми розташовано ідентифікаційну інформацію, яка включає унікальний ідентифікатор та назву плейбука. Ці дані дозволяють однозначно визначити плейбук серед інших і надати користувачам зрозуміле представлення про його призначення. Нижче наведені метадані, що містять інформацію про дату створення та останнього оновлення, забезпечуючи відстеження актуальності плейбука та внесених у нього змін.

Центральним елементом є набір дій, кожна з яких має свої параметри, тип, умови виконання та порядок. Дії організовані в логічній послідовності, що

дозволяє забезпечити зв’язність та взаємозалежність процесів. Виконання дій супроводжується переходами, включаючи умовні варіанти, що враховують результати попередніх дій, а також механізми обробки помилок для підвищення стійкості системи.

Важливу роль відіграє контекст виконання, який об’єднує всі компоненти плейбука. Він включає початкові дані, які передаються на вхід системи, а також дані, що оновлюються після виконання кожної дії.

Таким чином, представлена структура є універсальною основою для побудови автоматизованих процесів, забезпечуючи інтеграцію різних дій та компонентів у єдину ефективну систему.

Створення плейбуків включає кілька ключових етапів, що забезпечують адекватне реагування на нові та змінні загрози. Розробка нових плейбуків починається з ідентифікації актуальних загроз, які можуть вплинути на організацію, процес створення плейбуків наведено на рисунку 2.15.

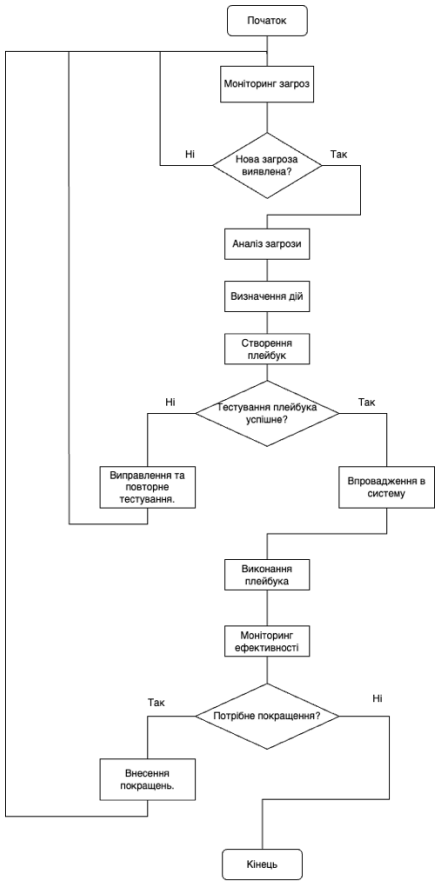


Рисунок 2.15 – Життєвий цикл процесу розробки плейбука

Команда кібербезпеки постійно моніторить ландшафт загроз, аналізуючи інформацію з різних джерел, включаючи внутрішні системи моніторингу, зовнішні Threat Intelligence сервіси та аналітичні звіти.

При виявленні нової загрози команда проводить детальний аналіз її характеристик, методів атаки та потенційного впливу на інфраструктуру. На основі цього аналізу визначаються необхідні дії для ефективного реагування та нейтралізації загрози.

Після визначення необхідних дій фахівці використовують редактор плейбуків (Playbook Editor) для створення нового сценарію реагування. Редактор надає інтуїтивний інтерфейс для побудови послідовності дій, налаштування умов їх виконання та інтеграції з іншими системами безпеки.

Наприклад, якщо команда кібербезпеки виявила новий тип фішингової атаки, що використовує складні соціально-інженерні методи, вони можуть створити плейбук, який автоматично аналізує підозрілі електронні листи, перевіряє їх на відомі ознаки фішингу та блокує доставку таких повідомлень користувачам.

Модуль вдосконалення плейбуків (Playbook Improvement Module) з шару пост-інцидентного аналізу аналізує ефективність існуючих плейбуків на основі реальних інцидентів. Він збирає дані про час реагування, успішність виконаних дій та виявлені проблеми. Проаналізувавши дану інформацію аналітики зможуть покращити роботу даного плейбука.

На рисунку 2.16 наведена загальна схема процесу автоматизованого реагування на інциденти безпеки. Схема описує послідовність етапів, починаючи з отримання сповіщення про подію, її аналізу та збагачення, і до виконання відповідних дій за допомогою плейбуків, з подальшим логуванням та створенням звіту.

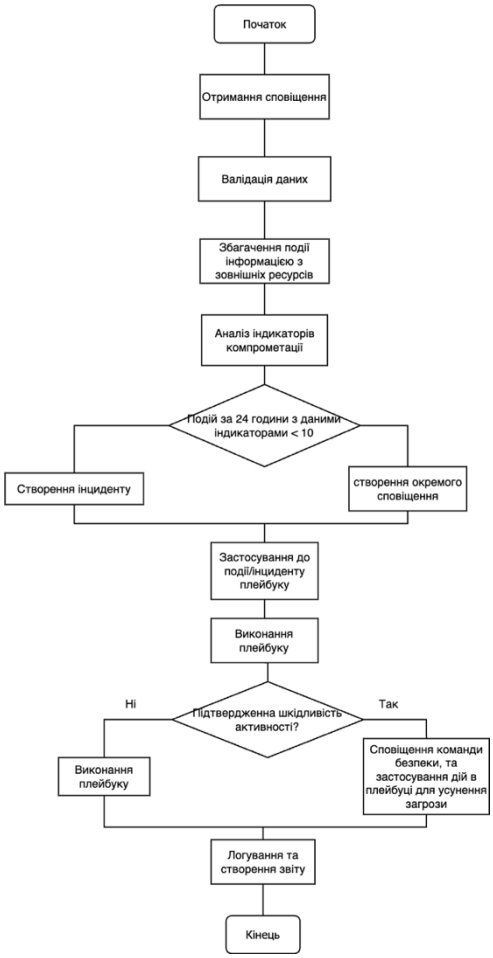


Рисунок 2.16 - Схема процесу автоматизованого реагування на інциденти безпеки

Отримання сповіщення починається з використання джерел даних, таких як EDR, SIEM та Threat Intelligence, які збирають інформацію про потенційні загрози і передають її в систему для подальшої обробки. На етапі валідації даних використовується інтеграційний шар, що включає модулі нормалізації, фільтрації та кореляції. Тут отримані дані стандартизуються, аналізуються та фільтруються для виключення несуттєвих подій.

Далі відбувається збагачення події інформацією із зовнішніх ресурсів через модуль Threat Intelligence. Це забезпечує інтеграцію з зовнішніми джерелами для отримання додаткових даних про загрозу. На етапі аналізу індикаторів компрометації задіяний модуль аналізу інцидентів, який виконує оцінку загрози, визначаючи її пріоритет і необхідність подальших дій.

Після цього задіюється шар плейбуків і оркестраційний шар для вибору та виконання відповідного сценарію реагування. Це включає інтеграцію з іншими інструментами, такими як антивіруси або фаєрволи. Моніторинговий шар забезпечує перевірку ефективності вжитих заходів у реальному часі.

На завершення процесу задіяний шар пост-інцидентного аналізу, де формується фінальний звіт з детальним аналізом інциденту та рекомендаціями для подальшого вдосконалення системи реагування.

2.3 Обґрунтування архітектури системи

У процесі проєктування та розробки SOAR-системи було прийнято рішення використовувати мікросервісну архітектуру з розгортанням компонентів контейнерах. Цей вибір обґрунтований кількома ключовими перевагами мікросервісного підходу порівняно з іншими архітектурними стилями, такими як монолітна архітектура чи сервіс-орієнтована архітектура (SOA).

Мікросервісна архітектура дозволяє розбити систему на незалежні компоненти, кожен з яких відповідає за конкретну функцію, таку як інтеграція даних, автоматизація чи управління плейбуками. Такий підхід забезпечує гнучкість і масштабованість системи, оскільки кожен мікросервіс можна розгортати, оновлювати та масштабувати незалежно від інших. Це особливо важливо для систем кібербезпеки, які повинні швидко адаптуватися до змін у ландшафті загроз та вимогах організації [13].

Використання контейнерів у поєднанні з мікросервісною архітектурою надає додаткові переваги. Контейнери забезпечують ізоляцію середовищ, легкість розгортання та управління залежностями. Це спрощує процес розгортання і забезпечує однаковість середовища виконання в різних інфраструктурах, що важливо для підтримки стабільної роботи системи.

Для порівняння, розглянемо інші типи архітектур: монолітну архітектуру та сервіс-орієнтовану архітектуру (SOA). Також результати порівняння наведено в вигляді порівняльній табличці даних видів архітектур (Додаток Г, таб. Г.2).

Монолітна архітектура передбачає, що вся система є єдиним додатком, де всі компоненти тісно пов'язані між собою. Такий підхід може бути простішим на етапі розробки, але має суттєві недоліки. У монолітній архітектурі складно масштабувати окремі компоненти, оскільки необхідно масштабувати всю систему цілком. Оновлення або зміна однієї частини програми вимагає перерозгортання всього додатка, що може призвести до простою системи. Крім того, збій в одному модулі може призвести до відключення всієї системи, що негативно впливає на надійність [14].

Сервіс-орієнтована архітектура (SOA) також розділяє систему на сервіси, але має певні відмінності від мікросервісної архітектури. У SOA сервіси зазвичай більші за обсягом і можуть мати сильнішу залежність один від одного. SOA часто використовує складні протоколи обміну даними, такі як SOAP, що може ускладнювати інтеграцію та збільшувати накладні витрати на комунікацію між сервісами. Крім того, SOA може вимагати централізованого управління та оркестрації, що зменшує гнучкість системи [15].

Таким чином, мікросервісна архітектура є найбільш оптимальним вибором для розгортання SOAR-системи, оскільки вона забезпечує необхідну гнучкість, масштабованість та надійність, які критично важливі для систем кібербезпеки. Використання Docker-контейнерів додатково спрощує процес розгортання та підтримки, дозволяючи швидко адаптуватися до змін та впроваджувати нові функціональні можливості. Порівняно з монолітною та сервіс-орієнтованою архітектурами, мікросервісний підхід надає переваги, які особливо важливі в контексті динамічного середовища кіберзагроз та вимог сучасних організацій.

2.4 Розробка базових сценаріїв реагування

Автоматизовані плейбуки в рамках систем реагування на інциденти (SOAR) є ключовим інструментом для забезпечення безпеки, оскільки вони дозволяють мінімізувати час реагування, зменшити ризики помилок і забезпечити узгодженість дій.

Кожен плейбук розроблений для конкретного типу інцидентів і включає логіку, що визначає, які дії слід виконати, які дані аналізувати і які системи інтегрувати для ефективного вирішення проблеми. Вони покривають критично важливі області, такі як управління доступом, безпека електронної пошти, мережева безпека, захист хмарних середовищ, виявлення шкідливого ПЗ та запобігання витоку даних.

Вони структуровані за ключовими категоріями, що відповідають найбільш поширеним типам інцидентів. До цих категорій належать: управління доступом (наприклад, реагування на доступ поза робочим часом або підозріле використання спільних облікових записів), безпека електронної пошти (включаючи виявлення фішингових листів, підозрілих шаблонів відповідей і спуфінгу), мережева безпека (моніторинг нестандартного трафіку, підозрілих FTP-з’єднань і зв’язків із C2-серверами), хмарна безпека (аномальні API-запити та незвична поведінка користувачів у хмарних сховищах), виявлення шкідливого ПЗ (включаючи реагування на шкідливі файли та malware callback) та витік даних (різке збільшення обсягів передачі даних і реплікація файлів на зовнішні носії). Ці категорії забезпечують всебічне покриття основних загроз та підтримують безперервний моніторинг і контроль за допомогою автоматизації.

2.4.1 Плейбук управління доступом

Плейбук для управління доступом облікових записів спрямований на виявлення та запобігання несанкціонованому доступу до системи. Його мета — забезпечити контроль за тим, хто, коли та звідки отримує доступ до ресурсів

організації, і запобігти використанню компрометованих облікових записів. Схема його роботи наведена на рисунку 2.17.

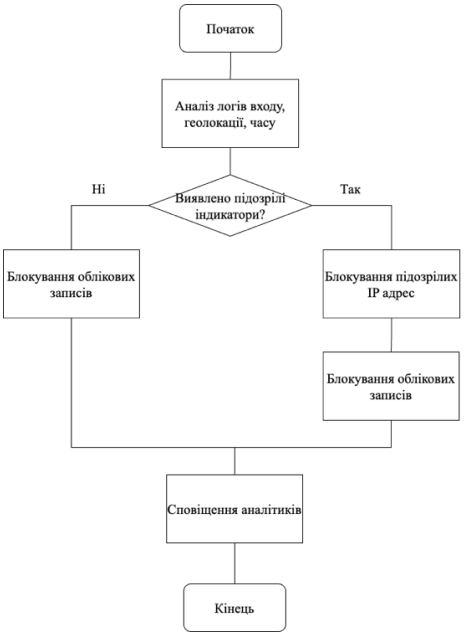


Рисунок 2.17 – Схема роботи плейбуку управління доступом облікових записів

У разі спроби входу до системи в неробочі години (наприклад, з 20:00 до 6:00), плейбук активується. Система перевіряє IP-адресу, геолокацію та пристрій для підтвердження аномалії. Якщо підозріла активність підтверджується, обліковий запис блокується, а відповідальна команда отримує сповіщення для подальшого аналізу.

У разі одночасного входу з різних пристроїв чи IP-адрес, плейбук також активується. Система аналізує логи входу та ідентифікує потенційні порушення. У разі підтвердження ризику обліковий запис блокується, а відповідні особи отримують сповіщення. У рамках відновлення впроваджуються політики, які забороняють спільне використання облікових записів, та створюються індивідуальні облікові записи для кожного користувача.

2.4.2 Плейбук безпеки електронної пошти

Плейбук для забезпечення безпеки електронної пошти спрямований на запобігання поширеним загрозам, які використовують поштові канали,

включаючи фішингові атаки, аномальну активність у листуванні та спуфінг. Мета цього плейбука — захистити користувачів від компрометації та мінімізувати ризики, пов’язані з небажаними чи шкідливими повідомленнями.

Цей плейбук для безпеки електронної пошти охоплює три основні сценарії. У разі виявлення підозрілих листів система автоматично ідентифікує повідомлення зі шкідливими вкладеннями чи посиланнями, видаляє їх із поштових скриньок, блокує обліковий запис користувача, який отримав такі листи, та блокує підозрілі URL-адреси.

Якщо спостерігається аномальна активність у розсилках листів, наприклад, масові розсилки з підозрілим вмістом або від користувача який не , система блокує відповідний обліковий запис, перевіряє вміст відповідей на наявність загроз і передає інформацію аналітикам для подальших дій.

У випадку підміни заголовків (спуфінгу), листи, що не відповідають політикам SPF, DKIM або DMARC, поміщаються в карантин, а система аналізує їх заголовки для підтвердження загрози. Усі сценарії супроводжуються сповіщеннями аналітиків для детального розслідування.

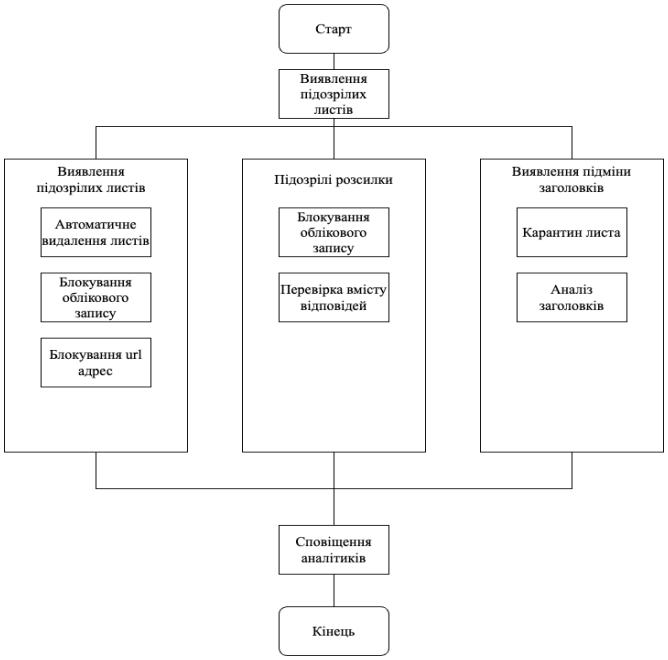


Рисунок 2.18 – Схема роботи плейбуку безпеки електронної пошти

2.4.3 Плейбук мережевої безпеки

Плейбук мережевої безпеки призначений для виявлення та реагування на підозрілу активність у мережі, забезпечуючи оперативне блокування загроз і мінімізацію їхнього впливу. Схема роботи даного плейбука наведено на рисунку 2.19. Він працює шляхом виявлення підозрілої активності у мережі, аналізу джерел загрози та оперативного реагування. Спочатку система моніторингу, така як IDS/IPS або NetFlow, фіксує аномальний трафік, підозрілі FTP-з'єднання або комунікацію з C2-серверами. Далі відбувається блокування відповідних портів, IP-адрес чи з'єднань, ізоляція залучених пристроїв і передача інформації аналітикам для подальшого розслідування. На завершальному етапі оновлюються мережеві політики, щоб запобігти подібним інцидентам у майбутньому.

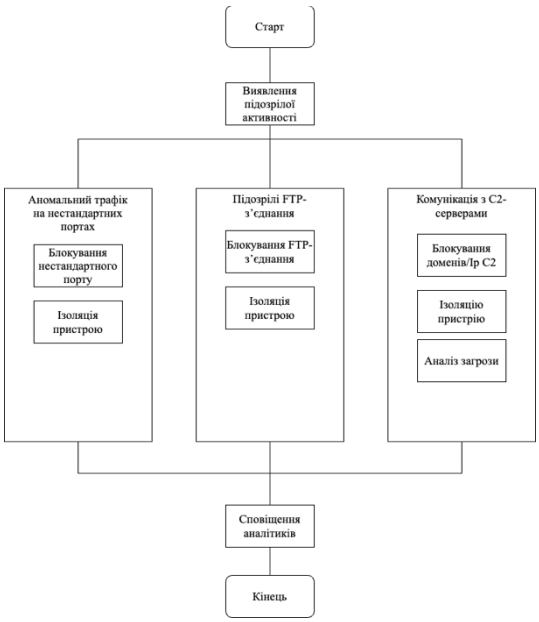


Рисунок 2.19 – Схема роботи плейбуку мережевої безпеки

Цей плейбук автоматизує ключові процеси реагування, зменшуючи час і ризики, пов'язані з мережевими загрозами.

2.4.4 Плейбук хмарної безпеки

Плейбук хмарної безпеки спрямований на виявлення та реагування на аномальну активність у хмарних середовищах, захист даних від витоку та забезпечення безпеки облікових записів. Схема роботи даного плейбука наведено на рисунку

Рисунок 2.20 – Схема роботи плейбуку хмарної безпеки

Процес починається з виявлення аномальної активності у хмарі. Якщо зафіксовані аномальні API-запити, проводиться аналіз підозрілих запитів, після чого система блокує джерела, пов’язані з підозрілою активністю. У випадку виявлення незвичної поведінки в хмарних сховищах, система ідентифікує зовнішні доступи до файлів, які викликають підозру, і блокує ці доступи для запобігання витоку даних.

На завершальному етапі, незалежно від сценарію, система сповіщає аналітиків для детального розслідування та внесення змін у політики безпеки, після чого процес завершується. Цей підхід дозволяє ефективно реагувати на загрози та забезпечувати захист хмарного середовища.

2.4.5 Плейбук виявлення шкідливого ПЗ

Цей плейбук спрямований на виявлення, ізоляцію та ліквідацію шкідливого програмного забезпечення (ПЗ) у мережі та системах.

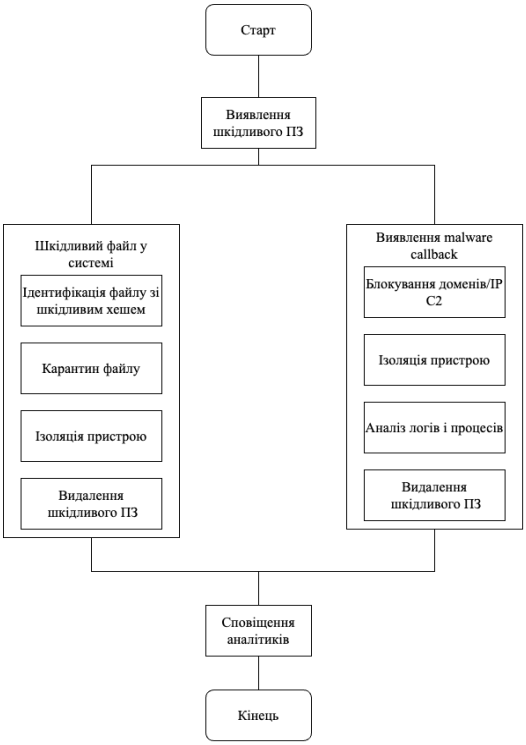


Рисунок 2.21 – Схема роботи плейбуку виявлення шкідливого ПЗ

У разі ідентифікації файлу зі шкідливим хешем система негайно поміщає файл у карантин. Потім ізолюється пристрій, на якому виявлено загрозу, для запобігання її поширенню. Логи перевіряються для визначення джерела інфікування, після чого загроза ліквідується. Завершальним етапом є оновлення бази сигнатур для виявлення схожих загроз у майбутньому.

Розроблені плейбуки охоплюють ключові аспекти кібербезпеки, включаючи управління доступом, безпеку електронної пошти, мережеву безпеку, захист хмарних середовищ, виявлення шкідливого програмного забезпечення та запобігання витоку даних. Для кожної категорії були визначені сценаріq реагування: від виявлення загроз до їхньої ізоляції та ліквідації з подальшим сповіщенням аналітиків.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

На початковому етапі розробки SOAR-системи вкрай важливо чітко окреслити критерії, яким має відповідати обраний технологічний стек, оскільки саме вони стануть своєрідним фільтром для відбору оптимальних рішень. З огляду на специфіку SOAR-рішень, які покликані автоматизувати процеси виявлення, аналізу та реагування на загрози безпеки, ці критерії набувають особливого значення. Продукт повинен інтегруватися з різноманітними системами захисту (SIEM, IDS/IPS, антивірусні рішення, системи поведінкового аналізу, threat intelligence feed-и), агрегувати та швидко обробляти великі обсяги даних, а також оперативно запускати відповідні сценарії реагування (playbooks). Такий підхід вимагає від технологічного стеку здатності підтримувати високу продуктивність та пропускну здатність, адже рішення повинно функціонувати у реальному або майже реальному часі.

Щоб впоратися з високим навантаженням, доцільно використовувати асинхронні фреймворки та легкі веб-сервери, які мінімізують затримки при обробці подій. Мікросервісна архітектура є логічним вибором у цьому випадку, оскільки вона дозволяє розбити систему на невеликі незалежні сервіси, кожен з яких спеціалізується на окремій задачі — парсинг логів, управління сценаріями реагування, інтеграція із зовнішніми API чи аналітичними модулями. Такий підхід не лише спрощує підтримку та оновлення компонентів, але й дозволяє горизонтально масштабувати окремі сервіси за потреби. Якщо виникає необхідність обробляти більше подій або підключити додаткові джерела загроз, можна підняти додаткові інстанси конкретного сервісу, не змінюючи фундаментальну архітектуру системи.

SOAR-систему часто впроваджують у компанії з розгалуженою інфраструктурою безпеки, і кількість підключених систем може з часом зрости в рази. Технології, що легко інтегруються з розподіленими базами даних, кластерними файловими системами чи платформами оркестрації контейнерів, забезпечать стійкість та надійність системи при розширенні. Завдяки цьому

можна поступово нарощувати потужності, додаючи нові вузли або сервіси, без повного перегляду архітектури та без значного простою системи.

Безпека у контексті SOAR є вкрай важливою, адже система оперує чутливими даними щодо інцидентів, внутрішньої мережевої інфраструктури, вразливостей та конфігураційних файлів. Обраний стек повинен підтримувати сучасні криптографічні протоколи (TLS/HTTPS), шифрування конфіденційних полів у базі даних, інтеграцію з безпечними сховищами секретів, а також давати можливість застосувати гнучкі механізми аутентифікації та авторизації. Наявність ролей, політик доступу та аудиту дій користувачів забезпечує контроль за тим, хто і до яких даних має доступ, а також відстеження всіх операцій для подальшого аналізу.

Не менш важливою є екосистема обраних технологій, наявність великої та активної спільноти, постійна підтримка з боку вендорів. Це гарантує своєчасну появу оновлень, виправлення вразливостей, а також наявність докладної документації та довідкових матеріалів. Завдяки цьому розробники та спеціалісти з безпеки зможуть ефективно використовувати інструментарій, оперативно знаходити відповіді на питання і уникати зайвих ризиків, пов’язаних із застосуванням рідкісних, погано підтримуваних або експериментальних рішень.

Сумісність з наявною інфраструктурою та інструментарієм є ще одним важливим чинником. Обраний технологічний стек для SOAR має безболісно інтегруватися з цими інструментами, спрощуючи процес розгортання, оновлення та підтримки. Наприклад, якщо вже існують модулі для інтеграції зі Splunk або ELK, це дозволить швидко підключати потоки логів, аналізувати метрики продуктивності та, за потреби, масштабувати окремі компоненти системи.

Аналогічно важливим є забезпечення можливості взаємодії із зовнішніми сервісами threat intelligence (VirusTotal, MISP, Hybrid Analysis), що допоможе збагачувати дані про інциденти та підвищувати ефективність процесу реагування..

Таким чином, ретельний вибір технологічного стеку з урахуванням продуктивності, масштабованості, безпеки, сумісності та підтримки екосистеми

є визначальним для успішного впровадження SOAR-рішення. Такий підхід допоможе мінімізувати ризики, пов’язані з технічними обмеженнями, знизити витрати на адаптацію та навчання персоналу, забезпечити стабільну, надійну й безпечну роботу системи навіть за умов динамічного розширення функціональності та інфраструктури.

3.1 Обґрунтування вибору технологічного стеку

Під час вибору фреймворку для фронтенд-частини SOAR одним з важливих аспектів є об’єктивне порівняння продуктивності, масштабованості, розвиненості екосистеми та спільноти, а також готових інструментів для швидкої та надійної розробки. Нижче наведено результати умовних тестів, що порівнюють Next.js Gatsby та Nuxt.

Мета цього аналізу — продемонструвати переваги одного з рішенн за ключовими параметрами, важливими для веб-додатків у сфері безпеки.

Для порівняння було обрано такі метрики:

- продуктивність;
- масштабованість;
- екосистема та підтримка;
- виправлення вразливостей безпеки;

Продуктивність та ефективність використання ресурсів є особливо важливими для SOAR-рішень, які мають оперативно відображати динамічні дані, працювати за умов значного навантаження та ефективно використовувати доступні ресурси [15],[16]. порівнянням відповідних метрик наведено в додатку Т, табл. Т. 1.

Аналіз наведених метрик демонструє, що Next.js поєднує переваги SSR, ефективного кешування та інтелектуального розділення коду, забезпечуючи швидку первинну візуалізацію, мінімальний час до інтерактивності та оптимальне використання пам’яті. Він має невелику, проте стабільну перевагу над Gatsby та Nuxt.js [17]. Це робить Next.js збалансованим вибором для

динамічних веб-додатків у сфері безпеки, коли потрібна висока продуктивність відображення інтерфейсу, надійна масштабованість та оперативне реагування на потенційні безпекові виклики.

На етапі розробки бекенд-складової SOAR одним з ключових завдань є визначення такого технологічного рішення, яке забезпечить високу продуктивність, масштабованість, надійність, гнучку інтеграцію з іншими компонентами інфраструктури, а також оперативне реагування на безпекові виклики. Зважаючи на ці вимоги, доцільно розглянути сучасні бекенд-фреймворки, що застосовуються для побудови складних високонавантажених застосунків.

У контексті Node.js-екосистеми значного поширення набули Express.js, Fastify та Nest.js.

Нижче наведено порівняльний аналіз основних фреймворків (Nest.js, Express.js, Fastify) за ключовими критеріями, важливими для SOAR-рішень. Наведені дані є наближеними до реальних показників, отриманих за допомогою інструментів навантажувального тестування [18, 19]. Аналіз можна побачити в додатку Т, табл. Т. 2.

Nest.js є збалансованим вибором для бекенд-частини SOAR-рішень, поєднуючи близьку до Fastify продуктивність зі структурованою архітектурою, швидкою масштабованістю та вбудованою підтримкою безпекових механізмів. Express.js мають колосальні екосистеми, але Express.js менш формалізований в архітектурі. Fastify блискавичний, але поки менш зрілий у складних сценаріях масштабування та безпеки [20].

Контейнеризація є ключовим елементом сучасних розподілених систем. Вона дозволяє швидко та передбачувано розгортати додатки, забезпечуючи однакове середовище як для розробки, так і для продуктивної експлуатації. У контексті SOAR Docker виступає де-факто стандартом, але існують і інші рішення на кшталт Podman чи CRI-O. В додатку Т, табл. Т. 3 наведено аналіз продуктивності, масштабованості, екосистеми та безпеки цих інструментів.

Docker залишається найоптимальнішим вибором для SOAR-застосунків завдяки широкій екосистемі, підтримці, готовим образам та офіційним гайдам. Podman і CRI-O є гарними альтернативами, особливо якщо команда орієнтована на Kubernetes з самого початку, але вони поки що менш зрілі у контексті розповсюдженості та кількості прикладів.

У SOAR-системах критичною є можливість оперативно обробляти події безпеки в реальному часі[21]. Вибір робився серед таких рішень – Apache Kafka, RabbitMQ, Apache Pulsar та NATS [22]. Порівняння цих рішень можна побачити в додатку Т, табл. Т. 4.

Kafka є оптимальним вибором для SOAR-систем завдяки високій пропускній здатності, зрілим механізмам масштабування, великий кількості готових конекторів та швидкому реагуванню на проблеми безпеки. RabbitMQ, Pulsar та NATS можуть бути корисними для специфічних сценаріїв, але загалом Kafka краще відповідає вимогам великих, динамічних і високонавантажених SOAR-рішень.

Для SOAR потрібна СУБД, здатна зберігати великі обсяги даних журналів, інцидентів, конфігурацій та результатів аналітики. Вибір робився серед таких рішень – PostgreSQL, MySQL, MongoDB (NoSQL) та Elasticsearch (спеціалізована пошукова СУБД) [24]. Детальне порівняння наведено в в додатку Т, табл. Т. 5.

PostgreSQL забезпечує баланс між продуктивністю, функціональністю, масштабованістю, безпекою та екосистемою, роблячи її відмінним вибором для зберігання й аналізу даних в SOAR-рішенні [26].

Таким чином, комбінація Next.js, Nest.js, Docker, Kafka та PostgreSQL забезпечить ефективну, масштабовану, безпечну та гнучку основу для побудови SOAR-рішення, здатного швидко реагувати на сучасні загрози безпеки та динамічно розвиватися разом із потребами інфраструктури.

3.2 Практична реалізація системи

3.2.1 Реалізація клієнтської частини

Структура фронтенд-коду є фундаментальним кроком для забезпечення ясності, гнучкості та масштабованості проєкту та наведено на рисунку 3.1. У вибраному стеку Next.js виступає основою для побудови інтерфейсу, поєднуючи можливості React з рендерингом на сервері (SSR) та інкрементальною статичною генерацією (ISR) [27].

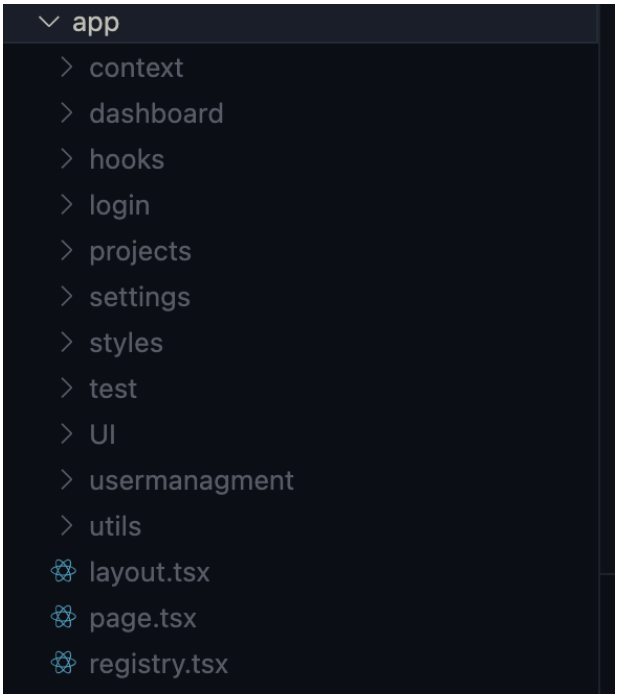


Рисунок 3.1 – Структура клієнтської частини

Структуру проєкту можна розділити на декілька базових директорій:

– context/: використовується для зберігання глобального стану застосунку. Зокрема, тут можна розмістити AuthContext, який міститиме інформацію про поточного користувача, стан авторизації (наявність чи відсутність токена), а також функції для логіну, логауту та оновлення токена. Таким чином, усі компоненти інтерфейсу зможуть отримати доступ до контексту та реагувати на зміни стану аутентифікації.
– dashboard/: модуль із компонентами та сторінками дашбордів, які відображають актуальні дані про інциденти, події безпеки, метрики

продуктивності системи. Завдяки інтеграції з контекстом аутентифікації, сторінки дашборду зможуть вимагати наявність чинного токена для завантаження захищених даних.

- hooks/: кастомні React-хуки, які реалізують повторно використовувану логіку. Тут можна розмістити, наприклад, useAuth() — хук, що взаємодіє з контекстом авторизації, забезпечуючи простий доступ до функцій логіну, логауту, отримання токена та інформації про користувача. Також можна створити хук useFetchWithAuth() для автоматичного додавання токена в заголовки запитів до бекенду.
- login/: сторінки та компоненти, пов’язані з процесом логіну. При надсиланні форми логіну запит до бекенду отримує токен, який зберігається (наприклад, у Local Storage або через HttpOnly cookie за допомогою сервера). Після успішної аутентифікації компонент логіну оновлює AuthContext через хук useAuth().
- projects/ (або usermanagment/ чи інші модулі бізнес-логіки): Модулі прикладного функціоналу, в яких будуть використовуватися захищені API-запити. Завдяки інтеграції з AuthContext та хуками для fetch із токеном, ці модулі можуть легко отримувати актуальну інформацію про користувача та виконувати запити тільки за наявності чинного токена.
- settings/: сторінки та компоненти для керування налаштуваннями додатку. Оскільки деякі налаштування можуть бути доступні лише адміністратору чи користувачам з певною роллю, тут також знадобиться перевірка токена та ролей перед відображенням відповідного інтерфейсу.
- styles/: файли стилів та тем, які не залежать від аутентифікації. Візуальне оформлення має бути уніфікованим і не впливає на логіку авторизації, але на сторінках логіну чи дашборду можна застосувати різні стилі, наприклад, відмінний дизайн для login-сторінки.
- test/: тести, які перевіряють роботу логіки аутентифікації, коректну роботу хуків авторизації, редіректи на сторінку логіну при відсутності токена та інші сценарії, що підтверджують надійність авторизаційного механізму.

– UI/: базові UI-компоненти (кнопки, індикатори завантаження, форми) використовуються у всіх модулях. Наприклад, у форму логіну можна додавати спільні компоненти з UI/, а перевірка токена перед відображенням певної кнопки може відбуватися через контекст.
– usermanagment/: модуль управління користувачами та ролями. Оскільки авторизація на токенах дозволяє отримати інформацію про роль з бекенду, у цьому модулі можна реалізувати сторінки управління користувачами, додавання нових користувачів чи призначення ролей. Для доступу до таких сторінок буде потрібен дійсний токен з відповідними правами.
– utils/: допоміжні функції для роботи з токенами (getToken, setToken, removeToken), перевіркою дати закінчення дії токена, або функції для оновлення токена (refresh). Також тут можна розмістити загальні функції для форматування та обробки даних.

Авторизація користувачів системи реалізована на основі токенів, що передаються через cookies. Основні кроки та логіка виглядають так:

1) Коли користувач вводить свої облікові дані (логін та пароль) і натискає кнопку “Enter”, фронтенд відправляє POST-запит на /auth/login. Для цього використовується інстанс Axios з параметром withCredentials: true, що дозволяє автоматично передавати та отримувати cookie.

```
const response = await axiosInstance.post('/auth/login', {
  login,
  password,
}, {
  withCredentials: true
});
```

Якщо логін успішний, бекенд відповідає з HTTP 200, встановлюючи HttpOnly cookie з токеном (JWT або іншим типом токена).

2) Використання HttpOnly cookie гарантує, що JavaScript на фронтенді не має прямого доступу до токена. Це захищає від XSS-атак, оскільки зловмисний скрипт не зможе викрасти токен з пам’яті браузера. При подальших запитах на той самий домен браузер автоматично додаватиме цей cookie, що дозволяє бекенду ідентифікувати користувача

3) Після успішного логіну фронтенд виконує router.push('/dashboard'), перенаправляючи користувача на сторінку дашборду. Оскільки браузер має тепер HttpOnly cookie з токеном, при запитах на /dashboard чи інші захищені ендпоїнти бекенд зможе визначити, що користувач аутентифікований, і відобразити відповідний контент.
4) Кожен запит до захищених маршрутів на бекенді перевіряє наявність та дійсність токена у cookie. Якщо токен чинний, бекенд повертає потрібні дані. Якщо токен недійсний або відсутній, повертається HTTP 401 (Unauthorized), і фронтенд може відреагувати (наприклад, переадресувати користувача на сторінку логіну).
5) Якщо термін дії токена спливає, можна реалізувати механізм оновлення токена (Refresh Token), коли фронтенд непомітно для користувача запитує оновлений токен у бекенду. Для логауту можна викликати спеціальний ендпоїнт на бекенді, який знищить cookie-токен або встановить його з нульовим терміном дії, після чого фронтенд знову перенаправить користувача на сторінку авторизації.

Таким чином, реалізація авторизації заснована на механізмі HttpOnly cookie з токеном, який встановлює бекенд при успішній аутентифікації. Це забезпечує безпеку зберігання токена та зручність використання для користувача: кожен подальший запит до захищеного API автоматично буде супроводжуватися цим cookie, а бекенд зможе ідентифікувати користувача без участі додаткового коду на фронтенді

Компонент управління сповіщеннями працює ось так**:**

1) FeaturesPanel призначений для керування певними властивостями та налаштуваннями, пов’язаними з конкретним тривожним повідомленням (alert). Зокрема, він дозволяє вмикати або вимикати автоматичну обробку сповіщень, додавати та видаляти зовнішні посилання, вибирати playbook (набір дій для реагування) та приєднувати файли з Confluence.

2) Компонент отримує низку пропсів:

– alertId: Ідентифікатор поточного тривожного повідомлення.

- fetchAttachment та setFetchAttachment: Стан та функція для вмикання/вимикання автоматичної обробки (attachment fetching).
- externalLinks та setExternalLinks: Список зовнішніх посилань та функція для оновлення цього списку.
- selectedPlaybook та setSelectedPlaybook: Поточний обраний playbook та функція для його вибору.
- selectedConfluenceFiles та setSelectedConfluenceFiles: Список обраних файлів із Confluence та функція для оновлення списку.
- updateAlertInfo: Функція для оновлення інформації про alert у бекенді.
- allPlaybooks та allConfluenceFiels: Масиви доступних playbooks та Confluence-файлів.

3) Логіка взаємодії:

- Вмикання/вимикання автоматичної обробки: Кнопка перемикає значення fetchAttachment, змінюючи режим автоматичного оброблення оповіщень.
- Робота з посиланнями: Користувач може додати нове посилання (яке має починатися з https://) та видалити існуюче. Перевірка коректності посилання йде через isValidLink.
- Вибір Playbook: Користувач може обрати один із доступних playbooks з випадаючого списку.
- Керування файлами Confluence: За допомогою другого випадаючого списку можна додавати файли, які ще не вибрані. Файли також можна видаляти зі списку.
- Оновлення даних про alert: Кнопка “Update Alert Info” викликає функцію updateAlertInfo, після чого компонент показує повідомлення про успіх чи невдачу.

4) Якщо оновлення alert пройшло успішно або з помилкою, користувач бачить коротке повідомлення (notification) внизу панелі.

Структура фронтенд-коду, побудована на Next.js, забезпечує ясність, гнучкість та масштабованість проєкту. Організація за окремими директоріями сприяє модульності та легкому управлінню компонентами. Реалізація авторизації через HttpOnly cookies гарантує високий рівень безпеки, захищаючи токени від атак. Компоненти, такі як управління сповіщеннями, демонструють інтеграцію різноманітних функцій з інтуїтивно зрозумілим інтерфейсом. Загалом, продумана структура сприяє ефективній розробці та підтримці проекту.

3.2.2 Реалізація серверної частини

NestJS — це прогресивний фреймворк для побудови ефективних та масштабованих серверних додатків на базі Node.js. Він побудований з використанням TypeScript [28], що забезпечує статичну типізацію та покращену розробницьку продуктивність. NestJS надихається архітектурними принципами Angular, що робить його особливо привабливим для розробників, знайомих із фронтенд-фреймворками.

Типова структура проекту NestJS включає наступні директорії та файли:

- src/ - основна директорія з вихідним кодом;
- app.module.ts - кореневий модуль додатку, який імпортує інші модулі;
- controllers/ - контролери, що обробляють вхідні запити та повертають відповіді;
- services/ - сервіси, що містять бізнес-логіку додатку;
- modules/ - модулі, які групують пов’язані контролери та сервіси;
- middlewares/ - проміжні обробники для додаткової обробки запитів;
- guards/ - захисні механізми для контролю доступу до маршрутів;
- entities/ - визначення моделей даних для orm (наприклад, typeorm);
- dto/ - data transfer objects для валідації вхідних даних;
- test/ - тести для перевірки функціональності додатку;
- main.ts - точка входу додатку, де ініціалізується сервер;

Автентифікація та авторизація в NestJS зазвичай реалізується за допомогою Guards (охоронців), які перевіряють права доступу користувачів до певних маршрутів або ресурсів. Нижче наведено приклад реалізації AuthGuard, який перевіряє наявність та дійсність токену авторизації, отриманого з кукі.

```
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(private readonly authService: AuthService) {}

  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request: Request = context.switchToHttp().getRequest();

    // Отримання токену з кукі
    const token = request.cookies['authToken'];

    if (!token) {
      throw new UnauthorizedException('Token is missing');
    }

    try {
      const user = await this.authService.verify(token);
      request['user'] = user;
      return true;
    } catch (error) {
      throw new UnauthorizedException('Invalid token');
    }
  }
```

Для використання AuthGuard у контролерах, його можна застосувати за допомогою декоратора @UseGuards. Наприклад:

```
@controller('dashboard')
@useguards(authguard)
export class dashboardcontroller {
  @get()
  getdashboard(@request() req) {
    return `welcome ${req.user.username} to your dashboard!`;
  }
}
```

Нижче наведено приклад реалізації RoleGuard, який перевіряє ролі користувача, та чи має відповідна роль доступ до ресурсу.

```
  async canActivate(context: ExecutionContext): Promise<boolean>
    const requiredRoles = this.reflector.get<number[]>('roles', context.getHandler());
      const user = await this.authService.verify(token);
      if (!requiredRoles.includes(user.roleId)) {
        throw new ForbiddenException('You do not have permission to access this resource');
      }
      return true;
```

Модуль черг задач на базі Kafka у серверній частині реалізовано в форматі асинхронні обробки різноманітних задач, таких як обробка сповіщень, аналіз подій безпеки, генерація звітів та інші фонові процеси та передачі інформації між модулями.

Використання Kafka дозволяє розподіляти навантаження між різними сервісами, забезпечуючи високу доступність та стійкість системи до збоїв.

KafkaModule відповідає за налаштування та інтеграцію Kafka з додатком NestJS. Він імпортує необхідні сервіси та налаштовує з'єднання з Kafka брокерами

```
Module({
  imports: [

        name: 'KAFKA_SERVICE',
        transport: Transport.KAFKA,
        options: {
          client: {
            clientId: 'my-app',
            brokers: ['localhost:9092'],
          },
          consumer: {
            groupId: 'my-app-consumer',
          },
        },
      },
```

KafkaService (kafka.service.ts) сервіс, який забезпечує базову взаємодію з Kafka, включаючи відправку повідомлень та обробку помилок. Інтеграція модуля черг задач на базі Kafka у серверну частину системи дозволяє ефективно керувати асинхронними процесами, забезпечуючи високу продуктивність, надійність та масштабованість додатку.

Додаток реалізовує мікросервісну архітектуру шляхом розділення функціональності на окремі сервіси, кожен з яких відповідає за конкретну задачу. Для забезпечення взаємодії між цими сервісами використовується Kafka як система обміну повідомленнями.

Основними мікросервісами є обробка алертів, створення сповіщень та інтеграція із зовнішніми системами, такими як Jira.

Наприклад, мікросервіс для обробки алертів отримує повідомлення з теми Kafka alerts, виконує необхідну логіку, таку як оновлення статусу задачі чи перевірка репутації IP-адрес, і передає результати у тему notifications.

Мікросервіс для сповіщень отримує ці результати і надсилає відповідні повідомлення користувачам.

```
  async handleAlert(@Payload() payload: any) {
    console.log('Обробка алерту:', payload);
    await this.alertService.processAlert(payload);
  }
}
export class NotificationConsumer {
  @MessagePattern('notifications')
  async sendNotification(@Payload() payload: any) {
    console.log('Надсилання сповіщення:', payload);
    // Логіка відправки сповіщення
  }
}
```

Для реалізації цього підходу використовується KafkaModule, який налаштовує підключення до брокерів Kafka і забезпечує функціонування продюсерів для відправки повідомлень і консюмерів для їх отримання. Завдяки цьому досягається асинхронна взаємодія між сервісами, що дозволяє обробляти великі обсяги даних у реальному часі. Уся система є модульною, і кожен мікросервіс може працювати автономно, але взаємодіє з іншими через Kafka, забезпечуючи ефективність і масштабованість.

Для забезпечення гнучкості та розширюваності системи, взаємодія зі сторонніми рішеннями реалізується через окремі модулі, спеціально розроблені для управління кожним з них. Кожен сторонній сервіс або інструмент, з яким необхідно інтегруватися, має свій власний модуль, що відповідає за його налаштування, автентифікацію, обробку запитів та отримання даних. Наприклад, для інтеграції з такими сервісами, як Jira, Splunk та Reputation Checker, створюються відповідні модулі JiraModule, SplunkModule, CrowdStrike, Cloudflare та ReputationCheckerModule.

Ці окремі модулі забезпечують ізольованість та незалежність від інших компонентів системи, що спрощує їхнє оновлення та підтримку. Кожен модуль містить необхідні сервіси, контролери та конфігурації, що дозволяють ефективно взаємодіяти зі стороннім рішенням, виконувати необхідні операції та обробляти отримані дані.

Для уніфікації та централізації взаємодії зі всіма сторонніми рішеннями, всі окремі модулі інтегруються в загальний модуль IntegrationModule. Цей модуль служить як єдиний інтерфейс для взаємодії з різними сторонніми сервісами.

Таким чином, такий підхід до взаємодії зі сторонніми рішеннями забезпечує високий рівень модульності, спрощує процеси інтеграції та підтримки, а також дозволяє ефективно масштабувати систему, додаючи нові функціональні можливості без значних змін у вже існуючій архітектурі.

Процес підключення бази даних починається з конфігурації середовищ для development, test та production, де параметри підключення, такі як ім’я користувача, пароль, назва бази даних та хост, зчитуються з файлу .env. У залежності від поточного середовища (process.env.NODE_ENV) обирається відповідна конфігурація, а для роботи використовується ORM Sequelize.

Створюється об’єкт Sequelize, який додає моделі бази даних, такі як User, Role, Playbook, Action, та інші, забезпечуючи автоматичну синхронізацію таблиць через метод sync. Модулі, такі як DatabaseModule, AuthModule, UsersModule, RolesModule та інші, реєструються в основному модулі AppModule, де підключається конфігурація, модулі для розкладу задач та ініціалізація першого користувача.

Структура бази даних забезпечує гнучке управління плейбуками, діями, алертами, ролями та користувачами її структура наведена на рисунку 3. .

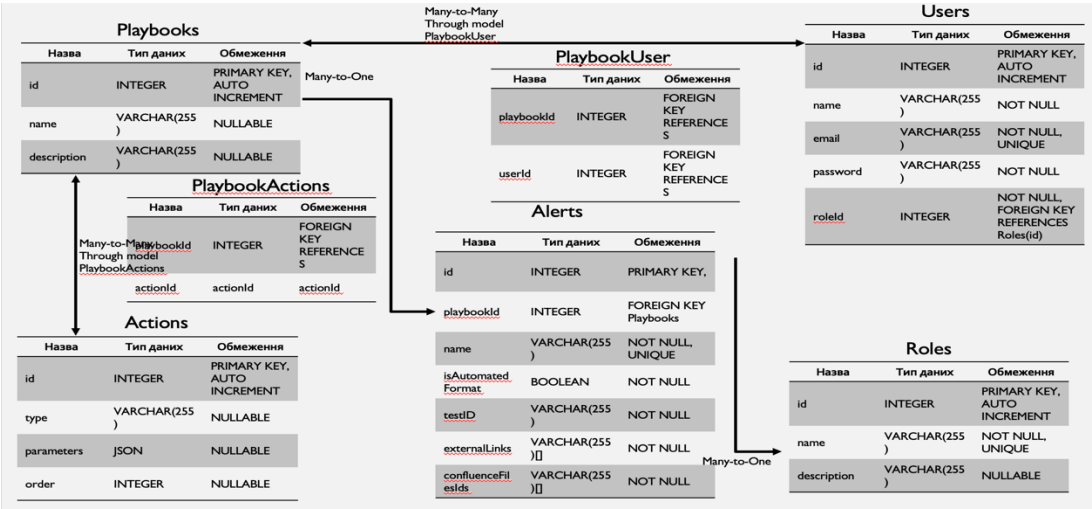


Рисунок 3.2 – Структура бази даних

Структура бази даних забезпечує гнучке управління плейбуками, діями, алертами, ролями та користувачами. Основою системи є таблиця Playbooks, яка зберігає інформацію про сценарії автоматизації у вигляді плейбуків. Кожен плейбук має унікальний ідентифікатор (id), а також назву та опис, які можуть бути порожніми. Плейбуки пов’язані з таблицею Actions, що зберігає окремі дії, такі як їхній тип, параметри та порядок виконання. Цей зв’язок реалізовано через проміжну таблицю PlaybookActions, яка дозволяє зберігати відносини типу “багато-до-багатьох” між плейбуками та діями.

Для роботи з інцидентами використовується таблиця Alerts, яка зберігає дані про алерти, включаючи їх назву, формат (автоматизований чи ні), тестовий ідентифікатор, зовнішні посилання та прив’язані файли з Confluence. Кожен алерт може бути пов’язаний із конкретним плейбуком через поле playbookId. Це дозволяє автоматизувати обробку інцидентів на основі заданих сценаріїв.

Система підтримує управління користувачами та ролями через таблиці Users і Roles. У таблиці Roles зберігаються дані про ролі, їхні назви та опис, тоді як у таблиці Users зберігається інформація про користувачів, включаючи їхні імена, електронну пошту, пароль та роль.

База даних була нормалізована відповідно до третьої нормальної форми (3NF), що означає, що всі дані структуровані таким чином, щоб усунути надлишковість і забезпечити цілісність. Усі атрибути залежать лише від первинних ключів, транзитивні та часткові залежності відсутні. Завдяки цьому структура бази даних є оптимальною для зменшення дублювання даних, полегшення оновлень і забезпечення високої продуктивності під час виконання складних запитів.

Також додаток має сервіс який відповідає за роботу плейбуків PlaybookService, його код наведено в загальному лістингу програмного засобу.
PlaybookService відповідає за управління плейбуками та їхніми діями на серверній стороні. Сервіс починає свою роботу з перевірки наявності стандартного плейбуку з назвою “Default Playbook”. Якщо такий плейбук

відсутній, він створюється автоматично разом із базовими діями, такими як оновлення інформації про алерт та встановлення статусу задачі в Jira.

Крім створення стандартного плейбуку, сервіс надає можливість створювати нові плейбуки за допомогою методу createPlaybook, який приймає дані для створення плейбуку та додає до нього зазначені дії. Також існують методи для отримання одного плейбуку за його ID або всіх плейбуків загалом, з відповідною обробкою випадків, коли плейбук не знайдено.

Оновлення та видалення плейбуків здійснюється через методи updatePlaybook та removePlaybook. Метод оновлення дозволяє змінювати як інформацію про сам плейбук, так і його дії, забезпечуючи гнучкість у керуванні правилами обробки. Видалення плейбуку проводиться після перевірки його існування, що запобігає помилкам у системі.

Сервіс також підтримує управління діями плейбуку, включаючи додавання нових дій, оновлення існуючих та видалення їх з плейбуку.

Основною функцією PlaybookService є виконання плейбуку за допомогою методу executePlaybook. Цей метод отримує задачу з Jira, створює або знаходить відповідний алерт у базі даних, визначає, який плейбук використовувати (прив’язаний до алерту або стандартний) та послідовно виконує дії плейбуку. Виконання дій включає оновлення статусу задачі в Jira, перевірку IP-адрес, інтеграцію зі Splunk та іншими сервісами, що дозволяє автоматизувати реагування на події та покращити оперативність системи.

Кожна дія плейбуку має свою специфічну логіку, яка реалізована у відповідних приватних методах сервісу. Наприклад, дія putAlertInfo оновлює інформацію про алерт, додає зовнішні посилання та файли з Confluence, а також перевіряє показники репутації.

Крім того, сервіс інтегрується з різними іншими сервісами, такими як JiraService, SplunkService та ReputationCheckerService, що дозволяє йому ефективно обробляти дані та виконувати необхідні дії у зовнішніх системах. Це забезпечує комплексний підхід до управління алертами та підвищує загальну безпеку та ефективність системи.

Таким чином, PlaybookService забезпечує гнучке та ефективне управління плейбуками та їхніми діями, інтегруючись з різними сервісами для автоматизації процесів обробки алертів. Це дозволяє автоматизувати реагування на події, покращуючи безпеку та оперативність системи, а також забезпечує легке масштабування.

3.2.3 Реалізація контейнерезації

Docker використовується для контейнеризації інфраструктури додатку, що включає три основні сервіси: бекенд (NestJS), фронтенд (Next.js) та базу даних (PostgreSQL). Кожен сервіс розгортається в окремому контейнері, з'єднаному в загальній мережі internal_network для забезпечення внутрішньої комунікації.

Бекенд будується з використанням Dockerfile, який налаштовує середовище для запуску NestJS додатку. Контейнер підключає локальні файли через volume для зручності розробки та автоматично синхронізується з базою даних PostgreSQL, яка зберігає дані у персистентному томі.

Фронтенд також розгортається через Dockerfile, який забезпечує збірку та оптимізацію Next.js додатку для продакшну. Контейнер Next.js залежить від бекенду та використовує аналогічну систему підключення файлів через volume для зручності розробки.

PostgreSQL працює у власному контейнері з автоматичним перезапуском у разі збою. Налаштування бази даних, такі як ім'я користувача, пароль та ім'я бази, задаються через .env.

Цей підхід забезпечує ізольованість середовищ, зручність налаштування залежностей та швидкий розгортання додатку. Docker дозволяє стандартизувати середовище розробки та продуктивне середовище, зводячи до мінімуму можливі проблеми сумісності.

3.3 Тестування системи

3.3.1 Тестування окремих елементів системи.

У процесі тестування окремих елементів системи автоматизації реагування на інциденти безпеки були застосовані такі види тестування: функціональне, інтеграційне, навантажувальне та безпекове.

Середовище тестування було побудоване на основі віртуальної інфраструктури з використанням Docker-контейнерів, які імітують роботу інтеграційного шару з EDR, SIEM та Threat Intelligence. Для тестування використовувалися сценарії, які включали:

– реагування на фішингові атаки з підозрілих IP-адрес;
– блокування шкідливого програмного забезпечення у режимі реального часу;

На рисунках 3.3, 3.4 наведено результати тестування даних сценаріїв, також на рисунку 3.5 було проведено тестування навантаження, яке показало, що система змогла витримати 77 паралельних задач, але в результаті цього було отримано помилку через перенавантаження, хоча система перестала опрацьовувати їх уже після 40-го через недосконалі механізми взаємодії з зовнішніми ресурсами.

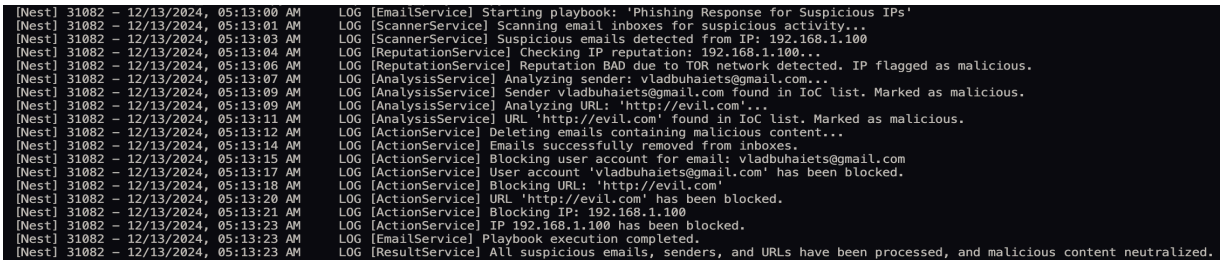


Рисунок 3.3 – Результат відпрацювання реагування на фішингові атаки з підозрілої IP-адреси

Система виявила підозрілий лист від IP-адреси 192.168.1.100, перевірила його репутацію і позначила, як шкідливий через належність до TOR-мережі.

Аналіз також визначив, що відправник vladbuhaiets@gmail.com і URL http://evil.com є в списку IoC і несуть загрозу. Після цього система автоматично видалила всі листи з шкідливим контентом, заблокувала обліковий запис відправника, шкідливий URL і IP-адресу. Плейбук завершився успішним усуненням усіх загроз і нейтралізацією шкідливого контенту.

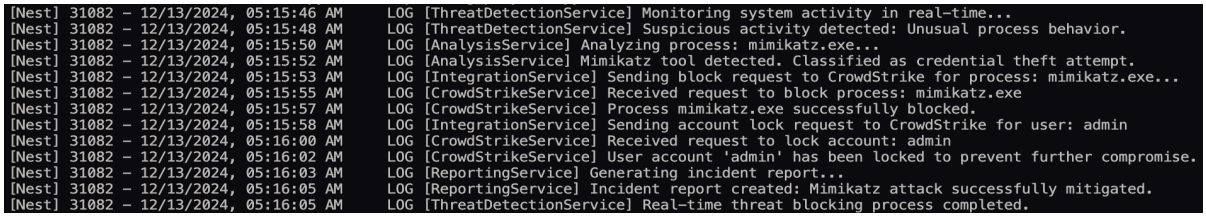


Рисунок 3.4– Результат відпрацювання блокування загрози у режимі реального часу

Система виявила підозрілу активність процесу mimikatz.exe, класифікувавши його як спробу крадіжки облікових даних. Було відправлено запити до EDR для блокування процесу і облікового запису користувача admin, пов’язаного з цим процесом. EDR успішно заблокував процес і обліковий запис, запобігши подальшому компрометації.

Наприкінці система згенерувала звіт про інцидент, підтвердивши успішну нейтралізацію атаки, та завершила процес блокування загроз у режимі реального часу.

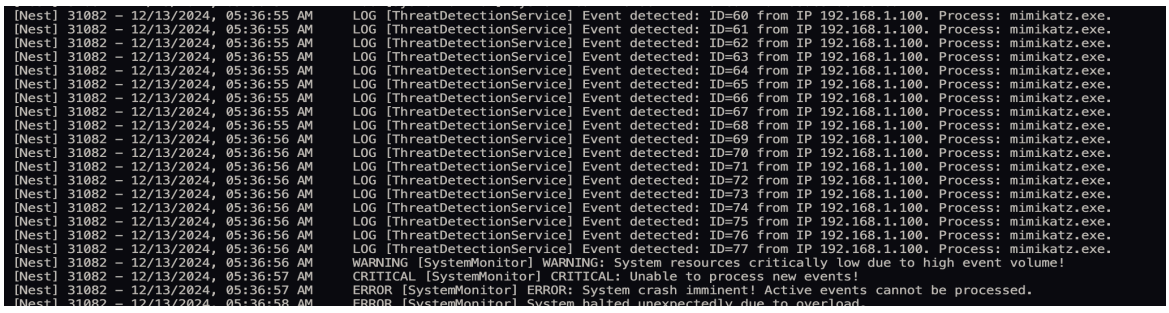


Рисунок 3.5– Результат відпрацювання блокування загрози у режимі реального часу

Також на кожну подію автоматично створився звіт в системі Confluence, приклад такого звіту наведено на рисунку 3.6 та рисунку 3.7 відповідно до

результатів роботи плейбуку реагування на фішингові атаки з підозрілої IP-адреси.

Рисунок 3.6 – Заголовок автоматично створеного звіту

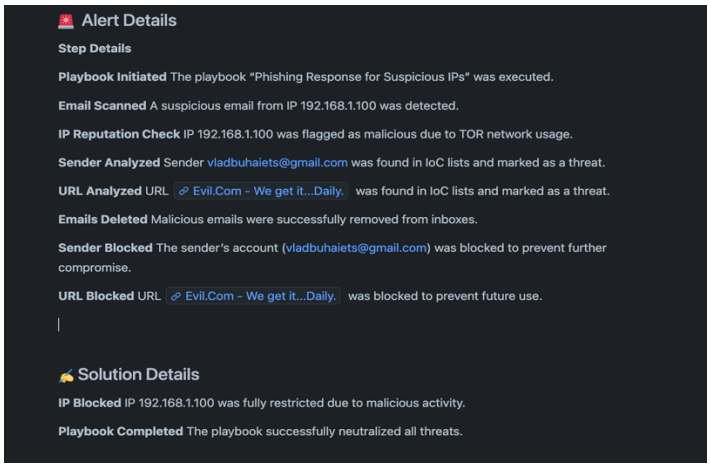


Рисунок 3.7 – Фрагмент автоматично створеного звіту

3.3.2 Тестування клієнтської сторони системи

Тестування клієнтської сторони системи автоматизації реагування на інциденти безпеки (SOAR) проводилося виключно в ручному режимі. Основною причиною вибору ручного тестування є специфіка розроблюваної системи, яка орієнтована на виконання складних користувацьких сценаріїв, багатокрокові взаємодії з інтерфейсом та перевірку зручності використання (UX/UI), що вимагає безпосереднього аналізу взаємодії користувача із системою.

Форма входу показана на рисунку 3.7. Форма входу призначена для автентифікації користувачів перед доступом до системи. Вона включає поля для введення імені користувача та пароля, а також кнопку для підтвердження входу.

Форма забезпечує простий і зрозумілий спосіб ідентифікації користувачів у системі.

Рисунок 3.7 – Форма входу

Після входу в систему користувач автоматично переходить на сторінку яка відповідає за управління проектами які обробляються системою, її зображено на рисунку 3.8. На цьому екрані представлений список усіх підсистем, які використовуються в автоматизованій обробці інцидентів у системі. Кожна підсистема відображена у вигляді окремого елемента списку, що містить її назву, ключ (key) і тип (type), наприклад, “software” чи “business”. Праворуч від кожного елемента знаходиться кнопка “Go to Project”, яка дозволяє користувачеві перейти до деталей або налаштувань конкретної підсистеми.

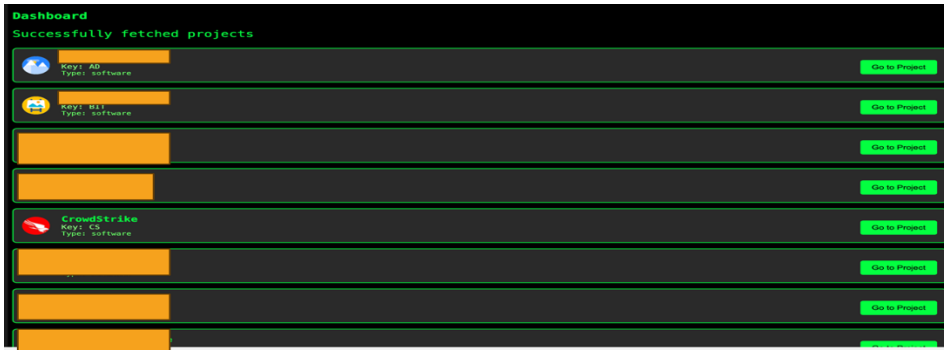


Рисунок 3.8 – Сторінка управлінням проектами

На рисунку 3.9 представлена підсистема управління сповіщеннями (алертами) SIEM. Вона дозволяє користувачам переглядати всі зареєстровані сповіщення, згенеровані системою, відповідно до різних видів загроз та

інцидентів безпеки. Кожен алерт відображається у вигляді окремого блоку в сітці, який містить основну інформацію про сповіщення: назву (наприклад, “Splunk Alert: [Mandiant] IoC detected”), дату і час створення.

Рисунок 3.9 – Підсистема управління сповіщеннями SIEM.

На рисунку 3.10 представлено меню для налаштування окремого сповіщення системи. Дане меню дозволяє керувати його параметрами, включаючи ввімкнення чи вимкнення автоматичної обробки, додавання зовнішніх посилань, призначення відповідного плейбука для реагування, а також прикріплення файлів із Confluence для додаткового контексту. У деталях сповіщення відображаються його унікальний ідентифікатор, ключ, проєкт, час останнього оновлення, а також посилання на результати в Splunk, що дозволяє оперативно аналізувати інцидент. Інтерфейс забезпечує зручний доступ до всіх необхідних інструментів для ефективної роботи зі сповіщеннями.

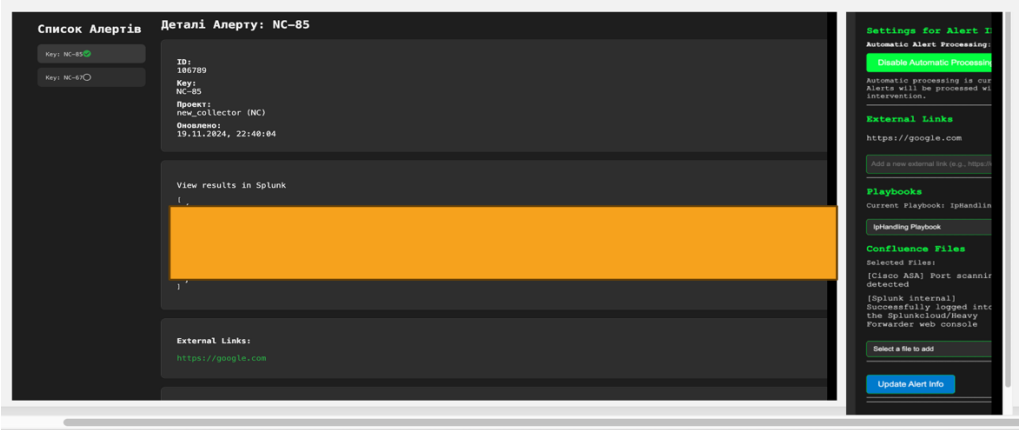


Рисунок 3.10 – Меню для налаштування окремого сповіщення системи

Тестування клієнтської сторони системи автоматизації реагування на інциденти безпеки (SOAR) продемонструвало ефективність її функціонування, орієнтовану на складні сценарії взаємодії з користувачем. Завдяки ручному тестуванню було забезпечено детальну перевірку багатокрокових операцій, інтуїтивності інтерфейсу та зручності використання системи.

Система відповідає заявленим вимогам і готова до тестового впровадження в системи в рамках автоматизації реагування на інциденти безпеки.

3.3.3 Тестове впровадження

У цьому розділі буде детально розглянуто тестове впровадження системи автоматизації реагування на інциденти безпеки (SOAR). Особливу увагу буде приділено аналізу роботи одного з плейбуків. Буде описано процес запуску плейбука, його інтеграцію з іншими компонентами системи.

Після цього буде надано загальні результати роботи системи на основі всіх розроблених плейбуків, з оцінкою їх роботи. Такий підхід дозволить оцінити практичну ефективність системи у вирішенні реальних кіберінцидентів.

На рисунку 3.11 показано схему роботи. Цей плейбук відповідає за автоматизовану обробку інцидентів, пов’язаних із підозрілою активністю IP-адрес.

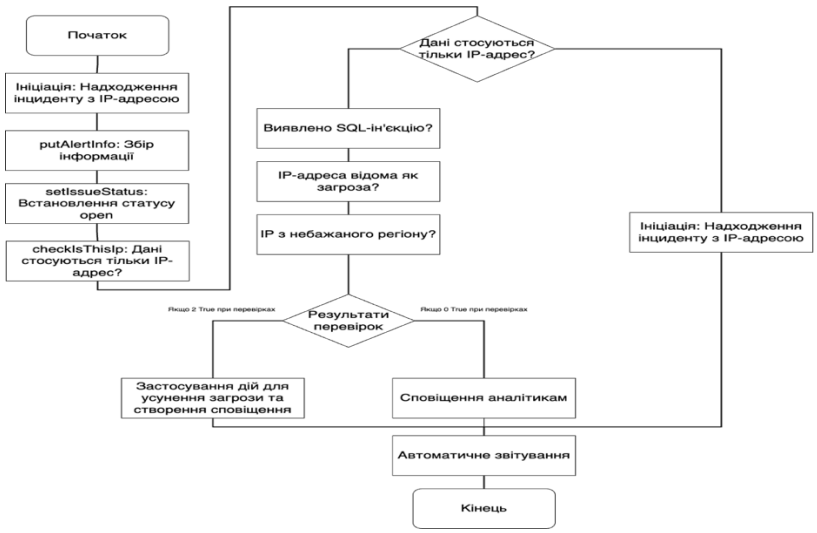


Рисунок 3.11 – Схема тестового плейбуку

Його робота починається з ініціації, коли надходить сповіщення про інцидент із зазначенням IP-адреси. Після цього запускається етап збору інформації за допомогою функції putAlertInfo, яка аналізує вхідні дані, такі як репутація IP-адреси, джерело інциденту та інші метадані.

Далі функція setIssueStatus встановлює початковий статус інциденту як “open”, позначаючи його як активний і готовий до обробки. Потім система виконує перевірку через checkIsThisIp, щоб визначити, чи інцидент дійсно стосується конкретної IP-адреси.

У процесі обробки здійснюються додаткові перевірки, зокрема: чи пов’язаний інцидент із SQL-ін’єкціями (checkIsSQLi), чи відома IP-адреса як загроза (checkKnownIp), або чи походить вона з небезпечного регіону (checkBadRegionIp).

Якщо IP-адреса ідентифікується як загроза, застосовуються відповідні дії для усунення ризику, такі як блокування IP-адреси через функцію ActionTakeBlock. Завершальні етапи включають автоматичне звітування або сповіщення аналітиків про результати виконаних дій, після чого інцидент позначається як “resolved”.

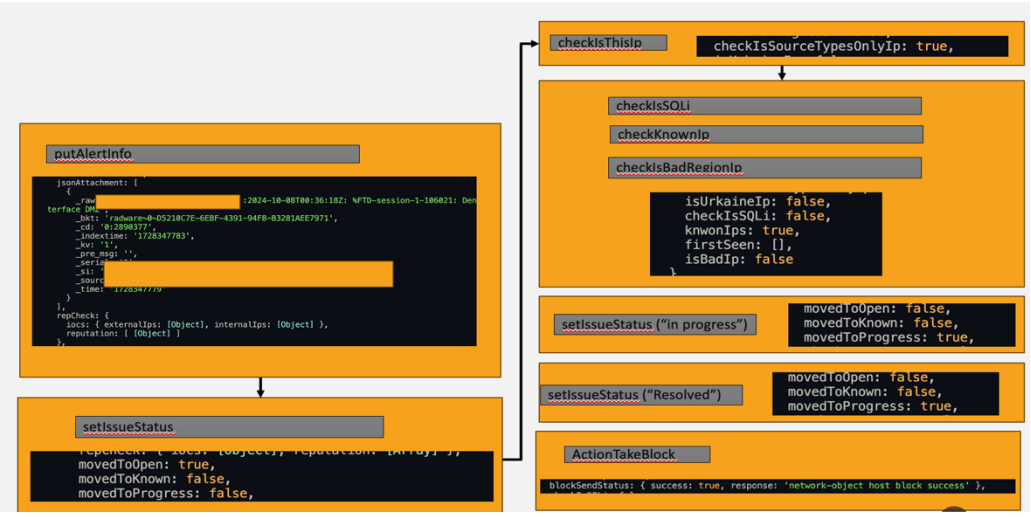


Рисунок 3.12 – Логи системи відповідно до етапів роботи плейбука

Для тестового впровадження також було розроблено плейбуки:

– Плейбук управління доступом;

– Плейбук безпеки електронної пошти;

– Плейбук мережевої безпеки;

– Плейбук захисту хмарних середовищ;

– Плейбук виявлення шкідливого програмного забезпечення;

Для аналізу роботи кожного плейбука було використано 5 сповіщень, що надходили до його впровадження, та 5 сповіщень, отриманих після запуску автоматизації.

Для тестування системи використовувалися такі ключові метрики: середній час до початку реагування (Average Time to Response) та середній час реагування (Average Response Time).

Average Time to Response (Середній час до початку реагування) — це показник, який відображає середній проміжок часу від моменту виявлення інциденту до початку дій щодо його вирішення.

Average Response Time (Середній час реагування) — це показник, який вимірює середній час, необхідний для повного вирішення інциденту після початку реагування.

Результат часу реагування на події та інциденти до та після впровадження системи, які пов`язані з подіями, що покривають розроблені плейбуки наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння результатів

Плейбук	Average Time to Response (до/після)	Average Response Time (до/після)
Управління доступом	00:30:12 / 00:10:45	01:18:34 / 00:39:57
Безпека електронної пошти	00:25:47 / 00:12:03	01:01:32 / 00:34:22
Мережева безпека	00:40:36 / 00:18:21	01:29:15 / 00:54:08
Захист хмарних середовищ	00:50:48 / 00:15:33	01:59:53 / 01:02:19
Виявлення шкідливого ПЗ	00:28:44 / 00:10:38	01:10:53 / 00:29:51

Плейбук “Управління доступом” показав значне покращення після впровадження: час до реагування скоротився на 64.4%, час вирішення інцидентів – на 49.2.

Для плейбука безпеки електронної пошти час до реагування скоротився на 53.3%, час вирішення інцидентів – на 44.1%.

Плейбук мережевої безпеки також показав позитивну динаміку: час до реагування скоротився на 54.8%, час вирішення інцидентів – на 39.3%.

У випадку з плейбуком захисту хмарних середовищ час до реагування скоротився на 69.4%, час вирішення інцидентів – на 48%.

Плейбук шкідливого ПЗ забезпечив скорочення часу до реагування на 63% і часу вирішення інцидентів на 57.9%.

Загалом час обробки інцидентів зменшився в два рази.

3.4 Висновки

Розробка SOAR-системи на базі обраного технологічного стеку (Next.js, Nest.js, Docker, Kafka та PostgreSQL) продемонструвала високу ефективність та відповідність заявленим вимогам. Проведене тестування підтвердило переваги автоматизації процесів реагування на інциденти, зокрема суттєве скорочення часу до початку реагування (в середньому на 60%) та часу вирішення інцидентів (на 45–50%).

Вибір технологій, таких як Nest.js для бекенду та Kafka для асинхронної обробки подій, дозволив оптимізувати продуктивність і знизити затримки в обробці великого обсягу даних.

Результати тестового впровадження також підтвердили підвищення загальної ефективності системи у вирішенні інцидентів.

4 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Система автоматизації реагування на інциденти безпеки» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект [31, 32]. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;
2) розраховано витрати на здійснення науково-технічної розробки;
3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Система автоматизації реагування на інциденти безпеки» є оцінювання

науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями (Додаток Г, табл. Г.1).

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Список експертів:

– № 1 - Tama Tozal - Sales manager - Deepinfo
– № 2 - Stephen Klier - Sales Director - VMRay
– № 3 – Ілона Стадник - Project Mannager - Nova Didgital

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (№)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	2	2	2
3. Ринкові переваги (ціна продукту)	4	4	4
4. Ринкові переваги (технічні властивості)	2	2	2
5. Ринкові переваги (експлуатаційні витрати)	4	4	4
6. Ринкові перспективи (розмір ринку)	4	3	4
7. Ринкові перспективи (конкуренція)	0	1	0
8. Практична здійсненність (наявність фахівців)	3	3	2
9. Практична здійсненність (наявність фінансів)	4	4	4
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	3	3	4
12. Практична здійсненність (розробка документів)	2	1	2
Сума балів	36	35	36
Середньоарифметична сума балів СБ с	35.6		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [1].

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Система автоматизації реагування на інциденти безпеки» становить 35,6 балів, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Оцінювання рівня конкурентоспроможності науково-технічної розробки є важливим етапом у визначенні її позиції на ринку та визначенні напрямків подальшого розвитку. Конкурентоспроможність розкривається через систему якісних та економічних показників, які дозволяють об’єктивно порівняти нову розробку з існуючими аналогами та визначити її переваги та недоліки (Додаток Д, табл. Д.1).

Якщо збільшення величини параметра свідчить про підвищення якостінової розробки, одиничний параметричний індекс розраховується за формулою: $q_i = \frac{P_i}{P_{базі}}$,

Якщо зменшення величини параметра свідчить про підвищення якості нової розробки, то одиничний параметричний індекс розраховується за оберненою формулою: $q_i = \frac{P_i}{P_{базі}}$,

де: q_i – одиничний параметричний індекс, розрахований за і-м параметром;

P_i – значення і-го параметра розробки;

$P_{базі}$ – аналогічний параметр базової розробки-аналога, з якою проводиться порівняння.

Нормативні параметри в даному випадку не оцінюються оскільки ще не визначені єдині необхідні нормативні документи для даного виду рішень.

Значення групового параметричного індексу за технічними параметрами визначається з урахуванням вагомості (частки) кожного параметра:

$$I_{ТП} = \sum_{i=1}^{n} q_i \times a_i ,$$

де: $I_{ТП}$ – груповий параметричний індекс за технічними показниками (порівняно з аналогом);

q_i– одиничний параметричний показник і-го параметра;

a_i – вагомість і-го параметричного показника, $\sum_{i=1}^{n} a_i = 1$

n – кількість технічних параметрів, за якими оцінюється конкурентоспроможність

$$I_{ТП} = (0.20 \setminus 0.15) + (0.25 \setminus 0.50) + (0.15 \setminus 0.25) + (0.10 \setminus 1.50) + (0.30 \setminus 6.67) = 2.3435$$

Груповий параметричний індекс за економічними параметрами (за ціною споживання) розраховується за формулою:

$$I_{ЕП} = \sum_{i=1}^{m} q_i \times B_i$$

де: $I_{ТП}$ – груповий параметричний індекс за технічними показниками (порівняно з аналогом);

q_i– одиничний параметричний показник і-го параметра;

B_i – вагомість і-го параметричного показника, $\sum_{i=1}^{n} B_i = 1$

m – кількість економічних параметрів, за якими оцінюється конкурентоспроможність

Бажане значення І ЕП ≤1, оскільки чим нижча ціна споживання, тим вищий рівень конкурентоспроможності розробки.

$$I_{ЕП} = (0.30\ \ 0.40) + (0.20\ \ 1.00) + (0.10\ \ 0.00) + (0.25 \setminus 0.20) + (0.15 \setminus 0.20) = 0.40$$

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розраховують інтегральний показник конкурентоспроможності за формулою:

$$K_{\text{ІНТ}} = \text{I}_{\text{НП}} \times \frac{\text{I}_{\text{ЕП}}}{\text{I}_{\text{ТП}}}$$

$$K_{\text{ІНТ}} = 1 \times \frac{2.34}{0.4} = 5.85$$

$K_{\text{ІНТ}} > 1.6$ показує те що розробка є дуже перспективною для її виведення на ринок.

4.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи на тему «Система автоматизації реагування на інциденти безпеки», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

4.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [31]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \times t_i}{T_p},$$

де: k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, Т р =21 день.

$З_о$ = 40000,00 · 5 / 5 = 40000,00 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	**Місячний посадовий** оклад, грн	**Оплата за робочий** день, грн	**Число днів** роботи	**Витрати на заробітну плату,** грн
Керівник проекту	40,000	2,000	5	10,000
Інженер-розробник програмного забезпечення	35,000	1,750	20	35,000
Інженер-розробник програмного забезпечення	35,000	1,750	20	35,000
Консультант	25,000	1,250	10	12,500
Всього				92,500

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \times t_i ,$$

де: C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду С і можна визначити за формулою:

$$C_i = \frac{M_M \times K_i \times K_c}{T_p \times t_{ЗМ}},$$

де: M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо M_M = 8000,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б);

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно Т р = 20 день;

$t_{ЗМ}$ – тривалість зміни, год.

C_i = 8000,00 · 1,10 · 1,65 / (20 · 8) = 90,75 грн.

$З_{р1}$ = 90,75 · 8,00 = 726 грн.

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	**Тривалість роботи,** год	**Розряд** роботи	**Тарифний** коефіцієнт	**Погодинна тарифна ставка,** грн	**Величина оплати на робітника** грн
Підготовка робочого місця дослідника	8	2	1,1	90.75	726
Інсталяція програмного забезпечення	3	3	1,35	111,38	334,14
Тестування	40	5	1,7	140,25	5610,00
Всього					6670,14

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \times \frac{Н_{дод}}{100\%},$$

де: $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод}$ = (95,000,00+ 6670,14) · 12 / 100% = 12200,42 грн.

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_д) \times \frac{Н_{дод}}{100\%},$$

де: $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

$З_н$ = (95,000,00 + 6670,14 + 12200,42) · 22 / 100% = 25051,52 грн.

4.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (М), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \times Ц_j \times K_j - \sum_{j=1}^{n} B_j \times Ц_{Bj} \ ,$$

де: H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, (К j = 1,1 … 1,15);

B_j – маса відходів j-го найменування, кг;

$Ц_{Bj}$ – вартість відходів j-го найменування, грн/кг.

М 1 = 3 · 190,00 · 1,1 - 0,000 · 0,00 = 570,0 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	**Ціна за** од	**Норма витрат,** од	**Величина відходів,** кг	**Ціна відходів**, грн/кг	**Вартість витраченого матеріалу,** грн
Офісний папір	190	3	0	0	570
Канцелярське приладдя (набір офісного	185	2	0	0	370

працівника)					
Flesh-пам’ять	140	2	0	0	280
Всього					1220

4.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі ($К_в$), які використовують при проведенні НДР на тему «Система автоматизації реагування на інциденти безпеки» відсутні.

4.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Витрати на «Спецустаткування для наукових (експериментальних) робіт» в роботі на тему «Система автоматизації реагування на інциденти безпеки» відсутні.

4.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$В_{прг} = \sum_{i=1}^{k} Ц_{іпрг} \times С_{прг \times i} \times К_i,$$

де: $Ц_{іпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$С_{прг \times i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (К і = 1,10…1,12);

k – кількість найменувань програмних засобів.

$В_{прг} = 7500{,}00 \cdot 2 \cdot 1{,}1 = 1500{,}00$ грн.

Отримані результати зведемо до таблиці:

Таблиця 4.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт.	Ціна за одиницю, грн	Вартість, грн
ОС Windows 11	2	7500	16500
Прикладний пакет Microsoft Office 2019	2	5900	12980
Комп'ютерна програма для розробки (WebStorm JetBrains IDEs)	2	4900	10780
Ubuntu Enterprise Edition	3	1200	3960
Всього			44220

4.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$А_{обл} = \frac{Ц_б}{Т_в} \times \frac{t_{вик}}{12},$$

де: $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$А_{обл} = (45000{,}00 \cdot 1) / (4 \cdot 12) = 937.50 \text{ грн.}$$

Таблиця 4.8 – Амортизаційні відрахування

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук Lenovo ThinkPad X1 Carbon	45000	2	1	1875
Сервер Dell PowerEdge R750	120000	2	1	5000.0
Стіл для робочого місця	18000	4	1	3750.0
Принтер HP LaserJet Pro	8000	2	1	333.3
Мультимедійний проектор Epson EB-X51	25000	2	1	1041.6
ОС Linux Enterprise	8000	2	1	333.3
Програмне забезпечення AutoCAD	15000	2	1	625.0
Мережевий комутатор Cisco Catalyst 2960	6000	2	1	250.0
Всього				9833.30

4.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (В е) розраховуємо за формулою:

$$В_с = \sum \frac{W_{yi} \times t_i \times Ц_е \times К_{впі}}{\eta_і},$$

де: W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо Ц е = 10,50 грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності;

$\eta_і$– коефіцієнт корисної дії обладнання.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук Lenovo ThinkPad X1 Carbon	0.45	126	595
Сервер Dell PowerEdge R750	1.2	150	1890
Принтер HP LaserJet Pro	0.15	60	94.5
Мультимедійний проектор Epson EB-X51	0.3	75	236.25
Мережевий комутатор Cisco Catalyst 2960	0.18	122	213
Всього			3028.72

4.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{св} = (З_о + З_р) \times \frac{Н_{св}}{100\%},$$

де: $Н_{св}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{св}$ = 20%.

$$В_{сп} = (95{,}000{,}00 + 6670{,}14) \cdot 20 / 100\% = 20334{,}03 \text{ грн.}$$

4.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{сп} = \left(З_о + З_р\right) \times \frac{Н_{сп}}{100\%},$$

де: $Н_{сп}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{сп}$ = 35%.

$В_{сп} =$ (95,000,00 + 6670,14) · 35 / 100% = 35584,55 грн.

4.2.11 Інші витрати

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = \left(З_о + З_р\right) \times \frac{Н_{ів}}{100\%},$$

де: $Н_{ів}$– норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в =$ (95000,00+ 6629,00) · 50 / 100% = 50814,50 грн.

4.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = \left(З_о + З_р\right) \times \frac{Н_{нзв}}{100\%},$$

де: $В_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $В_{нзв}$ = 120%.

$В_{нзв} =$ (95000,00+ 6629,00) · 120 / 100% = 121954,80 грн.

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{вод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{оьз} + В_е + В_{сб} + В_{сн} + І_с + В_{нзв}$$

$В_{заг}$ = 423402,56 грн.

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{В_{заг}}{\eta},$$

де: η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η = 0,8.

$$ЗВ = 423402,56 / 0.8 = 530503,20$$

4.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1-й рік – 10 користувачів/рік;

2-й рік – 40 користувачів/рік;

3-й рік – 100 користувачів/рік.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 100 користувачів;

$Ц_o$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 100000 грн /за рік користування;

$\pm\Delta Ц_o$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 20000,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора ΔП для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [31]:

$$\Delta П_i = (\pm\Delta Ц_o \times N \times Ц_o) \times \lambda \times \rho \times (1 - \frac{\vartheta}{100}),$$

де: λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а коефіцієнт λ = 0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ = 30%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році ϑ = 18%;

Збільшення чистого прибутку 1-го року:

$$\Delta П_1 = (100000 \times 10 \times 20000) \times 0.8333 \times 0.3 \times (1 - 0.18) = 4{,}099{,}836{,}00$$

Збільшення чистого прибутку 2-го року:

$$\Delta П_2 = (100000 \times 40 \times 20000) \times 0.8333 \times 0.3 \times (1 - 0.18) = 16{,}399{,}344{,}00$$

Збільшення чистого прибутку 3-го року:

$$\Delta П_3 = (100000 \times 100 \times 20000) \times 0.8333 \times 0.3 \times (1 - 0.18) = 40{,}998{,}360{,}00$$

Приведена вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+r)^t},$$

де: $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

r – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, r =0,25;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП = 4{,}099{,}836{,}00/(1-0.25)^{1} + 16{,}399{,}344{,}00/(1-0.25)^{2} + 40{,}998{,}360{,}00/(1-0.25)^{3} = 13{,}180{,}213{,}50$

Величина початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \times ЗВ,$$

де: $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадженнянауково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$= 4;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 530503,20 грн.

$PV = 4 \times 530503{,}20 = 2{,}122{,}012.80$

Абсолютний економічний ефект E_{abs} для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{abs} = ПП - PV,$$

де: ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 16211800,68 грн;

PV – теперішня вартість початкових інвестицій, 2,122,012.80 грн;

$$E_{abs} = \text{ПП} - PV = 13{,}180{,}213{,}50 - 2{,}122{,}012.80 = 11{,}058{,}200.70$$

Внутрішня економічна дохідність інвестицій $E_{\text{в}}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково- технічної розробки:

$$E_{\text{в}} = \sqrt[\text{Т}_\text{ж}]{1 + \frac{E_{abs}}{\text{PV}}} - 1,$$

де: E_{abs} – абсолютний економічний ефект вкладених інвестицій, 11,058,200.70 грн;

PV – теперішня вартість початкових інвестицій, 2,122,012.80 грн.

$\text{Т}_\text{ж}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_{\text{в}} = \sqrt[\text{Т}_\text{ж}]{1 + \frac{E_{abs}}{\text{PV}}} - 1 = (1 + 11{,}058{,}200.70 / 2{,}122{,}012.80)^{1/3} = 1{,}7$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій мін r_{min} :

$$r_{min} = d + f,$$

де: d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

r_{min} = 0,11+0,25 = 0,36 < 1,69 свідчить про те, що внутрішня економічна дохідність інвестицій $E_{\text{в}}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «» доцільно.

Період окупності інвестицій T_{ok} які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ok} = \frac{E_{\text{в}}}{1.69},$$

де: $E_{в}$ – внутрішня економічна дохідність вкладених інвестицій.

T_{ok} = 1 / 1,69 = 0,6 року.

$T_{ok} <$ 3-х років, що свідчить про комерційну привабливість науково- технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

4.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою « Система автоматизації реагування на інциденти безпеки» становить 35,6 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Також термін окупності становить 0,6 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Система автоматизації реагування на інциденти безпеки».

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи на тему «Система автоматизації реагування на інциденти безпеки» було досягнуто поставленої мети — розроблено систему автоматизації реагування на інциденти безпеки (SOAR-систему), яка відповідає сучасним вимогам кіберзахисту. У рамках роботи вирішено такі задачі:

- Проведено дослідження сучасних кіберзагроз, таких як фішинг, програми-вимагачі, атаки нульового дня та APT. Розглянуто підходи до їх виявлення і нейтралізації, включаючи використання SIEM та SOAR-систем.
- Досліджено існуючі SOAR-системи, такі як IBM QRadar SOAR, Splunk SOAR та Cortex XSOAR. визначено їх переваги (автоматизація, інтеграція, масштабованість) та недоліки (висока вартість, складність впровадження). Це дозволило сформулювати вимоги до розроблюваної системи.
- Сформульовано основні вимоги до системи: модульна архітектура, інтеграція з різними джерелами даних, захист від кібератак, підтримка плейбуків та можливість кастомізації. Концепція передбачає зниження залежності від ручного втручання шляхом автоматизації.
- Розроблено архітектуру системи, що складається з шарів: джерела даних, інтеграції, автоматизації, плейбуків, оркестрації, моніторингу та пост-інцидентного аналізу. Для кожного шару визначено функції, алгоритми роботи та механізми взаємодії.
- Прототип системи створено з використанням сучасних відкритих технологій. Реалізовано сценарії автоматизації реагування за допомогою плейбуків. Протестовано роботу системи на основі реальних кіберінцидентів, що показало скорочення часу реагування та підвищення точності.

– Система довела свою ефективність у зниженні часу обробки інцидентів і покращенні захищеності інформаційної інфраструктури. Визначено напрями подальшого розвитку, включаючи вдосконалення алгоритмів автоматизації, розширення інтеграцій та адаптацію під специфічні потреби організацій.

Результати роботи демонструють, що запропонована система здатна забезпечити швидке та узгоджене реагування на інциденти безпеки, мінімізуючи вплив людського фактора. Вона може бути інтегрована в існуючі інформаційні системи з метою підвищення рівня кіберзахисту організацій, зокрема тих, що мають обмежені ресурси.

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою « Система автоматизації реагування на інциденти безпеки» становить 35,6 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cisco talos 2023 year in review. URL: https://blog.talosintelligence.com/cisco-talos-2023-year-in-review/ (Last accessed: 08.11.2024).
2. Crowdstrike Global Threat Report 2024. URL: https://go.crowdstrike.com/rs/281-OBQ-266/images/GlobalThreatReport2024.pdf (Last accessed: 08.11.2024).
3. ENISA Threat Landscape 2024. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024 (Last accessed: 08.11.2024).
4. What Is SOAR? URL: https://paloaltonetworks.com/cyberpedia/what-is-soar (Last accessed: 05.11.2024).
5. What is incident response? URL: https://www.ibm.com/topics/incident-response (Last accessed: 05.11.2024).
6. Incident management for high-velocity teams. URL: https://www.atlassian.com/incident-management/incident-response/lifecycle#atlassian-incident-response-lifecycle (Last accessed: 05.11.2024).
7. Runbooks vs playbooks: A comprehensive overview. URL: https://cutover.com/blog/runbooks-vs-playbooks-comprehensive-overview (Last accessed: 03.11.2024).
8. Red Team vs Blue Team in Cybersecurity. URL: https://www.crowdstrike.com/en-us/cybersecurity-101/advisory-services/red-team-vs-blue-team/ (Last accessed: 03.11.2024).
9. What is threat intelligence. URL: https://www.ibm.com/topics/threat-intelligence (Last accessed: 03.11.2024).
10. IBM QRadar SOAR. URL: https://ibm.com/products/qradar-soar (Last accessed: 03.11.2024).
11. Splunk SOAR. URL: https://www.splunk.com/en_us/pdfs/resources/product-brief/splunk-soar.pdf (Last accessed: 03.11.2024).
12. Cortex XSOAR. URL: https://www.paloaltonetworks.com/cortex/cortex-xsoar (Last accessed: 03.11.2024).

13. Introduction to microservices. URL: https://cloud.google.com/architecture/microservices-architecture-introduction (Last accessed: 19.10.2024).
14. What is Microservices Architecture? Examples, Challenges, Benefits and Best Practices. URL: https://dev.to/hyscaler/what-is-microservices-architecture-examples-challenges-benefits-and-best-practices-10be (Last accessed: 10.10.2024).
15. Next.js vs. Gatsby: Comparing React Frameworks. URL: https://blog.logrocket.com/next-js-vs-gatsby-comparing-react-frameworks/ (Last accessed: 13.12.2024).
16. Comparison of Gatsby vs Next.js vs Nuxt.js. URL: https://www.gatsbyjs.com/features/jamstack/gatsby-vs-nextjs-vs-nuxtjs/ (Last accessed: 13.12.2024).
17. Gatsby vs Next vs Nuxt: Key Features and Differences. URL: https://thebcms.com/blog/gatsby-vs-next-vs-nuxt-key-features (Last accessed: 13.12.2024).
18. NestJS vs. Express.js. URL: https://blog.logrocket.com/nestjs-vs-express-js/ (Last accessed: 13.12.2024).
19. NestJS HTTP Adapters: Express vs Fastify. URL: https://www.javacodegeeks.com/2024/08/nestjs-http-adapters-express-vs-fastify.html (Last accessed: 13.12.2024).
20. Docker vs. CRI-O: Comparing Container Runtimes. URL: https://www.geeksforgeeks.org/docker-vs-cri-o/ (Last accessed: 13.12.2024).
21. Docker vs. containerd vs. CRI-O: An In-Depth Comparison. URL: https://phoenixnap.com/kb/docker-vs-containerd-vs-cri-o (Last accessed: 13.12.2024).
22. The Complete Podman vs Docker Analysis: Features, Performance & Security. URL: https://uptrace.dev/blog/podman-vs-docker.html (Last accessed: 13.12.2024).
23. Comparative Analysis: Pulsar vs RabbitMQ vs NATS. URL: https://streamnative.io/pulsar/pulsar-vs-rabbitmq-vs-nats (Last accessed: 13.12.2024).

24. Kafka vs. RabbitMQ vs. NATS: A Comparative Guide. URL: https://flarecompare.com/Cloud%20Orchestration/Kafka%20vs%20RabbitMQ%20vs%20NATS%20%20A%20Comparative%20Guide/ (Last accessed: 13.12.2024).
25. Adaptive and Scalable Database Management with Machine Learning Integration. URL:https://www.researchgate.net/publication/384131104_Adaptive_and_Scalable_Database_Management_with_Machine_Learning_Integration_A_PostgreSQL_Case_Study (Last accessed: 12.12.2024).
26. Postgresql Database Management System. URL: https://www.researchgate.net/publication/375138644_Postgresql_Database_Management_System_ODAK (Last accessed: 12.12.2024).
27. Next.js Documentation. URL: https://nextjs.org/docs (Last accessed: 12.12.2024).
28. NestJS Documentation. URL: https://docs.nestjs.com/ (Last accessed: 12.12.2024).
29. Docker Documentation. URL: https://docs.docker.com/ (Last accessed: 12.12.2024).
30. Apache Kafka Quickstart. URL: https://kafka.apache.org/quickstart (Last accessed: 12.12.2024).
31. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.
32. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

ДОДАТКИ

Додаток А

Акт перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Система автоматизації реагування на інциденти безпеки

Автор роботи: **Бугаєць Владислав Сергійович**

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ

Керівник Куперштейн Леонід Михайлович, к.т.н., доцент

Показники звіту подібності

Turnitin	
Оригінальність	**98 %**
Загальна схожість	**2 %**

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи.

Автор роботи _____ Владислав БУГАЄЦЬ
(підпис) (прізвище, ініціали)

Особа, відповідальна за перевірку _____ Валентина КАПЛУН
(підпис) (прізвище, ініціали)

Керівник роботи _____ _____
(підпис) (прізвище, ініціали)

Додаток Б

Текст програмної реалізації

Main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import * as cookieParser from 'cookie-parser';
import { ConfigService } from '@nestjs/config';
import { ValidationPipe } from '@nestjs/common';
import { AllExceptionsFilter } from './filtres/all-exceptions.filter';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(new ValidationPipe({ whitelist: true, transform: true }));
  app.useGlobalFilters(new AllExceptionsFilter());
  const configService = app.get(ConfigService);

  app.setGlobalPrefix('api/v1');

  app.use(cookieParser());

  app.enableCors({
    origin: configService.get<string>(
      'CORS_ORIGIN',
      'https://localhost:3000',
    ),
    credentials: true,
  });

  const port = configService.get<number>('PORT', 3500);
  await app.listen(port);

  console.log(`Application is running on: ${await app.getUrl()}`);
}

bootstrap();
```

Playbooks.service.ts

```
import {
  forwardRef,
  Inject,
  Injectable,
  NotFoundException,
} from '@nestjs/common';
import { Action } from './action.entity';
import { ALERT_REPOSITORY, PLAYBOOK_REPOSITORY } from 'src/constants';
import { Playbook } from './playbook.entity';
import { JiraService } from '../jira/jira.service';
import { ReputationCheckerService } from
'../reputationChecker/reputationChecker.service';
import { CreatePlaybookDto } from './dto/create-playbook.dto';
import { UpdatePlaybookDto } from './dto/update-playbook.dto';
import { CreateActionDto } from './dto/create-action.dto';
import { SplunkService } from '../splunk/splunk.service';
import { Alert } from '../splunk/alert.entity';
import * as fs from 'fs';
import * as path from 'path';
import { UpdateActionDto } from './dto/update-action.dto';

@Injectable()
export class PlaybookService {
  private readonly filePath = '../../../files/data.csv';

  constructor(
    @Inject('PLAYBOOK_REPOSITORY')
    private readonly playbookModel: typeof Playbook,
    @Inject('ACTION_REPOSITORY')
    private readonly actionModel: typeof Action,
    @Inject('ALERT_REPOSITORY')
    private readonly alertRepository: typeof Alert,
    private readonly jiraService: JiraService,
    @Inject(forwardRef(() => SplunkService))
    private readonly splunkService: SplunkService,
    private readonly reputationService: ReputationCheckerService,
  ) {}

  async createDefaultPlaybook(): Promise<Playbook> {
    const defaultPlaybookName = 'Default Playbook';
```

```
  // Перевіряємо, чи існує плейбук із дефолтною назвою
  let playbook = await this.playbookModel.findOne({
    where: { name: defaultPlaybookName },
    include: [Action],
    order: [[{ model: Action, as: 'actions' }, 'order', 'ASC']],
  });

  if (!playbook) {
    // Якщо плейбука немає, створюємо його
    playbook = await this.playbookModel.create({
      name: defaultPlaybookName,
      description: 'This is the default playbook for alerts',
    });

    const actionsData = [
      {
        type: 'putAlertInfo',
        parameters: {
          links: [],
          confluencePageIds: [],
        },
        order: 1,
      },
      {
        type: 'setIssueStatus',
        parameters: { status: 'open' },
        order: 2,
      },
    ];

    for (const actionData of actionsData) {
      const action = await this.actionModel.create(actionData);
      await playbook.$add('actions', action);
    }
  }

  return playbook;
}
// Метод створення плейбуку
async createPlaybook(
```

```
  createPlaybookDto: CreatePlaybookDto,
): Promise<Playbook> {
  const { actions, ...playbookData } = createPlaybookDto;
  const playbook = await this.playbookModel.create(playbookData);

  if (actions && actions.length > 0) {
    // Отримання тільки ідентифікаторів дій
    const actionIds = actions.map((action) => action.id).filter((id) => !!id);

    if (actionIds.length > 0) {
      const actionInstances = await this.actionModel.findAll({
        where: { id: actionIds },
      });
      await playbook.$set('actions', actionInstances);
    }
  }

  return playbook;
}

// Метод отримання плейбуку за ID
async findOne(id: number): Promise<Playbook> {
  const playbook = await this.playbookModel.findByPk(id, {
    include: { model: this.actionModel },
  });

  if (!playbook) {
    throw new NotFoundException('Плейбук не знайдено');
  }

  return playbook;
}

async findAll() {
  const playbooks = await this.playbookModel.findAll();

  if (!playbooks) {
    throw new NotFoundException('Плейбук не знайдено');
  }

  return playbooks;
```

```
}

// Метод оновлення плейбуку
async updatePlaybook(
  id: number,
  updatePlaybookDto: UpdatePlaybookDto,
): Promise<Playbook> {
  const playbook = await this.findOne(id);

  const { actions, ...playbookData } = updatePlaybookDto;

  await playbook.update(playbookData);

  if (actions) {
    // Отримання тільки ідентифікаторів дій
    const actionIds = actions.map((action) => action.id).filter((id) => !!id);

    if (actionIds.length > 0) {
      const actionInstances = await this.actionModel.findAll({
        where: { id: actionIds },
      });
      await playbook.$set('actions', actionInstances);
    }
  }

  return this.findOne(id);
}

// Метод видалення плейбуку
async removePlaybook(id: number): Promise<void> {
  const playbook = await this.findOne(id);
  await playbook.destroy();
}

// Додавання дії до плейбуку
async addActionToPlaybook(
  playbookId: number,
  createActionDto: CreateActionDto,
): Promise<void> {
  const playbook = await this.findOne(playbookId);
  const action = await this.actionModel.create(createActionDto);
```

```
  await playbook.$add('actions', action);
}

async updateAction(
  actionId: number,
  updateActionData: UpdateActionDto,
): Promise<Action> {
  const action = await this.actionModel.findByPk(actionId);

  if (!action) {
    throw new NotFoundException(`Action with id ${actionId} not found`);
  }

  // Оновлення даних
  await action.update(updateActionData);

  return action;
}

// Видалення дії з плейбуку
async removeActionFromPlaybook(
  playbookId: number,
  actionId: number,
): Promise<void> {
  const playbook = await this.findOne(playbookId);
  const action = await this.actionModel.findByPk(actionId);

  if (!action) {
    throw new NotFoundException('Дію не знайдено');
  }

  await playbook.$remove('actions', action);
}

async updateActionsOrder(
  playbookId: number,
  actionsOrder: { actionId: number; order: number }[],
): Promise<void> {
  const playbook = await this.findOne(playbookId);

  if (!playbook) {
```

```
    throw new NotFoundException(`Плейбук з ID ${playbookId} не знайдено`);
  }

  const transaction = await this.playbookModel.sequelize.transaction();
  try {
    for (const { actionId, order } of actionsOrder) {
      // Шукаємо зв'язок через PlaybookAction
      const playbookAction = await playbook.$get('actions', {
        where: { id: actionId },
        transaction,
      });

      if (!playbookAction.length) {
        throw new NotFoundException(
          `Дію з ID ${actionId} у плейбуку ${playbookId} не знайдено`,
        );
      }

      // Оновлюємо порядок
      await playbookAction[0].update({ order }, { transaction });
    }

    await transaction.commit();
  } catch (error) {
    await transaction.rollback();
    throw error;
  }
}

// Отримання всіх дій для плейбуку
async getActionsForPlaybook(playbookId: number): Promise<Action[]> {
  const playbook = await this.findOne(playbookId);
  return playbook.$get('actions');
}

async executePlaybook(issueId: number, initialContext: any = {}) {
  // Отримуємо алерт з Jira
  const issue = await this.jiraService.getIssueById(`${issueId}`);
  const summary = issue.summary;

  // Знаходимо або створюємо алерт у базі даних
```

```
  let alert = await this.alertRepository.findOne({
    where: { name: summary },
  });

  if (!alert) {
    alert = await this.alertRepository.create({
      name: summary,
      isAutomatedFormat: false,
      testID: '',
      externalLinks: [],
      confluenceFilesIds: [],
    });
  }

  // Знаходимо прив'язаний плейбук або створюємо дефолтний
  let playbook: Playbook;
  if (alert.playbookId) {
    playbook = await this.playbookModel.findByPk(alert.playbookId, {
      include: [Action],
      order: [[{ model: Action, as: 'actions' }, 'order', 'ASC']],
    });
  } else {
    playbook = await this.createDefaultPlaybook();
    alert.playbookId = playbook.id;
    await alert.save();
  }

  // Виконуємо дії плейбуку
  let context = { issueId, ...initialContext }; // Початковий контекст

  for (const action of playbook.actions) {
    context = await this.executeAction(action, context);

    console.log(context);
  }

  return context;
}

private async executeAction(action: Action, context: any) {
  switch (action.type) {
```

```
    case 'putAlertInfo':
      return await this.putAlertInfo(action.parameters, context);
    case 'setIssueStatus':
      return await this.setIssueStatus(action.parameters, context);
    case 'checkIsThisIp':
      return await this.checkIsSourceTypesOnlyIp(action.parameters, context);
    case 'checkKnownIp':
      return await this.checkIsExternalIpKnown(action.parameters, context);
    case 'checkIsSQLi':
      return await this.checkIsSQLi(action.parameters, context);
    case 'checkIsUAIp':
      return await this.checkIsUAip(action.parameters, context);
    case 'checkIsBadRegionIp':
      return await this.checkIsBadRegionIp(action.parameters, context);
    case 'checkIsExclusion':
      return await this.checkIpLoginSplunkExclusion(
        action.parameters,
        context,
      );
    default:
      throw new Error(`Невідомий тип дії: ${action.type}`);
  }
}

private async setIssueStatus(parameters: any, context: any) {
  console.log(parameters, context);

  const { status } = parameters;
  const { issueId, isExcluded, knwonIps, checkIsSourceTypesOnlyIp } = context;

  console.log(issueId);

  if (status === 'open') {
    await this.jiraService.setIssueStatusOpen(issueId);
    return {
      ...context,
      movedToOpen: true,
      movedToKnown: false,
      movedToProgress: false,
    };
  }
```

```
  if (
    (status === 'in progress' && isExcluded) ||
    (status === 'in progress' && knwonIps && checkIsSourceTypesOnlyIp)
  ) {
    console.log('moved to progress');

    await this.jiraService.moveInProgress(issueId);
    return {
      ...context,
      movedToKnown: false,
      movedToProgress: true,
      movedToOpen: false,
    };
  }

  if (
    (status === 'known issue' && isExcluded) ||
    (status === 'known issue' && knwonIps && checkIsSourceTypesOnlyIp)
  ) {
    console.log('moved to known');

    await this.jiraService.moveToKnownIssue(issueId);
    return {
      ...context,
      movedToKnown: true,
      movedToProgress: false,
      movedToOpen: false,
    };
  }

  return {
    ...context,
    movedToOpen: false,
    movedToKnown: false,
    movedToProgress: false,
  };
}

private async checkIsSourceTypesOnlyIp(params: any, context: any) {
  try {
```

```
const counts: { [key: string]: number } = {};

const { jsonAttachment, repCheck } = context;
const { iocs } = repCheck;

// Перевірка наявності та валідності `jsonAttachment`
if (
  !jsonAttachment ||
  !Array.isArray(jsonAttachment) ||
  jsonAttachment.length === 0
) {
  throw new Error('jsonAttachment is missing or invalid');
}

const data = JSON.stringify(jsonAttachment[0]);

// Перевірка наявності та валідності `iocs.externalIps`
if (!iocs || !iocs.externalIps || typeof iocs.externalIps !== 'object') {
  console.log(iocs);
  throw new Error('iocs.externalIps is missing or invalid');
}

const sourcetypes = [
  'cisco:firepower:syslog',
  'nginx:plus:kv',
  'cloudflare:http',
];

// Обчислення кількості IP
const extIpCount = Object.keys(iocs.externalIps).length;

if (extIpCount === 0) {
  console.warn('externalIps is empty.');
}

// Пошук `_sourcetype` у даних
for (const sourcetype of sourcetypes) {
  const escapedSourcetype = sourcetype.replace(
    /[.*+?^${}()|[\]\\]/g,
    '\\$&',
  );
```

```
    const regex = new RegExp(
      `"_sourcetype"\\s*:\\s*"${escapedSourcetype}"`,
      'g',
    );
    const matches = data.match(regex);
    const count = matches ? matches.length : 0;
    counts[sourcetype] = count;
  }

  console.log('Кількість знайдених `_sourcetype`:', counts);

  // Обчислення загальної кількості
  const totalCount = Object.values(counts).reduce(
    (sum, current) => sum + current,
    0,
  );

  if (totalCount === extIpCount) {
    console.log(
      'Загальна кількість збігів відповідає очікуваному значенню.',
    );
  } else {
    console.log(
      'Загальна кількість збігів не відповідає очікуваному значенню.',
    );
  }

  console.log({
    checkIsSourceTypesOnlyIp: {
      ...context,
      checkIsSourceTypesOnlyIp: true,
    },
  });

  return { ...context, checkIsSourceTypesOnlyIp: true };
} catch (error) {
  console.error('Error in checkIsSourceTypesOnlyIp:', error.message);
  return {
    ...context,
    checkIsSourceTypesOnlyIp: false,
    error: error.message,
```

```
    };
  }
}

private async putAlertInfo(parameters: any, context: any) {
  const {
    issueId,
    dataValues: { externalLinks, confluenceFilesIds },
  } = context;

  const jsonAttachment =
    await this.splunkService.parseAlertAttacment(issueId);

  console.log(jsonAttachment);

  const addConfluencePromises = confluenceFilesIds.map(
    async (confluencePageId) =>
      await this.splunkService.addConfluenceLink(issueId, confluencePageId),
  );

  const addExternalLinkPromises = externalLinks.map(
    async (link) => await this.splunkService.addExternalLink(issueId, link),
  );

  const repCheck = await this.splunkService.putIoCs(
    issueId,
    JSON.stringify(jsonAttachment),
  );

  await Promise.all([...addConfluencePromises, ...addExternalLinkPromises]);

  console.log({
    putAlertInfoContext: { ...context, jsonAttachment, repCheck },
  });

  return { ...context, jsonAttachment, repCheck };
}

private async checkIsExternalIpKnown(params: any, context: any) {
  const { repCheck } = context;
  const { iocs, reputation } = repCheck;
```

```
const externalIps = Object.keys(iocs.externalIps || {});

const filePath = path.join(__dirname, this.filePath);

if (!fs.existsSync(filePath)) {
  console.log(`Файл ${this.filePath} не знайдено.`);
  return { ...context, knwonIps: false };
}

// Перевірка IP у `reputation`
const result = {};

const unknownIps = []; // Для збереження невідомих IP

for (const ip of externalIps) {
  const entry = reputation.find((item) => item.ip === ip);
  if (entry) {
    const blockStatus =
      entry.details?.talos?.blockList?.status || 'NOT FOUND';
    result[ip] = blockStatus === 'ACTIVE';
  } else {
    result[ip] = false;
    unknownIps.push(ip); // Додати IP у список
  }
}

const allActive = Object.values(result).every((status) => status);

if (allActive) {
  console.log({
    checkIsExternalIpKnownContext: { ...context, knwonIps: false },
  });

  return { ...context, knwonIps: true };
}

return new Promise((resolve, reject) => {
  // Перевірка наявності файлу
  if (!fs.existsSync(filePath)) {
    console.log(`Файл ${this.filePath} не знайдено.`);
```

```
  console.log({
    checkIsExternalIpKnownContext: {
      ...context,
      knwonIps: false,
    },
  });

  return resolve({ ...context, knwonIps: false });
}
const result = {};

const foundIpsSet = new Set<string>();
const readStream = fs.createReadStream(filePath);

readStream
  .pipe(csv({ headers: false }))
  .on('data', (row) => {
    const ip1 = row[0]?.trim();
    const ip2 = row[1]?.trim();

    if (externalIps.includes(ip1)) {
      foundIpsSet.add(ip1);
    }

    if (externalIps.includes(ip2)) {
      foundIpsSet.add(ip2);
    }
  })
  .on('end', () => {
    const notFoundIps = externalIps.filter(
      (ip) => !foundIpsSet.has(ip) && !result[ip],
    );

    const foundIps = Array.from(foundIpsSet);
    if (foundIps.length == externalIps.length) {
      console.log(foundIps.length, externalIps.length);
      console.log(`Знайдені IP-адреси: ${foundIps.join(', ')}`);
      console.log({
        checkIsExternalIpKnownContext: {
          ...context,
          knwonIps: false,
```

```
            firstSeen: notFoundIps,
          },
        });

        resolve({ ...context, knwonIps: true, firstSeen: [] });
      } else if (foundIps.length > 0) {
        console.log(`Знайдені IP-адреси: ${foundIps.join(', ')}`);
        console.log({
          checkIsExternalIpKnownContext: {
            ...context,
            knwonIps: false,
            firstSeen: notFoundIps,
          },
        });

        resolve({ ...context, knwonIps: false, firstSeen: notFoundIps });
      } else {
        console.log('Жодна з IP-адрес не знайдена.');
        console.log({
          checkIsExternalIpKnownContext: {
            ...context,
            knwonIps: false,
            firstSeen: notFoundIps,
          },
        });

        resolve({ ...context, knwonIps: false, firstSeen: notFoundIps });
      }
    })
    .on('error', (error) => {
      console.log('Помилка при читанні CSV-файлу:', error.message);
      reject(error);
    });
  });
}

private async checkIsSQLi(params: any, context: any) {
  console.log({
    checkIsSQLiContext: {
      ...context,
      knwonIps: false,
```

```
      },
    });
    return { ...context, checkIsSQLi: false };
  }

  private async checkIsUAip(params, context: any) {
    console.log({
      isUrkaineIpContext: {
        ...context,
        isUrkaineIp: false,
      },
    });
    return {
      ...context,
      isUrkaineIp: false,
      blockSendStatus: {
        success: true,
        response: 'network-object host block success',
      },
    };
  }

  private async checkIsBadRegionIp(params, context: any) {
    console.log({
      checkIsBadRegionIpContext: {
        ...context,
        isBadIp: false,
      },
    });
    return { ...context, isBadIp: false };
  }

  private async checkIpLoginSplunkExclusion(params: any, context: any) {
    const {
      repCheck: { iocs },
    } = context;
    // Визначаємо списки виключень
    const excludedIps = ['195.162.88.161', '176.36.134.14', '192.168.0.1']; // Список
виключених IP
    const excludedUsernames = [
      'buhaiets.v',
```

```
      'shevchenko.dm2',
      'shysholik.m',
      'poligenko.o@novaposhta.ua',
    ]; // Список виключених імен користувачів

    // Отримуємо IP-адреси та імена користувачів із params
    const externalIps = Object.keys(iocs.externalIps || {});
    const usernames = Object.keys(iocs.usernames || {});

    // Перевіряємо, чи є виключення для IP
    const ipExclusions = externalIps.filter((ip) => excludedIps.includes(ip));

    // Перевіряємо, чи є виключення для імен користувачів
    const usernameExclusions = usernames.filter((username) =>
      excludedUsernames.includes(username),
    );

    // Логіка виключення
    if (ipExclusions.length > 0 && usernameExclusions.length > 0) {
      console.log(
        `Знайдено виключення для IP: ${ipExclusions}, Usernames:
${usernameExclusions}`,
      );
      return {
        ...context,
        isExcluded: true,
        excludedIps: ipExclusions,
        excludedUsernames: usernameExclusions,
      };
    }

    // Якщо виключень немає, виконуємо стандартну логіку
    console.log('Виключень не знайдено. Продовження стандартної логіки...');
    return {
      isExcluded: false,
    };
  }

  async setExcludedSplunkLogin(context: any) {
    const { issueId, isExcluded } = context;
    if (isExcluded) {
```

```
      // await this.jiraService.moveToKnownIssue(issueId);
      return { movedToKnown: true };
    }

    return { movedToKnown: false };
  }
}
```

Auth.service.ts

```
import {
  Injectable,
  UnauthorizedException,
  InternalServerErrorException,
  ConflictException,
} from '@nestjs/common';
import { UsersService } from '../users/users.service';
import { RolesService } from '../roles/roles.service';
import { V4 } from 'paseto';
import { addHours } from 'date-fns';
import { UserDto } from '../users/dto/user.dto';
import * as argon2 from 'argon2';

@Injectable()
export class AuthService {
  private privateKey: string;
  private publicKey: string;

  constructor(
    private readonly usersService: UsersService,
    private readonly rolesService: RolesService,
  ) {
    this.initializeKeys();
  }

  private async initializeKeys(): Promise<void> {
    try {
      const keyPair = await V4.generateKey('public', { format: 'paserk' });
      this.privateKey = keyPair.secretKey;
      this.publicKey = keyPair.publicKey;
      console.log('Keys initialized successfully');
    } catch (err) {
```

```
    console.error('Error generating keys:', err);
    throw new InternalServerErrorException('Failed to generate keys');
  }
}

private async ensureKeysInitialized(): Promise<void> {
  if (!this.privateKey || !this.publicKey) {
    console.error('Keys are not initialized');
    await this.initializeKeys();
  }
}

private async generateToken(user: any): Promise<string> {
  const now = new Date();
  const expirationDate = addHours(now, 24);

  const payload = {
    sub: user.id.toString(),
    roleId: user.roleId,
    iat: now.toISOString(),
    exp: expirationDate.toISOString(),
    aud: 'api.soar.novasoc.space',
    iss: 'auth.soar.novasoc.space',
  };

  try {
    const token = await V4.sign(payload, this.privateKey, {
      audience: 'api.soar.novasoc.space',
      issuer: 'auth.soar.novasoc.space',
    });

    console.log('Token signed successfully:', token);
    return token;
  } catch (err) {
    console.error('Error signing token:', err);
    throw new InternalServerErrorException('Failed to sign token');
  }
}

async register(userDto: UserDto): Promise<string> {
  const existingUser = await this.usersService.findOneByEmail(userDto.email);
```

```
    if (existingUser) {
      throw new ConflictException('Email is already taken');
    }

    const roleName = userDto.role || 'user'; // Використовуємо роль за замовчуванням,
якщо не вказана
    const role = await this.rolesService.findByName(roleName);

    if (!role) {
      throw new ConflictException(`Role ${roleName} does not exist`);
    }

    const user = await this.usersService.create(userDto, role);

    // Після створення користувача генеруємо токен
    return this.generateToken(user);
  }

  async login(login: string, password: string): Promise<string> {
    await this.ensureKeysInitialized();

    const user = await this.usersService.findOneByEmail(login);
    if (
      !user ||
      !(await this.usersService.verifyPassword(password, user.password))
    ) {
      throw new UnauthorizedException('Invalid credentials');
    }

    return this.generateToken(user);
  }

  async verify(token: string): Promise<any> {
    await this.ensureKeysInitialized();

    try {
      const payload = await V4.verify(token, this.publicKey, {
        audience: 'api.soar.novasoc.space',
        issuer: 'auth.soar.novasoc.space',
      });
```

```
    console.log('Token verified successfully:', payload);
    return payload;
  } catch (err) {
    console.error('Error verifying token:', err);
    throw new UnauthorizedException('Invalid token');
  }
}

async changePassword(
  id: number,
  currentPassword: string,
  newPassword: string,
): Promise<void> {
  const user = await this.usersService.findOneById(id);

  if (!user) {
    throw new UnauthorizedException('User not found');
  }

  const isPasswordValid = await this.usersService.verifyPassword(
    currentPassword,
    user.password,
  );
  if (!isPasswordValid) {
    throw new UnauthorizedException('Current password is incorrect');
  }

  const hashedNewPassword = await argon2.hash(newPassword);

  user.password = hashedNewPassword;
  await user.save();

  console.log('Password changed successfully');
}

async resetPassword(login: string) {
  const user = await this.usersService.findOneByEmail(login);
  const newPassword = 'upadeteME12@';
  if (!user) {
    throw new UnauthorizedException('User not found');
```

```
    }

    const hashedNewPassword = await argon2.hash(newPassword);

    user.password = hashedNewPassword;
    await user.save();

    console.log('Password reseted successfully');
  }
}
```

User.service.ts

```
import { Injectable, Inject } from '@nestjs/common';
import { User } from './user.entity';
import { UserDto } from './dto/user.dto';
import { USER_REPOSITORY } from '../../constants';
import * as argon2 from 'argon2';
import { Role } from '../roles/roles.entity';

@Injectable()
export class UsersService {
  constructor(
    @Inject(USER_REPOSITORY) private readonly userRepository: typeof User,
  ) {}

  async create(user: UserDto, role: Role): Promise<User> {
    const hashedPassword = await argon2.hash(user.password);
    return await this.userRepository.create<User>({
      ...user,
      password: hashedPassword,
      role: role,
      roleId: role.id,
    });
  }

  async findOneByEmail(email: string): Promise<User> {
    return await this.userRepository.findOne<User>({ where: { email } });
  }

  async findOneById(id: number): Promise<User> {
```

```
    return await this.userRepository.findOne<User>({ where: { id } });
  }

  async verifyPassword(
    plainPassword: string,
    hashedPassword: string,
  ): Promise<boolean> {
    return await argon2.verify(hashedPassword, plainPassword);
  }
  async findAllEmails(): Promise<string[]> {
    const users = await this.userRepository.findAll<User>({
      attributes: ['email'], // Витягуємо тільки поле email
    });
    return users.map((user) => user.email); // Повертаємо масив email-адрес
  }
}
```

login/page.tsx

```
"use client";

import { useState } from 'react';
import styled, { keyframes } from 'styled-components';
import axiosInstance from '../utils/axiosInstance';
import { useRouter } from 'next/navigation';

const Container = styled.div`
  display: flex;
  height: 100vh;
`;

const FormWrapper = styled.div`
  width: 35%;
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #1a1a1a;
`;

const MatrixTextAnimation = keyframes`
  0% { transform: translateY(0); }
```

```
  100% { transform: translateY(-200%); }
`;

const Background = styled.div`
  position: relative;
  width: 65%;
  overflow: hidden;
  background-color: black;
  display: flex;
  justify-content: center;
  align-items: center;
`;

const BlurredOverlay = styled.div`
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.5);
  backdrop-filter: blur(2px);
  z-index: 1;
  display: flex;
  justify-content: center;
  align-items: center;
`;

const NovaSoarText = styled.h1`
  font-size: 4rem;
  color: #00ff41;
  z-index: 2;
  text-shadow: 0 0 10px #00ff41, 0 0 20px #00ff41;
`;

const MatrixText = styled.div`
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 400%;  // Збільшена висота для безперервної анімації
  font-family: 'Inconsolata', monospace;
```

```
  font-size: 2rem;
  color: #00ff41;
  opacity: 0.4;
  animation: ${MatrixTextAnimation} 100s ease-in-out infinite; // Збільшена тривалість і
плавніша анімація
  z-index: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
`;

const LoginForm = styled.form`
  width: 100%;
  max-width: 400px;
  padding: 40px;
  background-color: rgba(0, 0, 0, 0.9);
  border-radius: 10px;
  box-shadow: 0 0 10px #00ff41;
`;

const Title = styled.h1`
  margin-bottom: 20px;
  font-size: 2rem;
  color: #00ff41;
  text-align: center;
`;

const Input = styled.input`
  display: block;
  width: 100%;
  padding: 10px;
  margin-bottom: 20px;
  font-size: 1rem;
  border: 2px solid #00ff41;
  border-radius: 5px;
  background-color: transparent;
  color: #00ff41;
  outline: none;

  &:focus {
    box-shadow: 0 0 10px #00ff41;
```

```
  }
`;

const Button = styled.button`
  display: block;
  width: 100%;
  padding: 10px;
  font-size: 1rem;
  color: black;
  background-color: #00ff41;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  text-transform: uppercase;
  transition: background-color 0.3s ease;

  &:hover {
    background-color: #00cc33;
  }
`;

export default function LoginPage() {
    const [login, setLogin] = useState('');
    const [password, setPassword] = useState('');
    const router = useRouter();

    const handleLogin = async (event: React.FormEvent) => {
      event.preventDefault();
      try {
        const response = await axiosInstance.post('/auth/login', {
            login,
            password,
        }, {
            withCredentials: true, // Додаємо цей параметр для відправки cookies
        });

        router.push('/dashboard');

      } catch (error) {
        console.error('Login failed:', error);
      }
```

```
    };

    return (
      <Container>
        <Background>
          <MatrixText>
            <p>{'01101100 01101111 01100111 01101001 01101110'.repeat(2000)}</p>
          </MatrixText>
          <BlurredOverlay>
            <NovaSoarText>NovaSoar</NovaSoarText>
          </BlurredOverlay>
        </Background>
        <FormWrapper>
          <LoginForm onSubmit={handleLogin}>
            <Title>Login</Title>
            <Input
              type="text"
              placeholder="Username"
              value={login}
              onChange={(e) => setLogin(e.target.value)}
            />
            <Input
              type="password"
              placeholder="Password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
            />
            <Button type="submit">Enter</Button>
          </LoginForm>
        </FormWrapper>
      </Container>
    );
  }
```

project/page.tsx

```
"use client";
import React, { useState, useEffect } from "react";
import { useParams } from "next/navigation";
import styled from "styled-components";
import Header from "@/app/UI/Header";
```

```
import Sidebar from "@/app/UI/Sidebar";
import ContentArea from "@/app/UI/ContentArea";
import PlaybooksPanel from "../../../../modules/PlayBookPanel/PlaybooksPanel";
import { useAlert } from "../../../../../hooks/useAlert";
import { useConfluenceFiles } from "../../../../../hooks/useConfluence";
import AlertsManagement from "../../../../modules/SplunkAlertController/index";
import { usePlaybooks } from "@/app/hooks/usePlaybooks";
import axiosInstance from "@/app/utils/axiosInstance";
import { AxiosError } from "axios";

const Wrapper = styled.div`
  display: flex;
  flex-direction: row;
  align-items: stretch;
  justify-content: space-between;
  background-color: #1e1e1e;
  width: 100%;
  height: 100vh;
  overflow: hidden;
  color: #f0f0f0;
`;

const MainContent = styled.div`
  flex-direction: column;
  display: flex;
  flex-wrap: wrap;
  background-color: #1e1e1e;

  @media (max-width: 768px) {
    flex-direction: column;
  }
`;

const TabContainer = styled.div`
  width: 100vw;
  display: flex;
  gap: 10px;
  padding: 10px;
`;

const TabButton = styled.button<{ isActive: boolean }>`
```

```
  background-color: ${({ isActive }) => (isActive ? "#333" : "#444")};
  color: #f0f0f0;
  border: none;

  width: 100%;
  padding: 10px 20px;
  cursor: pointer;
  &:hover {
    background-color: #555;
  }
`;
const ContentContainer = styled.div`
  display: flex;
  justify-content: center;
  align-items: center;
  width: 100%;
`;

const Page: React.FC = () => {
  const { alertId } = useParams();
  const singleAlertId = Array.isArray(alertId) ? alertId[0] : alertId;

  const numericAlertId = singleAlertId ? parseInt(singleAlertId, 10) : null;

  const [selectedAlertKey, setSelectedAlertKey] = useState<string | null>(null);
  const [activeTab, setActiveTab] = useState<"alert" | "playbook">("alert");

  const {
    alert,
    loading: alertLoading,
    error: alertError,
  } = useAlert(numericAlertId);

  const {
    confluenceFiles: allConfluenceFiles,
    loading: confluenceLoading,
    error: confluenceError,
  } = useConfluenceFiles();

  const {
    playbooks: allPlaybooks,
```

```
  loading: playbooksLoading,
  error: playbooksError,
} = usePlaybooks();

const [fetchAttachment, setFetchAttachment] = useState<boolean>(false);
const [externalLinks, setExternalLinks] = useState<string[]>([]);
const [selectedPlaybook, setSelectedPlaybook] = useState<any>();
const [selectedConfluenceFiles, setSelectedConfluenceFiles] = useState<any[]>(
  []
);

useEffect(() => {
  if (alert) {
    setFetchAttachment(alert.isAutomatedFormat);
    setExternalLinks(alert.externalLinks);

    const selectedFiles = allConfluenceFiles.filter((file) =>
      alert.confluenceFilesIds.includes(file.id.toString())
    );
    setSelectedConfluenceFiles(selectedFiles);
  }

  if (alert && alert.playbookId && allPlaybooks.length > 0) {
    const activePlaybook = allPlaybooks.find(
      (playbook) => playbook.id === alert.playbookId
    );

    if (activePlaybook) {
      setSelectedPlaybook({
        name: activePlaybook.name,
        id: activePlaybook.id,
      });
    }
  }
}, [alert, allConfluenceFiles, allPlaybooks]);

if (numericAlertId === null) {
  return <p style={{ color: "red" }}>Невірний ID алерту.</p>;
}

if (alertLoading || playbooksLoading || confluenceLoading || !alert) {
```

```
  return <p>Завантаження алерту...</p>;
}

if (alertError || playbooksError || confluenceError) {
  return (
    <p style={{ color: "red" }}>
      {alertError || playbooksError || confluenceError}
    </p>
  );
}

const updateAlertInfo = async () => {
  try {
    const url = `splunk/alerts/${numericAlertId}`;

    const payload = {
      data: {
        isAutomatedFormat: fetchAttachment,
        name: alert.name,
        externalLinks: externalLinks,
        confluenceFilesIds: selectedConfluenceFiles.map((item) => item.id),
        playbookId: selectedPlaybook?.id,
      },
    };

    const response = await axiosInstance.put(url, payload);

    console.log("Alert information updated successfully:", response.data);

    return response.data;
  } catch (error: any) {
    if (error.response) {
      console.error("Server error:", {
        status: error.response.status,
        data: error.response.data,
      });
    } else if (error.request) {
      console.error("No response received from the server:", error.request);
    } else {
      console.error("Error setting up the request:", error.message);
    }
```

```
    throw new Error(
      "Failed to update alert information. Please try again later."
    );
  }
};

return (
  <ContentArea>
    <Header />
    <Sidebar />
    <Wrapper>
      <MainContent>
        <TabContainer>
          <TabButton
            isActive={activeTab === "alert"}
            onClick={() => setActiveTab("alert")}
          >
            Управління Алертом
          </TabButton>
          <TabButton
            isActive={activeTab === "playbook"}
            onClick={() => setActiveTab("playbook")}
          >
            Управління Playbookом Алерту
          </TabButton>
        </TabContainer>
        <ContentContainer>
          {activeTab === "alert" && (
            <AlertsManagement
              alert={alert}
              selectedAlertKey={selectedAlertKey}
              setSelectedAlertKey={setSelectedAlertKey}
              numericAlertId={numericAlertId}
              fetchAttachment={fetchAttachment}
              setFetchAttachment={setFetchAttachment}
              externalLinks={externalLinks}
              setExternalLinks={setExternalLinks}
              selectedPlaybook={selectedPlaybook}
              setSelectedPlaybook={setSelectedPlaybook}
              selectedConfluenceFiles={selectedConfluenceFiles}
```

```
                setSelectedConfluenceFiles={setSelectedConfluenceFiles}
                updateAlertInfo={updateAlertInfo}
                allPlaybooks={allPlaybooks}
                allConfluenceFiels={allConfluenceFiles}
              />
            )}
            {activeTab === "playbook" && <PlaybooksPanel />}
          </ContentContainer>
        </MainContent>
      </Wrapper>
    </ContentArea>
  );
};

export default Page;
```

setting/page.tsx

```
'use client';

import styled from 'styled-components';
import { useContext, useState } from 'react';
import Header from '../UI/Header';
import Sidebar from '../UI/Sidebar';
import axiosInstance from '../utils/axiosInstance';
import Alert from '../UI/Alert';
import { SidebarContext } from '../context';

const SettingsContainer = styled.div`
  display: flex;
  position: relative;
  height: 100vh;
  background-color: black;
  overflow-x: hidden;
`;

const ContentArea = styled.div<{ $isOpen: boolean }>`
  margin-left: ${({ $isOpen }) => ($isOpen ? '250px' : '0')};
  margin-top: 60px;
  padding: 20px;
  flex: 1;
  color: #00ff41;
```

```
  transition: margin-left 0.3s ease;
`;

const MainContent = styled.div`
  padding: 20px;
  color: #00ff41;
`;

const Form = styled.form`
  display: flex;
  flex-direction: column;
  max-width: 400px;
  margin-bottom: 40px;
`;

const Label = styled.label`
  margin-bottom: 10px;
  font-size: 1rem;
  color: #00ff41;
`;

const Input = styled.input`
  margin-bottom: 20px;
  padding: 10px;
  font-size: 1rem;
  color: black;
  border-radius: 5px;
  border: none;
`;

const Button = styled.button`
  padding: 10px;
  font-size: 1rem;
  color: black;
  background-color: #00ff41;
  border: none;
  border-radius: 5px;
  cursor: pointer;

  &:hover {
    background-color: #00cc33;
```

```
  }
`;

export default function UsersManagmentPage() {
  const [alertMessage, setAlertMessage] = useState<string | null>(null);

  const [currentPassword, setCurrentPassword] = useState('');
  const [newPassword, setNewPassword] = useState('');

  const sidebarContext = useContext(SidebarContext);

  // Перевірка, чи контекст визначений
  if (!sidebarContext) {
    throw new Error('SidebarContext must be used within a SidebarProvider');
  }

  const { isSidebarOpen } = sidebarContext;

  const handlePasswordChange = async (e: React.FormEvent) => {
    e.preventDefault();
    try {
      const response = await axiosInstance.patch('/auth/change-password', {
        currentPassword,
        newPassword,
      });
      setAlertMessage('Password was changed successfully');
      setCurrentPassword(''); // Очищення поля після успішного оновлення
      setNewPassword(''); // Очищення поля після успішного оновлення
    } catch (error) {
      console.error('Error changing password:', error);
      setAlertMessage('Error changing password');
    }
  };


  return (
    <SettingsContainer>
      <Header />
      <Sidebar />
      <ContentArea $isOpen={isSidebarOpen}>
        <MainContent>
```

```
          <h2>User Settings</h2>

          <h3>Change Password</h3>
          <Form onSubmit={handlePasswordChange}>
            <Label htmlFor="currentPassword">Current Password</Label>
            <Input
              id="currentPassword"
              type="password"
              value={currentPassword}
              onChange={(e) => setCurrentPassword(e.target.value)}
              required
            />

            <Label htmlFor="newPassword">New Password</Label>
            <Input
              id="newPassword"
              type="password"
              value={newPassword}
              onChange={(e) => setNewPassword(e.target.value)}
              required
            />

            <Button type="submit">Change Password</Button>
          </Form>
        </MainContent>
      </ContentArea>
      {alertMessage && <Alert message={alertMessage} />}
    </SettingsContainer>
  );
}
```

Додаток В

Порівняльні таблиці характеристик технологій

Таблиця В.1 – Порівняння показників рендерингу та інтерактивності

Продуктивність			
Параметр	Next.js	Gatsby	Nuxt.js
Час до повної інтерактивності	~1.2 с	~1.4 с	~1.3 с
FCP (First Contentful Paint)	~0.7 с	~0.8 с	~0.9 с
Підтримка SSR	Так	Статичний рендеринг	Так (SSR для Vue)
Масштабованость			
Фреймворк	Тест-навантаження (1000 одночасних користувачів)		
Next.js	Мінімальні просідання продуктивності, легка підтримка 2-3 тис. підключень за умов оркестрації		
Gatsby	Придатний для статичних сторінок, для динамічних сценаріїв потрібні CDN та додаткові налаштування		
Nuxt.js	Схожі показники до Next.js, але менша екосистема призводить до ускладнень в особливо високонавантажених сценаріях		
Навантаження			
Фреймворк	Середнє споживання пам’яті (на 1000 підкл.)	Оптимізація кешування та Code Splitting	Загальна ефективність використання ресурсів
Next.js	~200 МБ	SSR + автоматичний Code Splitting	~12–24
Gatsby	~220 МБ	Статичний контент, CDN	SSR, Code Splitting, але менше оптимізацій
Nuxt.js	Висока	Висока (для статичного контенту)	Нижча через відсутність SSR і більший bundle
Порівняння безпеки			
Фреймворк	Час реакції на відомі вразливості (умовно, з моменту виявлення)	Частота оновлень безпеки (патч-релізи)	Підтримка з боку ком’юніті та вендора при виявленні вразливостей
Next.js	1-2 дні для критичних, регулярні патчі	Часті оновлення, офіційна підтримка та швидкий вихід патчів	Дуже активна, швидкі issue у GitHub та швидкі PR з виправленнями
Gatsby	Зазвичай 2-5 днів, залежно від критичності	Постійно підтримується, але інколи запізнення у патчах	Активна, але менша база контриб’юторів
Nuxt.js	Схожі показники до Gatsby, іноді повільніше через меншу спільноту	Патчі виходять, але через меншу базу користувачів вони можуть бути не такими оперативними	Менша спільнота — інколи довше чекати на Pull Request

Таблиця В.2 Порівняння фреймворків для клієнтьского інтерфейсу.

Продуктивність			
Фреймворк	RPS (середнє)	Середня затримка (мс)	Пам’ять (MB) при 1000 з’єднаннях
Nest.js	~9 500–10 000	~12–15	~250
Express.js	~8 000–9 000	~18–22	~270
Fastify	~11 000	~10–12	~240
Масштабованость			
Інструмент	Легкість інтеграції з Kubernetes	Наявність офіційних образів	Інтеграція з CI/CD інструментами
Фреймворк	Підтримка мікросервісів	Легкість інтеграції з Docker/K8s	Готові патерни міжсервісної комунікації
Nest.js	Вбудований модуль мікросервісів (TCP, gRPC, NATS)	Легка інтеграція, офіційні гайди	Так, готові драйвери та адаптери
Fastify	Вбудований модуль мікросервісів (TCP, gRPC, NATS)	Легка інтеграція, офіційні гайди	Немає готових драйверів та адаптерів
Express.js	Потрібні сторонні бібліотеки, ручна реалізація	Легко контейнеризувати, але менше офіційних прикладів	Менше готових рішень, все налаштовується вручну
Порівняння екосистеми та підтримки СУБД			
Фреймворк	Кількість пакетів NPM з ключовим словом	GitHub Stars (на гол. репоз.)	Середній час відповіді на Issues (години)
Nest.js	~10 000+	~45k	~12–24
Express.js	~50 000+	~60k	~6–12 (дуже велика спільнота)
Fastify	~5 000+	~26k	~24–48
Порівняння безпеки			
Фреймворк	Час реакції на критичні вразливості (дні)	Частота випуску безпекових патчів	Готові інструменти для безпеки з “коробки”
Nest.js	1–3 дні	Регулярні оновлення	Так (модулі автентифікації, шифрування, інтеграція з Vault)
Express.js	2–5 днів (залежить від контриб’юторів)	Достатньо часті оновлення	Ні, потрібно інтегрувати сторонні модулі
Fastify	3–7 днів (менша спільнота)	Менше регулярних патчів	Обмежено, в основному через плагіни

Таблиця В.3 – Порівняння рішень для контейнеризації

Продуктивність			
Інструмент	Час виконання CPU-тесту (умовні дані, с)	Використання пам'яті при 10 контейнерах (MB)	Накладні витрати на запуск контейнера
Docker	~30 с	~600 MB	Мінімальні, швидкий старт
Podman	~32 с	~580 MB	Мінімальні, аналогічні до Docker
CRI-O	~33 с	~590 MB	Мінімальні, трохи повільніший старт
Масштабованость			
Інструмент	Легкість інтеграції з Kubernetes	Наявність офіційних образів	Інтеграція з CI/CD інструментами
Docker	Пряма інтеграція з Docker Hub та Helm charts	Велика кількість офіційних образів	Дуже широка підтримка (Jenkins, GitLab CI)
Podman	Сумісність із Kubernetes API, але менше офіційних гайдів	Підтримка OCI-образів, менше офіційних готових рішень	Інтегрується, але менше прикладів ніж Docker
CRI-O	Створений для K8s, нативна підтримка	Використовує образи в OCI-форматі, менше готових образів	Інтеграція з CI/CD можлива, але менше документована
Порівняння екосистеми та підтримки СУБД			
Іструмент	Кількість репозиторіїв з Dockerfile/Containerfile	Документація та гайди	Час відповіді спільноти на Issues
Docker	Мільйони Dockerfile у GitHub	Дуже детальна, багато прикладів	~6–12 годин відповіді на популярних форумах
Podman	Менше порівняно з Dockerfile, але росте	Гарна документація, але менше прикладів	~12–24 години
CRI-O	Менше публічних репозиторіїв	Документація в основному орієнтована на Kubernetes-адміністраторів	~24–48 годин
Порівняння безпеки			
Інструмент	Швидкість реагування на вразливості	Інструменти сканування образів	Офіційна підтримка безпеки
Docker	1–3 дні для критичних вразливостей	Docker Scan (вбудований), інтеграція з Trivy	Так, офіційні патчі та CVE-рекомендації
Podman	2–5 днів	Використання сторонніх сканерів (Trivy)	Спільнота активно реагує, офіційна команда Red Hat
CRI-O	2–4 дні	Сторонні інструменти (Trivy, Clair)	Підтримка з боку Kubernetes-ком'юніті

Таблиця В.4 – Порівняння рішень для управління подіями.

Продуктивність			
Платформа	Горизонтальне масштабування	Інструменти для K8s інтеграції	Реплікація та відмовостійкість
Kafka	Легко додаються брокери, шардінг топіків	Operators (Strimzi Operator)	Реплікація з настройками фактору
RabbitMQ	Масштабування можливе, але складніше	Менше офіційних операторів	Можлива кластеризація, але менш прозора
Pulsar	Вбудована підтримка шардінгу та сегментації	Helm Charts та оператори є	Висока відмовостійкість, гнучкі механізми зберігання
NATS	Легко масштабувати, але менш складні сценарії	Сторонні оператори, Helm Charts	Мінімальна відмовостійкість, фокус на простоті
Масштабованість			
Платформа	Горизонтальне масштабування	Інструменти для K8s інтеграції	Реплікація та відмовостійкість
Kafka	Легко додаються брокери, шардінг топіків	Operators (Strimzi Operator)	Реплікація з настройками фактору
RabbitMQ	Масштабування можливе, але складніше	Менше офіційних операторів	Можлива кластеризація, але менш прозора
Pulsar	Вбудована підтримка шардінгу та сегментації	Helm Charts та оператори є	Висока відмовостійкість, гнучкі механізми зберігання
NATS	Легко масштабувати, але менш складні сценарії	Сторонні оператори, Helm Charts	Мінімальна відмовостійкість, фокус на простоті
Порівняння екосистеми та підтримки СУБД			
Платформа	Кількість готових конекторів (Kafka Connect, Pulsar IO)	Документація та приклади	Час відповіді спільноти
Kafka	Сотні офіційних і сторонніх конекторів	Дуже детальна, безліч гайдів	~6–12 годин
RabbitMQ	Безліч клієнтів для різних мов, але менше стрім-конекторів	Дуже відома, проста	~6–12 годин
Pulsar	Менше конекторів, але асортимент росте	Документація хороша, менше прикладів	~12–24 години
NATS	Менше готових конекторів, простий набір клієнтів	Мінімальна документація	~24 години і більше
Порівняння безпеки			
Платформа	TLS, SASL, ACL підтримка	Час реакції на вразливості	Патчі безпеки
Kafka	Так, вбудовано	~2–4 дні	Регулярні оновлення від Apache
RabbitMQ	TLS, OAuth2 плагіни	~2–5 днів	Часті оновлення, залежить від контриб'юторів
Pulsar	TLS, аутентифікація, доступ на рівні топіка	~3–7 днів	Оновлення менш регулярні, але команда реагує
NATS	TLS, базова аутентифікація	~3–7 днів	Менше регулярних оновлень, залежить від спільноти

Таблиця В.5 – Порівняння СУБД

Продуктивність			
СУБД	TPS (транзакцій за секунду) при 50 коннектах	Середня затримка (мс)	Тип даних
PostgreSQL	~10 000–12 000 TPS	~5–10	Реляційна (SQL), ACID
MySQL	~8 000–10 000 TPS	~8–12	Реляційна (SQL)
MongoDB	~5 000–7 000 TPS	~10–15	Документна (NoSQL)
Elasticsearch	~3 000–5 000 TPS (залежно від індексації)	~20–30	Пошуково-орієнтована
Масштабованість			
СУБД	Реплікація та Failover	Шардінг даних	Інструменти для K8s інтеграції
PostgreSQL	Стримінгова реплікація, Patroni для автоматичного failover	Обмежений шардінг через розширення (Citus)	Postgres Operators для K8s
MySQL	Master-Slave реплікація, Group Replication	Обмежений шардінг через ProxySQL	Helm Charts, Operators
MongoDB	Вбудований шардінг та репліка, Replica Sets	Легкий шардінг документів	Офіційні MongoDB Kubernetes Operators
Elasticsearch	Вбудований шардінг та репліка в кластері	Автоматичний шардінг індексів	Elastic Cloud on K8s Operator
Порівняння екосистеми та підтримки СУБД			
СУБД	Кількість драйверів та ORM	Документація та гайди	Час відповіді спільноти
PostgreSQL	Дуже багато (TypeORM, Sequelize, Hasura)	Дуже детальна, офіційні керівництва	~6–12 годин
MySQL	Аналогічно багато драйверів та ORM	Дуже відома, багато гайдів	~6–12 годин
MongoDB	Безліч драйверів, Mongoose для Node.js	Хоч і документна, дуже детально описано	~12–24 години
Elasticsearch	Офіційні клієнти для більшості мов	Документація детальна, але фокус на пошук	~12–24 години
Порівняння безпеки СУБД			
СУБД	TLS підтримка	Час реакції на вразливості	Інструменти безпеки з “коробки”
PostgreSQL	Так	~2–4 дні	Ролі, права доступу, SSL/TLS, PG Audit
MySQL	Так	~2–5 днів	Ролі, SSL, базовий аудит
MongoDB	Так, SCRAM аутентифікація	~3–7 днів	Ролі, TLS, але налаштування складніші
Elasticsearch	TLS, RBAC (через X-Pack)	~3–7 днів	Безпека через X-Pack (платна опція)

Додаток Г

Порівняльні таблиці архітектури та аналогів

Таблиця Г.1 – Порівняльна різних видів архітектур програмного забезпечення

Критерій	IBM QRadar SOAR	Splunk SOAR	Cortex XSOAR
Автоматизація	Плейбуки для автоматизації реагування на інциденти.	Автоматизація рутинних завдань через інтеграцію з різними інструментами.	Розширені плейбуки та ранбуки для покрокової обробки інцидентів.
Оркестрація	Інтеграція з продуктами IBM та сторонніми інструментами для централізованого управління інцидентами.	Широкий спектр інтеграцій з різними системами безпеки для створення єдиного центру управління загрозами.	Інтеграція з багатьма інструментами безпеки для ефективної координації між командами.
Інтеграція	Підтримка інтеграції з SIEM, антивірусами та фаєрволами для збору та аналізу даних з різних джерел.	Інтеграція з антивірусами, SIEM-системами, фаєрволами та сервісами Threat Intelligence.	Інтеграція з різними інструментами безпеки в єдиному інтерфейсі.
Масштабованість	Підходить для великих організацій з високими вимогами до безпеки.	Гнучка та масштабована платформа, що підходить для організацій різного розміру.	Висока масштабованість для обробки великої кількості інцидентів.

Продовження таблиці Г.1

Кастомізація	Можливість налаштування плейбуків під специфічні потреби організації.	Підтримка кастомізації плейбуків для конкретних бізнес-вимог.	Гнучкі плейбуки, які можна адаптувати до потреб організації.
Підтримка	Професійна підтримка від IBM з доступом до оновлень та ресурсів.	Підтримка від Splunk з доступом до спільноти користувачів та ресурсів	Підтримка від Palo Alto Networks з доступом до навчальних матеріалів та спільноти.
Вартість	Висока вартість, що може бути виправдана для великих організацій з високими вимогами до безпеки.	Висока вартість, особливо для менших компаній або організацій з обмеженими ресурсами.	Висока вартість, що може бути обмеженням для менших компаній з обмеженим бюджетом.
Вимоги до ресурсів	Потребує значних технічних знань та ресурсів для налаштування та підтримки.	Потребує технічних знань та досвіду для повноцінного використання.	Високі вимоги до ресурсів та технічної підтримки.
Переваги	- Ефективне управління інцидентами. - Інтеграція з продуктами IBM та сторонніми інструментами. - Автоматизація процесів реагування на інциденти.	- Гнучкість та масштабованість. - Підтримка широкого спектру автоматизованих дій. - Можливість кастомізації плейбуків під конкретні вимоги бізнесу.	- Гнучкість та масштабованість. - Підтримка широкого спектру інтеграцій. - Інструменти для командної співпраці та швидкого обміну інформацією.

Додаток Д

Порівняльна таблиця архітектур

Таблиця Д.1 – Порівняльна різних видів архітектур програмного забезпечення

Характеристика	Мікросервісна архітектура	Сервіс-орієнтована архітектура (SOA)	Монолітна архітектура
Гнучкість та масштабованість	Висока. Кожен мікросервіс можна масштабувати окремо	Середня. Масштабування складніше через залежності між сервісами	Низька. Масштабування всієї системи цілком
Незалежне розгортання	Можливе. Мікросервіси розгортаються та оновлюються окремо	Обмежене. Сервіси можуть вимагати синхронізації під час розгортання	Неможливе без перерозгортання всієї системи
Незалежне розгортання	Можливе. Мікросервіси розгортаються та оновлюються окремо.	Обмежене. Сервіси можуть вимагати синхронізації під час розгортання.	Неможливе без перерозгортання всієї системи.
Надійність та відмовостійкість	Висока. Збій одного мікросервісу не впливає на інші.	Середня. Залежності можуть спричиняти каскадні збої.	Низька. Збій може вивести з ладу всю систему.
Технологічна незалежність	Висока. Мікросервіси можуть використовувати різні технології	Обмежена. Часто потрібна стандартизація технологій.	Низька. Одна технологія для всієї системи.
Протоколи обміну даними	Легкі, як правило, REST, JSON	Складніші, наприклад, SOAP, XML	Внутрішні виклики без стандартних протоколів
Впровадження нових технологій	Легке. Можна експериментувати в окремих мікросервісах	Складніше через необхідність узгодженості сервісів	Важко. Потрібна зміна всієї системи
Управління залежностями	Просте. Ізоляція в контейнерах спрощує управління	Складніше. Спільні залежності між сервісами	Складне. Залежності в межах одного додатка
Масштаб розробки	Підходить для розподілених команд, що працюють паралельно	Потребує координації між командами	Єдина команда, складно масштабувати розробку

Додаток Е ІЛЮСТРАТИВНА ЧАСТИНА

ІЛЮСТРАТИВНА ЧАСТИНА

СИСТЕМА АВТОМАТИЗАЦІЇ РЕАГУВАННЯ НА ІНЦИДЕНТИ БЕЗПЕКИ

Виконав: студент 2 курсу, групи 1БС-23м
Спеціальність 125 Кібербезпека та захист інформації
_____Владислав БУГАЄЦЬ

Керівник: к. т. н., доцент каф. ЗІ
_____ Леонід КУПЕРШТЕЙН
«13» ____12____ 2024 р.

СТРУКТУРНА СХЕМА СИСТЕМИ АВТОМАТИЗАЦІЇ РЕАГУВАННЯ НА ІНЦИДЕНТИ

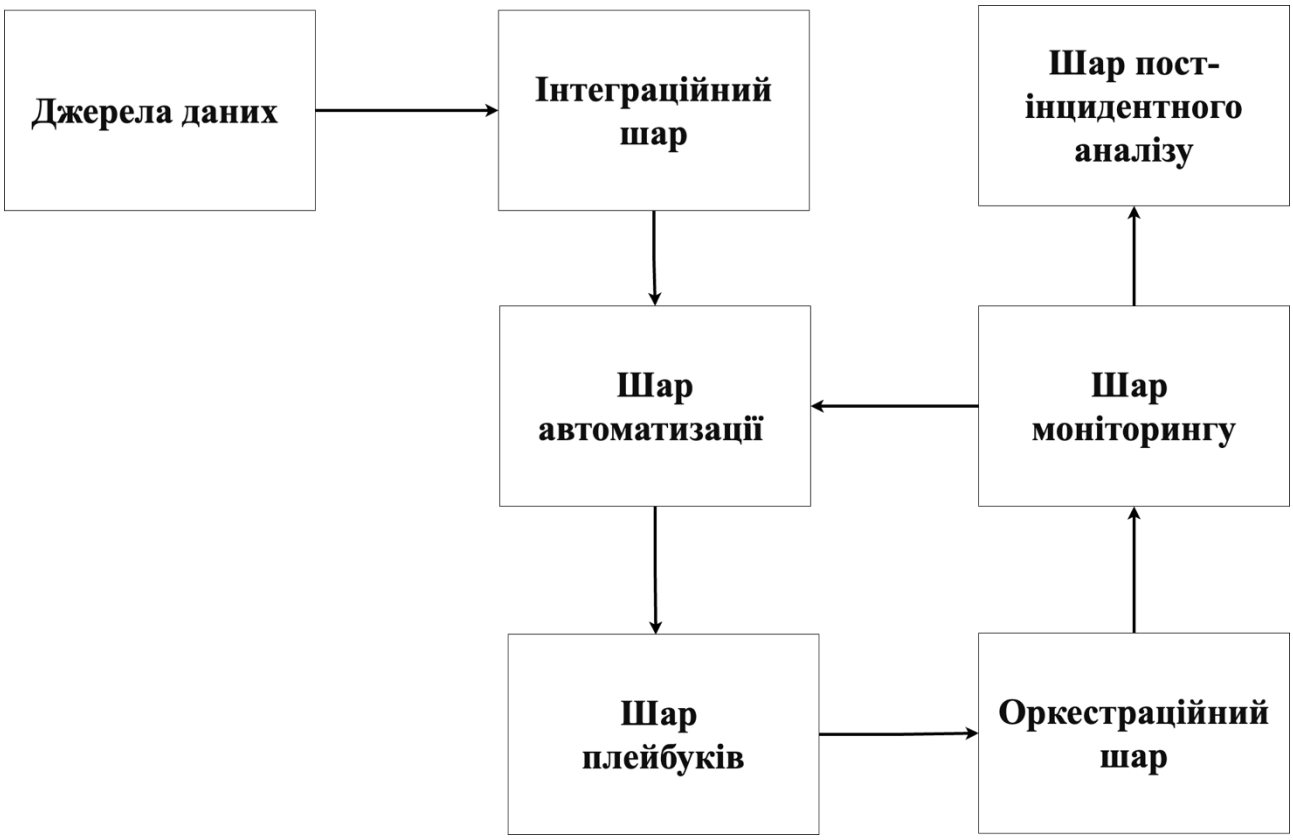


СХЕМА РОБОТИ ЧЕРГ

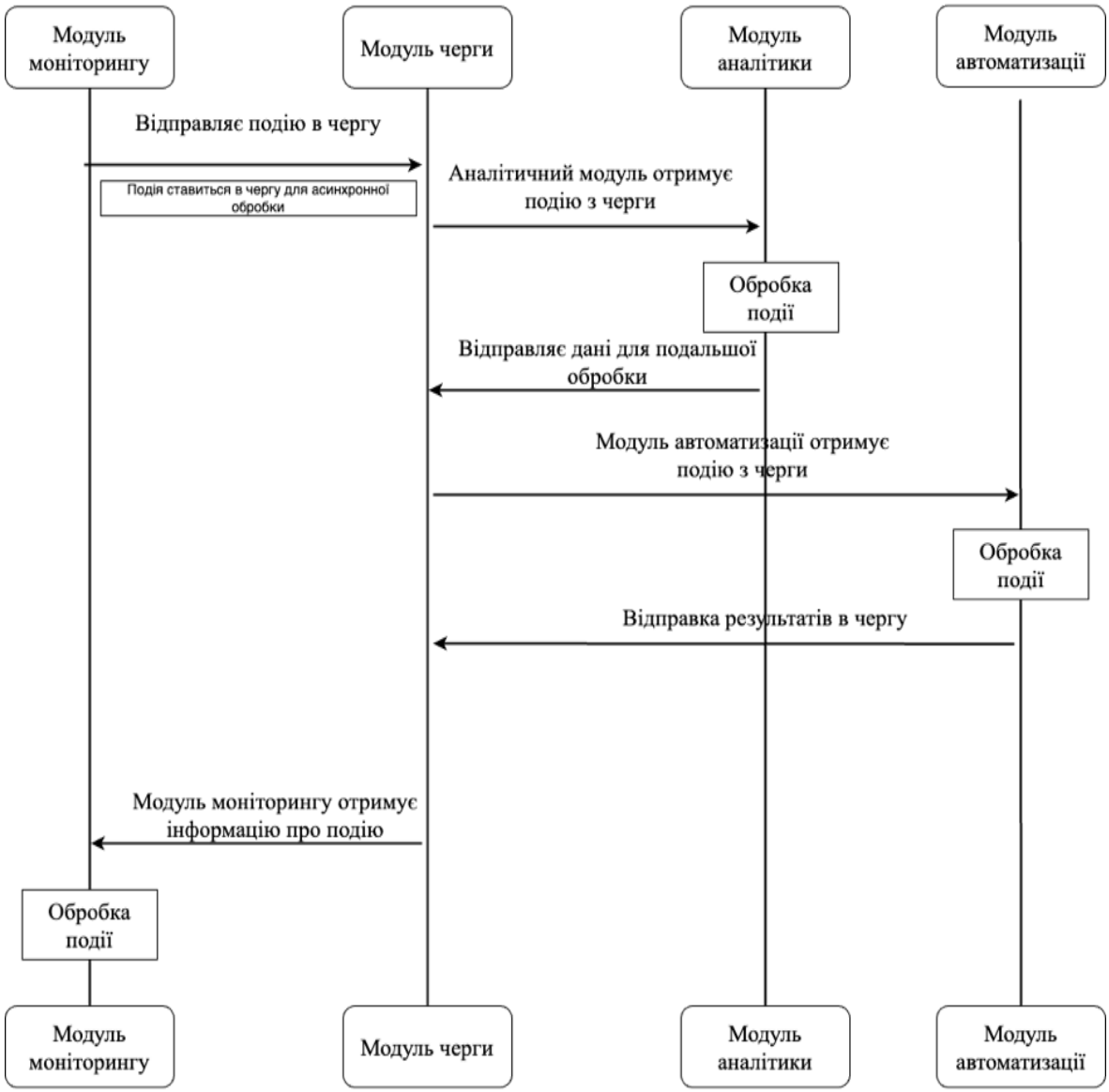
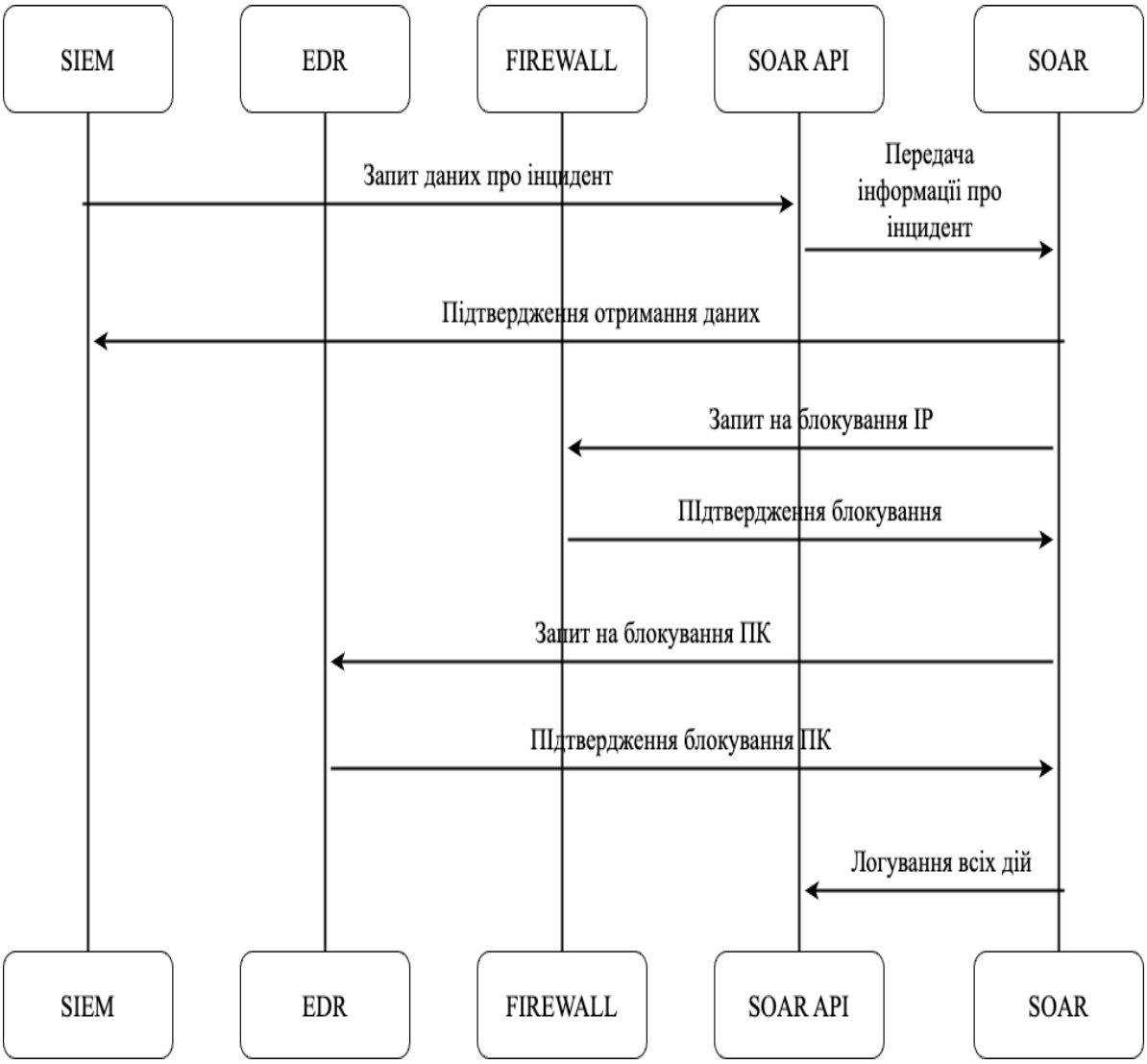
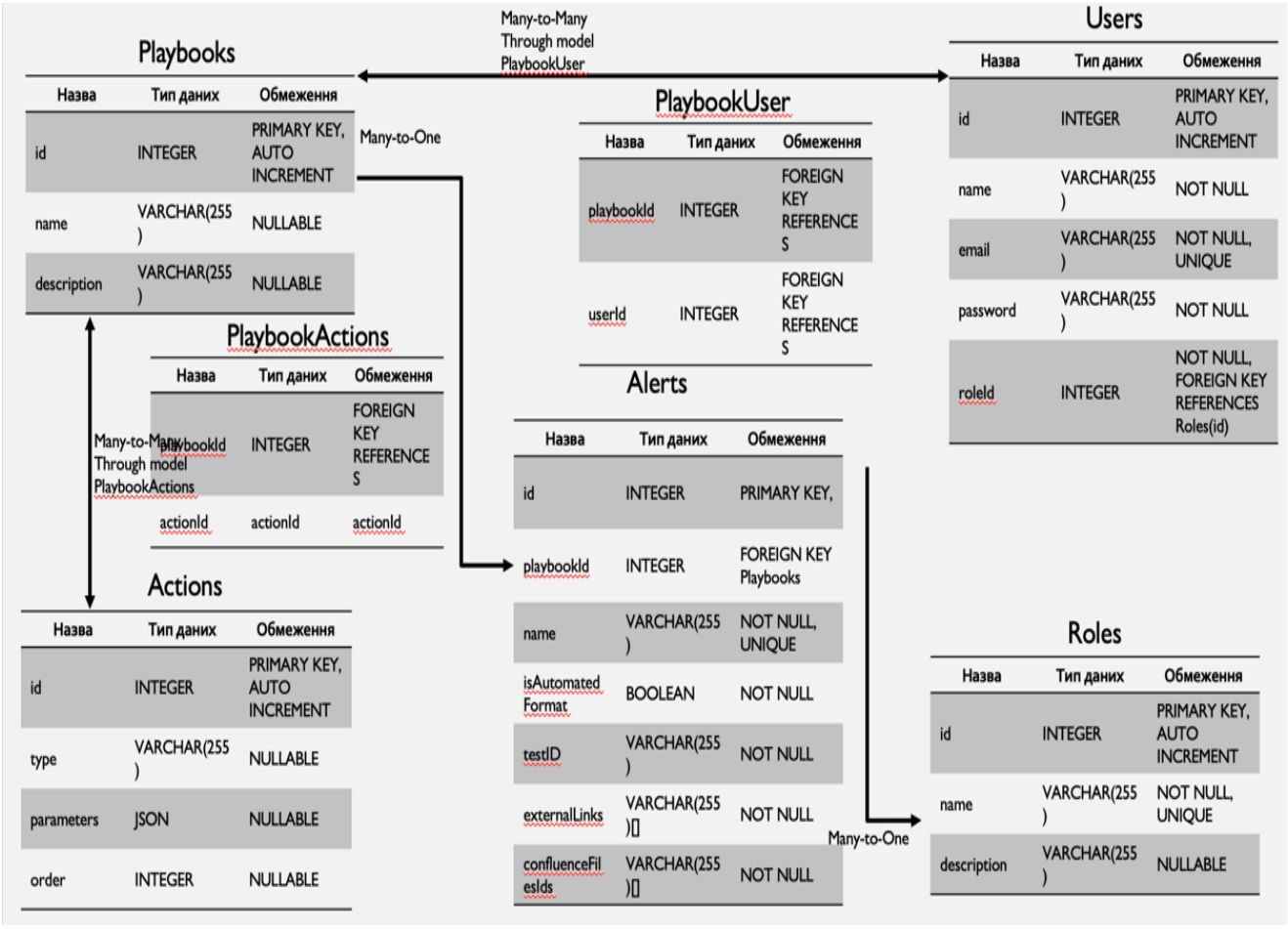


СХЕМА РОБОТИ МЕХАНІЗМУ АВТЕНТИФІКАЦІЇ

СХЕМА ЗВ`ЯЗКІВ З СИСТЕМАМИ БЕЗПЕКИ



СТРУКТУРА БАЗИ ДАНИХ



ЖИТТЄВИЙ ЦИКЛ РОЗРОБКИ ПЛЕЙБУКІВ

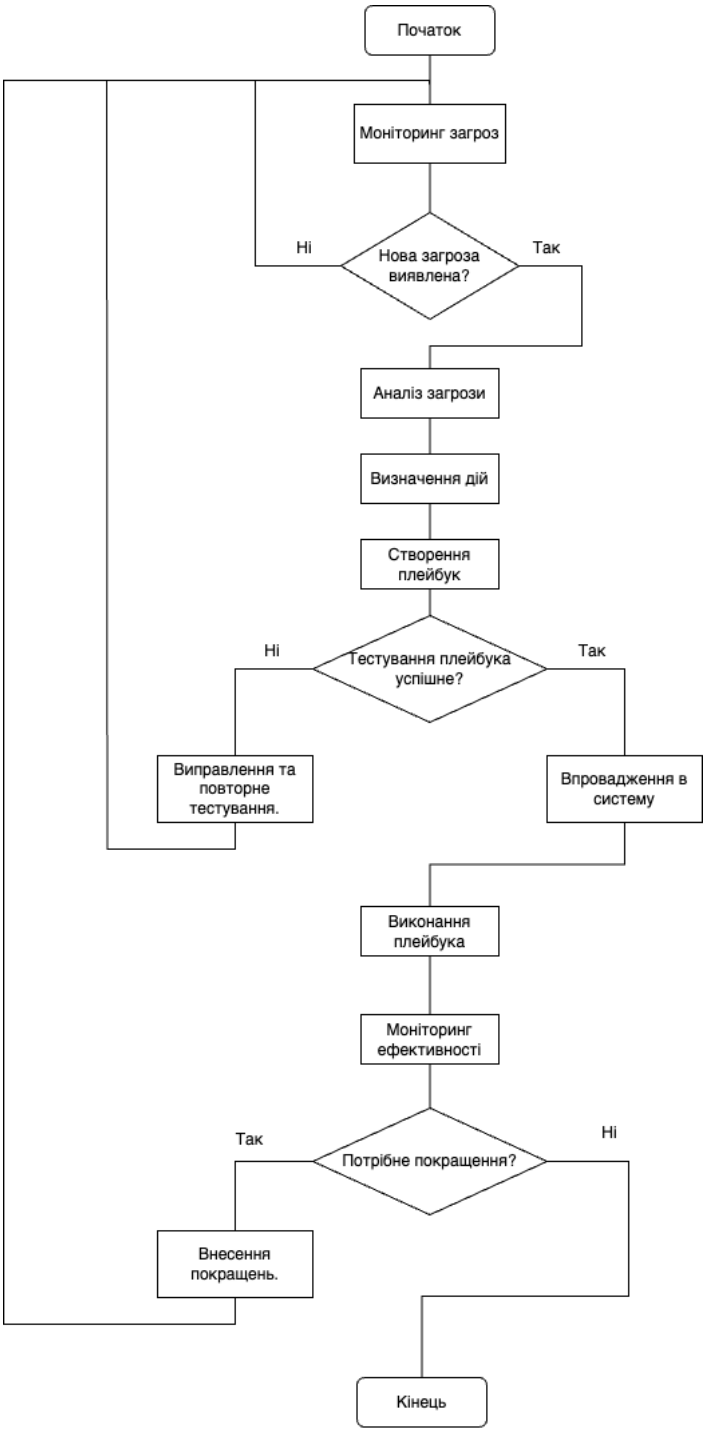


СХЕМА АВТОМАТИЗОВАНОГО РЕАГУВАННЯ НА ІНЦИДЕНТИ

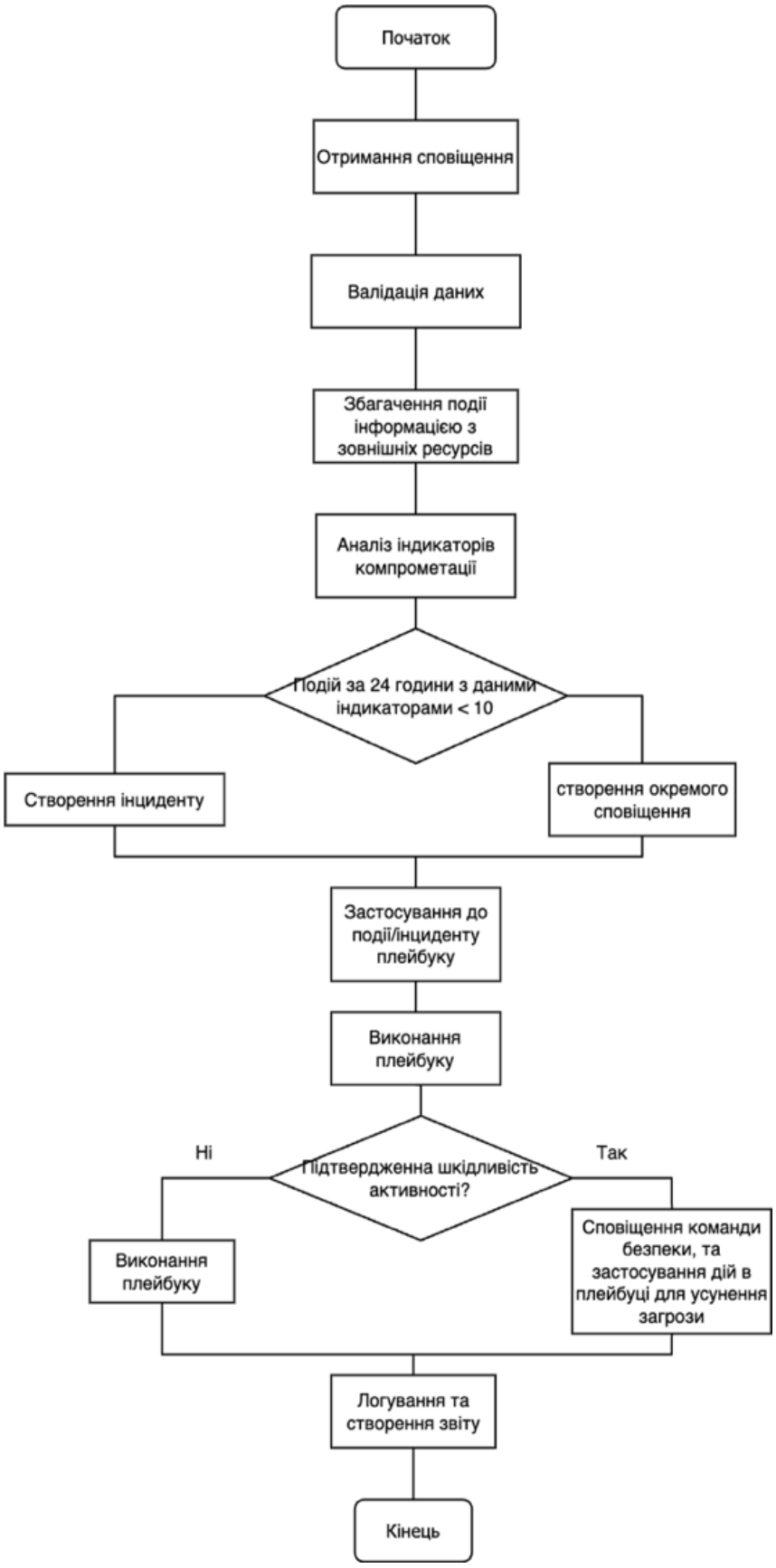


СХЕМА ПЛЕЙБУКУ ЗАХИСТУ ОБЛІКОВИХ ЗАПИСІВ

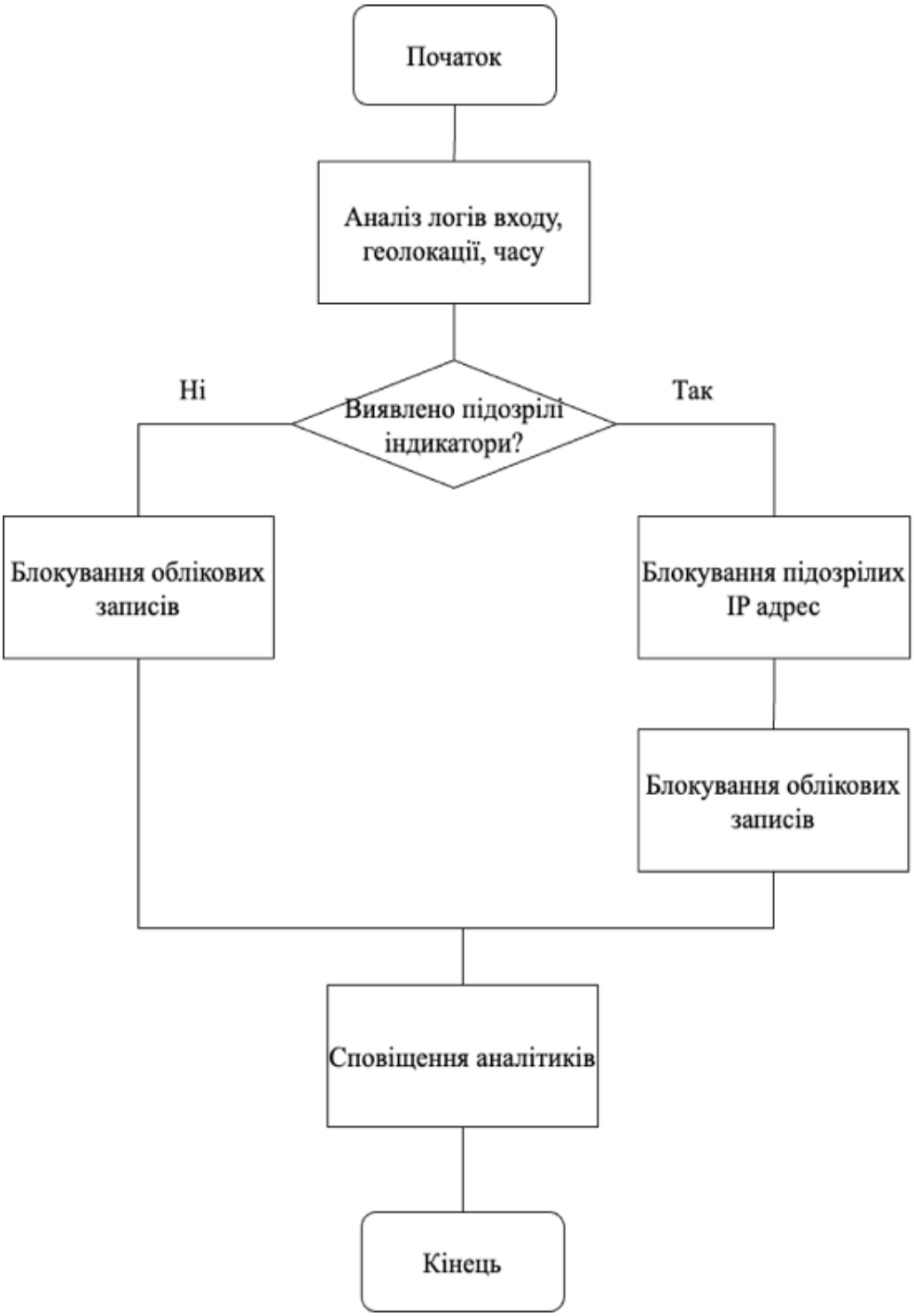


СХЕМА РОБОТИ ПЛЕЙБУКУ БЕЗПЕКИ ЕЛЕКТРОННОЇ ПОШТИ

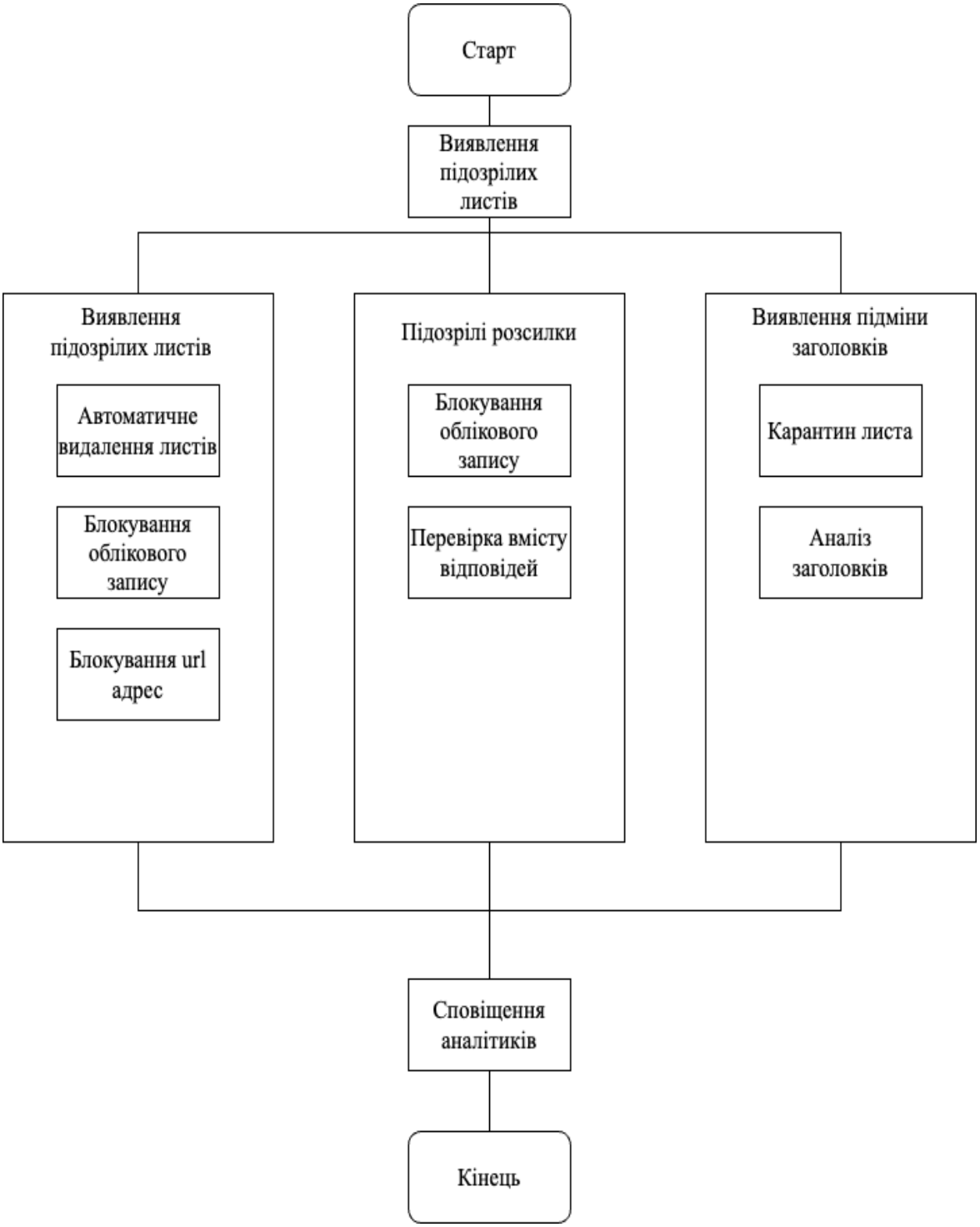


СХЕМА РОБОТИ ПЛЕЙБУКУ ЗАХИСТУ ХМАРНОГО СЕРЕДОВИЩА

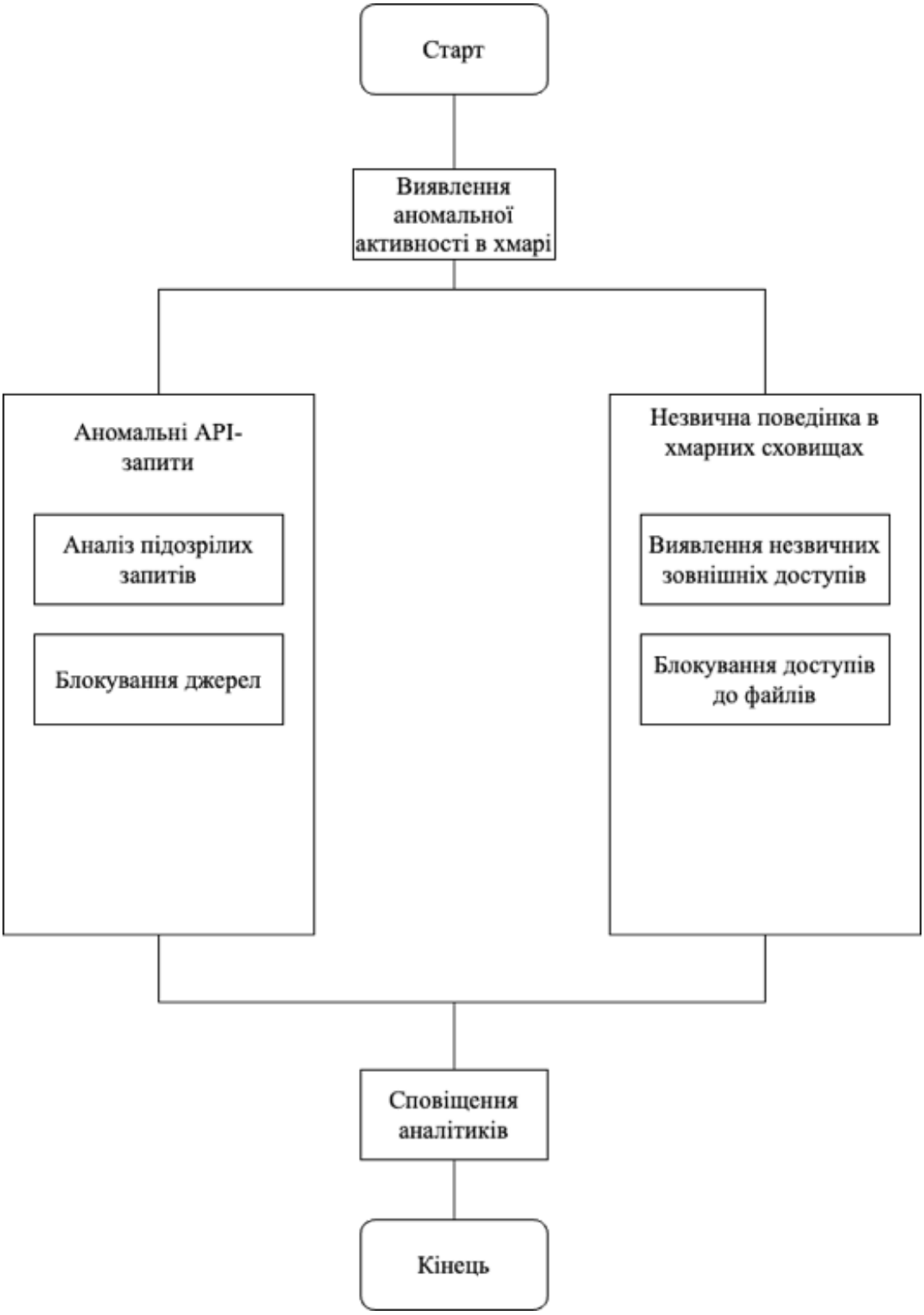


СХЕМА РОБОТИ ПЛЕЙБУКУ МЕРЕЖЕВОЇ БЕЗПЕКИ

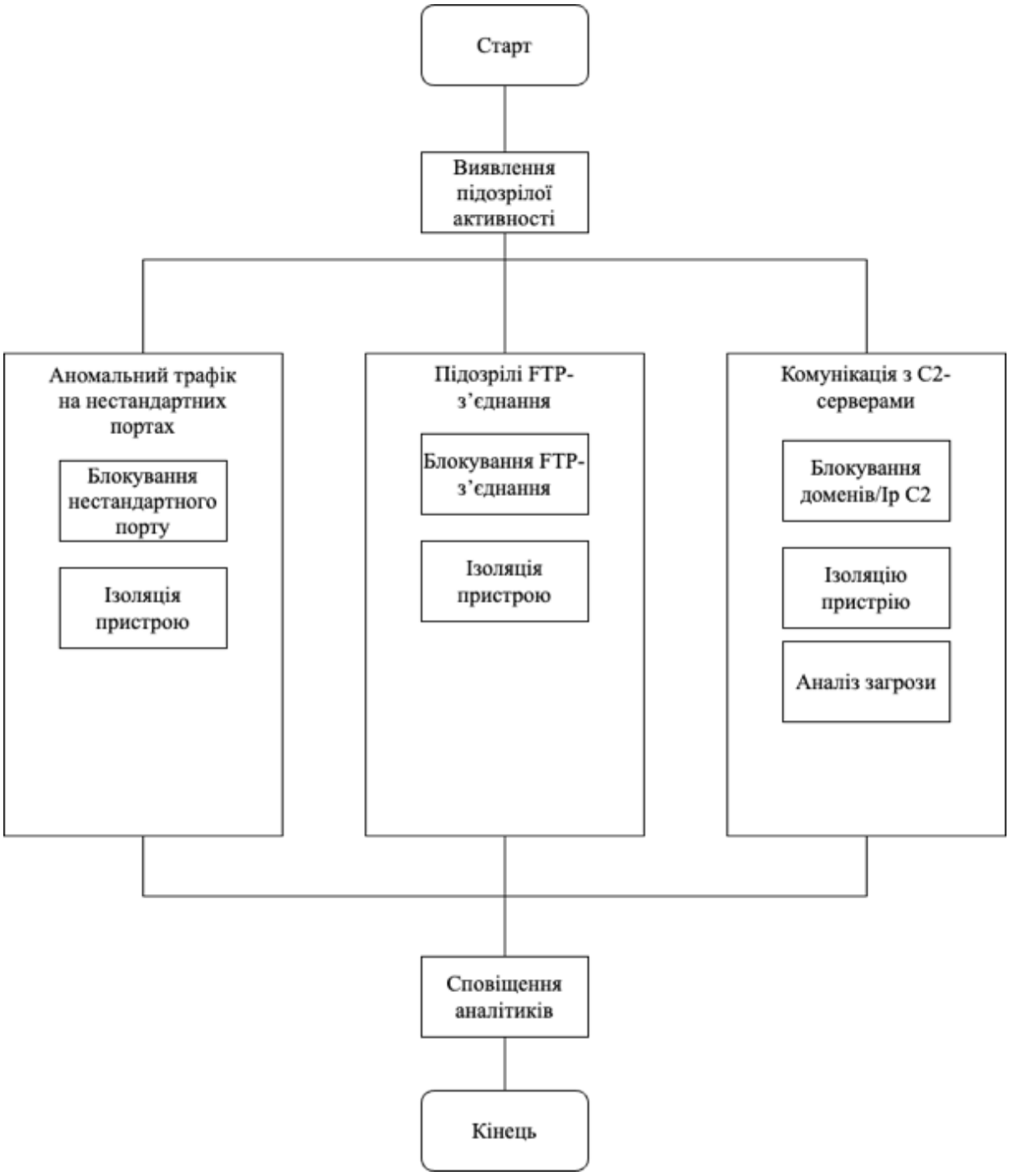
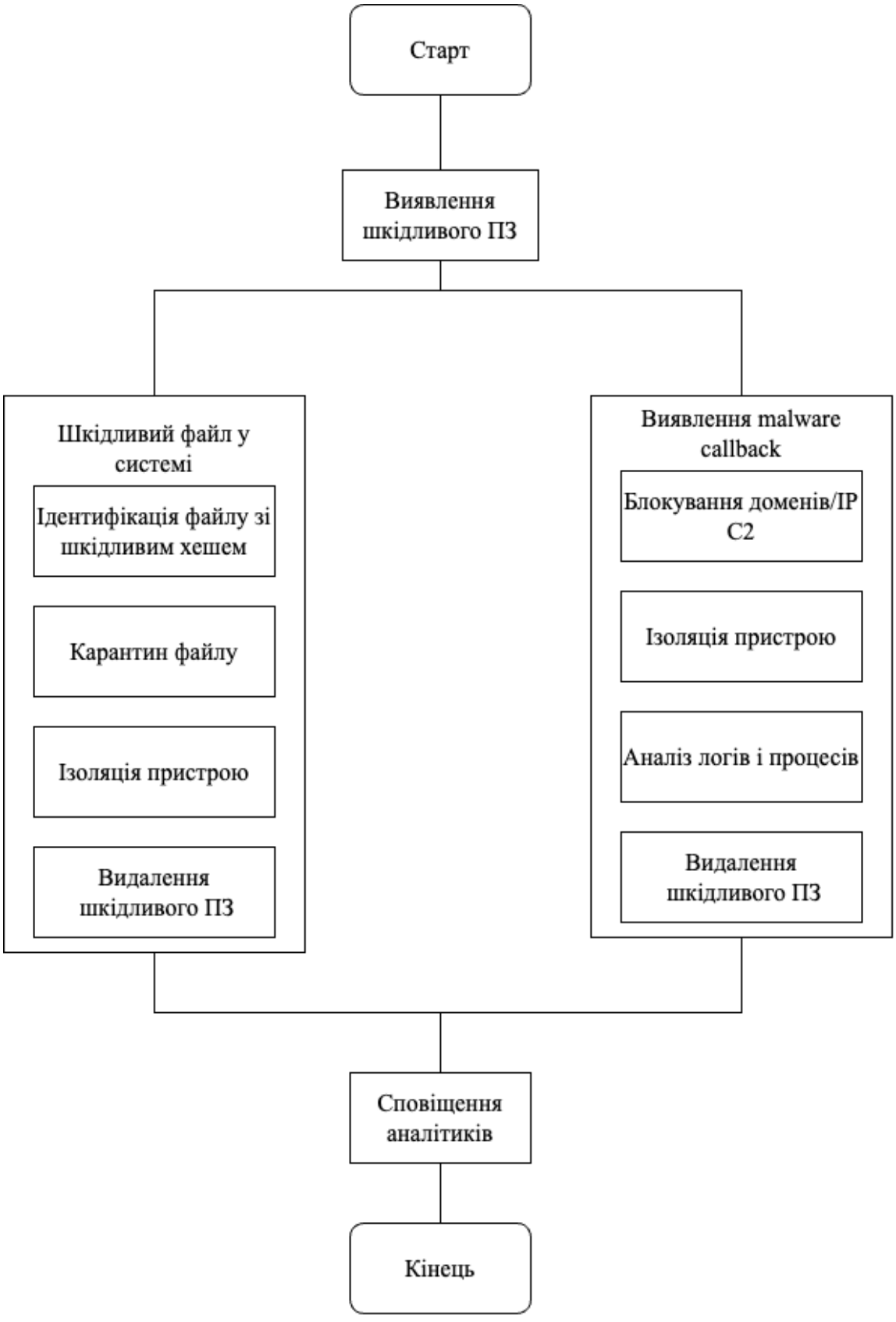


СХЕМА РОБОТИ ПЛЕЙБУКУ ПРОТИДІЇ ШКІДЛИВОМУ ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННЮ



КЛІЄНТСКИЙ ІНТЕРФЕЙС

ТОВ "НП Діджитал"

Україна 03026

м. Київ, вул. столичне шосе, будинок 103

АКТ ВПРОВАДЖЕННЯ

Цим актом підтвердужується, що окремі результати магістерської дипломної роботи на тему «Система автоматизації реагування на інциденти безпеки», яку виконав студент групи 1БС-23м Бугаєць Владислав за спеціальністю 125 «Кібербезпека», впроваджені в діяльність організації «ТОВ "НП Діджитал"».

Цей документ не є підставою для фінансових розрахунків.

Дата 16.12.2024

Керівник відділу реагування та моніторингу

Дмитро ДМИТРІЄНКО