Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(освітній рівень)

на тему: Розробка системи моніторингу безпеки для вебдодатків

Виконав: студент (ка) VI курсу, групи СБмз-62
Спеціальності:
125 «Кібербезпека та захист інформації»
(шифр і назва напряму підготовки, спеціальності)
Покидко Олександр Вікторович
підпис (прізвище та ініціали)

Керівник Лечаченко Т. А.
підпис (прізвище та ініціали)

Нормоконтроль Стадник М. А.
підпис (прізвище та ініціали)

Завідувач кафедри Загородна Н.В.
підпис (прізвище та ініціали)

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека та захист інформації
(шифр і назва спеціальності)

Студенту Покидко Олександру Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка системи моніторингу безпеки для вебдодатків

Керівник роботи Лечаченко Тарас Анатолійович, доктор філософії., старший викладач кафедри КБ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «11» 12 2025 року № 4/7-1066

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи артефакти моніторингу простого вебдодатку

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1 Основи моніторингу в контексті кібербезпеки. 1.1 Контекст, сфера застосування та стандарти. 1.2 Основи спостереження для безпеки. 1.3 Цілі моніторингу та їхні питання безпеки. 1.4 Шаблони виявлення на основі даних моніторингу. Розділ 2 Інструменти, архітектури та моделі розгортання. 2.1 Загальна архітектура платформ моніторингу. 2.2 Моніторинг на основі SaaS. 2.3 Стек з відкритим вихідним кодом та власним хостингом. 2.4 Гібридні моделі. 2.5 Методи розгортань систем моніторингу з акцентом на безпеку. Розділ 3 Програмна реалізація засобу автоматичного моніторингу витоку інформації. 3.1 Проектування відношень компонентів автоматизованої системи 3.2 Реалізація системи моніторингу веб додатків. 3.3 Інтеграція демонстраційного вебдодатку та E2E тестування. 3.4 Дашборди для централізованого моніторингу та сповіщення Розділ 4 Охорона праці та безпека в надзвичайних ситуаціях. 4.1 Охорона праці. 4.2 Ергономічні вимоги до робочого місця користувача персональним комп'ютером (ПК)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульна сторінка. 2. Актуальність теми і наукова новизна. 3. Мета роботи і завдання. 4. Ключові артефакти та компоненти моніторингу. 5. Типові моделі розгортання систем моніторингу вебдодатків. 6. Реалізація системи моніторингу вебдодатків. 7. Інтеграція об'єкту моніторингу 8. Моделювання загрози та оцінка ефективності системи. 9. Перспективи застосування. 10. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г.М., к.т.н., доцент		
Безпека в надзвичайних ситуаціях	Теслюк В. М., проректор з адміністративно-господарської роботи та будівництва		

7. Дата видачі завдання 19.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	20.09 – 22.09	Виконано
2.	Збір джерел та інформації за досліджуваною темою	22.09 – 25.09	Виконано
3.	Огляд сучасних систем моніторинг вебдодатків	25.09 – 28.09	Виконано
4.	Аналіз сфер застосування систем моніторингу	28.09 – 30.09	Виконано
5.	Планування підходів та архітектури з реалізації системи моніторингу вебдодатків	01.10 – 03.10	Виконано
6.	Оформлення розділу 1 “Основи моніторингу у контексті кібербезпеки”	03.10 – 05.10	Виконано
7.	Оформлення розділу 2 “Інструменти, архітектури та моделі розгортання платформ моніторингу”	05.10 – 10.10	Виконано
8.	Створення модулів для системи моніторингу	10.10 – 19.10	Виконано
9.	Проєктування та інтеграція демонстраційного вебдодатку у систему моніторингу	20.10 – 28.10	Виконано
10.	Впровадження інструментів візуалізації результатів роботи	06.11 – 15.11	Виконано
11.	Оформлення розділу 3 “Програмна реалізація автоматичного моніторингу витоку інформації”	04.12 – 8.12	Виконано
12.	Оформлення розділу 4 “Охорона праці та безпека в надзвичайних ситуаціях”	12.12 – 14.12	Виконано
13.	Нормоконтроль	18.12.2025	Виконано
14.	Перевірка на плагіат	18.12 – 19.12	Виконано
15.	Попередній захист кваліфікаційної роботи	22.12 – 23.12	
16.	Захист кваліфікаційної роботи	24.12.2025	

Студент _____ Покидко О. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Лечаченко Т. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка системи моніторингу безпеки для вебдодатків // ОР «Магістр» // Покидко Олександр Вікторович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБмз-62 // Тернопіль, 2025 // С. 98, рис. – 16, табл. – 1, кресл. – 10, додатк. – 3.

Ключові слова: вебдодатки, моніторинг, метрики, логи, Grafana.

У магістерській роботі розглянуто розробку системи моніторингу безпеки для вебдодатків на основі даних журналів подій та телеметрії.

У першому розділі проаналізовано роль моніторингу в кібербезпеці, визначено контекст застосування та релевантні стандарти, описано основи спостереження для задач виявлення, розслідування та реагування на інциденти. Сформульовано цілі моніторингу й перелік аспектів безпеки, на для оцінки яких використовуються метрики, логи (журнали подій), а також запропоновано підхід до побудови шаблонів виявлення інцидентів на основі цих даних.

У другому розділі досліджено інструменти, архітектури та моделі розгортання платформ моніторингу, включно з SaaS, власним хостингом (з фокусом на екосистемі Grafana) і гібридними підходами, а також методи впровадження з урахуванням вимог безпеки.

У третьому розділі кваліфікаційної роботи, реалізовано прототип системи моніторингу та інтегрований демонстраційний веб застосунок з автоматичним розгортанням на базі системи Docker. Система моніторингу збирає телеметрію вебзастосунку та реалізує правила й сповіщення для виявлення підозрілих подій і потенційних витоків інформації.

ABSTRACT

Development of a security monitoring system for web applications// Thesis of educational level "Master"// Pokydko Oleksandr // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, group SBmz-62 // Ternopil, 2025 // p. 98, figs. – 16, tbls. – 1, drws. – 10, apps. – 3.

Keywords: web applications, monitoring, metrics, logs, Grafana

The master's thesis considers the development of a security monitoring system for web applications based on event log and telemetry data.

The first section analyzes the role of monitoring in cybersecurity, identifies the context of application and relevant standards, describes the basics of surveillance for detection, investigation, and incident response tasks. The monitoring goals and a list of security questions that metrics and logs (event logs) should answer are formulated, and an approach to building detection templates based on this data is proposed.

The second section explores tools, architectures, and deployment models for monitoring platforms, including SaaS, self-hosting (with a focus on the Grafana ecosystem), and hybrid approaches, as well as implementation methods taking into account security requirements.

In the third chapter of the qualification work, the practical result is the software implementation of a prototype monitoring system, along with an integrated demonstration web application with automated Docker-based deployment, which collects web application telemetry and supports rules and notifications to detect suspicious events and potential information leaks.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

API	—	Application Programming Interface
CI/CD	—	Continuous Integration / Continuous Delivery
DNS	—	Domain Name System
DoS	—	Denial of Service
E2E	—	End-to-End
HTTP	—	Hypertext Transfer Protocol
HTTPS	—	Hypertext Transfer Protocol Secure
IAM	—	Identity and Access Management
IDS	—	Intrusion Detection System
JSON	—	JavaScript Object Notation
MFA	—	Multi-factor authentication
OTP	—	One-Time Password
RBAC	—	Remote Desktop Protocol
SaaS	—	Software as a Service
SIEM	—	Security Information and Event Management
SQL	—	Structured Query Language
SOC	—	Security Operations Center
TLS	—	Transport Layer Security
WAF	—	Web Application Firewall

ВСТУП

Актуальність теми. Вебдодатки є основою більшості сучасних сервісів і водночас залишаються однією з найчастіших цілей кібератак через постійну доступність з мережі, складність технологій та велику кількість інтеграцій. Навіть за наявності захисних механізмів (автентифікація, WAF, контроль доступу, ізоляція середовищ) безперервне виявлення інцидентів і швидке реагування неможливі без якісного моніторингу. Дані, що отримуються з метрик і логів, дозволяють не лише оцінювати надійність і продуктивність вебдодатків, а й відстежувати підозрілу поведінку, аномалії, витоки інформації, зловживання обліковими записами та помилки конфігурацій. Тому розробка системи моніторингу вебдодатків із фокусом на безпеку та підготовкою шаблонів виявлення інцидентів є актуальною задачею для організацій, що використовують вебсервіси як у хмарі, так і у власній інфраструктурі.

Мета і задачі дослідження. Метою роботи є розробка системи моніторингу безпеки вебдодатків, яка збирає та узгоджує дані метрик та логів, на основі яких формуються правила виявлення підозрілих подій з метою забезпечення базових механізмів сповіщення для підтримки процесів реагування на інциденти.

Задачами дослідження є:

– Проаналізувати роль моніторингу вебдодатків у контексті кібербезпеки.

– Описати артефакти спостереження для безпеки вебдодатків.

– Систематизувати ключові аспекти безпеки на різних рівнях моніторингу.

– Дослідити інструменти, архітектури та моделі розгортання платформ моніторингу, включно з SaaS та гібридними підходами.

– Розробити методи розгортання системи моніторингу з урахуванням вимог безпеки.

– Реалізувати прототип системи моніторингу вебдодатків із використанням обраних технологій та утиліт.

– Розробити демонстраційний вебдодаток, налаштувати збір телеметрії та виконати E2E тестування сценаріїв, що імітують типові події безпеки.

Об’єкт дослідження. Процеси моніторингу та забезпечення безпеки вебдодатків.

Предмет дослідження. Методи, моделі та програмні засоби побудови системи моніторингу вебдодатків.

Наукова новизна одержаних результатів кваліфікаційної роботи. Наукова новизна роботи полягає в впровадженні підходу побудови шаблонів виявлення інцидентів на основі узгоджених даних метрик, логів і телеметрії, орієнтованих на нетипові сценарії кібератак спрямованих на вебдодатки.

Практичне значення одержаних результатів. Практичне значення роботи полягає у створенні прототипу системи моніторингу вебдодатків, який може бути використаний як основа для впровадження в реальному середовищі або для навчальних і демонстраційних цілей. Результати включають опис архітектурних підходів і моделей розгортання, рекомендації з безпечного впровадження платформи моніторингу, а також набір практичних сценаріїв і E2E тестів, що перевіряють готовність системи до виявлення підозрілих подій та підтримки реагування на інциденти.

Апробація результатів магістерської роботи. Основні результати проведених досліджень обговорювались на: ITTAP 2021. The 1st International Workshop on Information Technologies: Theoretical and Applied Problems 2021.

Публікації. Основні результати кваліфікаційної роботи опубліковано у працях конференції (див. Додаток А).

РОЗДІЛ 1 ОСНОВИ МОНІТОРИНГУ ВЕБДОДАТКІВ У КОНТЕКСТІ КІБЕРБЕЗПЕКИ

1.1 Контекст, сфера застосування та стандарти моніторингу

Моніторинг у інформаційних системах відноситься до безперервного збору та аналізу даних про роботу системи. У контексті кібербезпеки моніторинг є незамінним, оскільки захист не може функціонувати без видимості системної діяльності. Як зазначається в галузевих рекомендаціях, для ефективного моніторингу системи на предмет безпеки, а також продуктивності та безперебійності, організаціям потрібен комплексний стек моніторингу, який забезпечуватиме основу для виявлення та оповіщення [1]. Іншими словами, кіберзахист спирається на спостережуваність – ви не можете захистити те, що не бачите. Наприклад, у рейтингу OWASP Top 10 наголошується, що без достатнього ведення журналу (також логування) та моніторингу порушення часто залишаються непоміченими протягом тривалого часу [2]. Ефективний моніторинг забезпечує телеметрію, необхідну для виявлення вторгнень, розслідування підозрілої поведінки та своєчасного реагування на інциденти.

Дана робота буде зосереджена саме на моніторингу веб-додатків та суміжних з ними компонентами. Типовий стек веб-додатків включає такі елементи, як зворотній проксі-сервер, середовище виконання додатків, як наприклад Python, базу даних, а також операційну систему та служби хоста. Кожен з цих рівнів генерує цінні дані – від метрик HTTP-запитів на проксі-сервері до журналів додатків, журналів запитів до бази даних та подій на рівні ОС. Звужуючи сферу застосування до вебдодатків, робота підкреслюватиме, як моніторинг може акцентувати увагу на атаки, спрямовані на системи, орієнтовані на веб. Належні практики моніторингу для вебдодатків охоплюють не лише сам додаток, але й навколишню інфраструктуру. Ця цілісна видимість дозволяє виявляти атаки, підтримує реагування на інциденти, та забезпечує журнали

аудиту, необхідні для відповідності вимогам (compliance) та криміналістичному аналізу (forensic).

Моніторинг безпеки забезпечує виявлення, реагування, аудит та дотримання вимог. За умови правильного виконання моніторингу, він створює дані, які можна аналізувати для виявлення індикаторів компрометації та ініціювання сповіщень. Він також створює аудит з записом подій – необхідний для розслідування інцидентів після їх виникнення та для демонстрації відповідності стандартам безпеки. Фактично, структури та нормативні акти чітко вимагають ведення журналу та моніторингу подій, пов’язаних з безпекою. Наприклад, у керівництві OWASP щодо реєстрації та моніторингу збоїв наголошується, що виявлення порушень та реагування на них є критично важливими для підзвітності, прозорості та криміналістики [2]. Стандарт безпеки даних індустрії платіжних карток (PCI DSS) [29] також містить спеціальні вимоги до ведення журналів. Вимога PCI DSS 10 вимагає відстеження та моніторингу будь-якого доступу до конфіденційних систем і даних. Зокрема Вимога PCI DSS 4.0 10.2 вимагає від організацій впроваджувати журнали аудиту для виявлення підозрілої системної активності, перш ніж вона стане активною загрозою [3]. ISO/IEC 27001 (провідний стандарт управління безпекою) також передбачає засоби контролю для реєстрації та моніторингу подій – журнали подій повинні створюватися, зберігатися та регулярно переглядатися для запису дій користувачів, винятків, несправностей та подій інформаційної безпеки [4]. Ці стандарти підкреслюють, що надійний моніторинг — це не просто найкраща практика, а й офіційне очікування в рамках системи дотримання вимог безпеки.

Наведена на рисунку 1.1 діаграма ілюструє середовище з рівнями, кожен з яких, починаючи від проксі-сервера до програми та бази даних може генерувати та надсилати дані до системи моніторингу. Це дозволяє отримати агреговану видимість системи, яка дозволить детальніше виявлення, позначаючи аномалії в стеку надаючи контекст, наприклад пов’язуючи підозрілий веб запит із запитом до бази даних та помилками сервера. Як зазначає одне галузеве джерело, ведення журналу та моніторинг «фіксують кожну значущу дію», щоб можна було

відстежувати закономірності, досліджувати проблеми та впевнено виконувати реактивні дії [5].

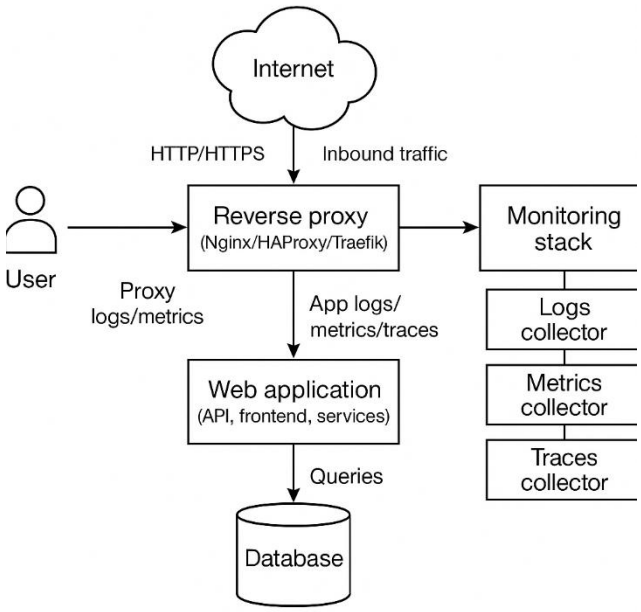

Рисунок 1.1 – Приклад простого середовища вебдодатку з моніторингом

Йдеться не лише про виявлення атак – це також життєво важливо для підтвердження відповідності та забезпечення належної роботи систем, саме тому такі фреймворки, як ISO 27001, вимагають цього не лише заради відповідності, але й для перевірки ефективності засобів контролю безпеки на практиці.

Рисунок 1.2 – Приклад дашборду для моніторингу веб-додатку

На запропонованому рисунку 1.2 можна побачити такі метрики як кількість запитів до веб додатку, підрахунок різних статус кодів HTTP, загальна успішність запитів, та затримка запитів у розділенні по p99, p95, p90 та p50.

1.2 Артефакти спостереження для безпеки вебдодатків

Сучасну спостережуваність часто описують з точки зору трьох стовпів – метрик, логів та трейсів. Усі три відіграють певну роль у моніторингу, і кожен з них пропонує різний погляд на поведінку системи. У даній роботі розглянуто ці артефакти з точки зору безпеки, підкреслюючи, як кожен тип даних спостереження може допомогти у виявленні або розслідуванні інцидентів безпеки.

1.2.1 Метрики

Метрики – це числові дані часових рядів, які вимірюють певні аспекти продуктивності або навантаження системи з плином часу. Зазвичай вони позначені мітками (вимірами), такими як назва хоста, назва сервісу, шлях звернення, тощо. У контексті вебдодатку приклади корисних метрик включають:

– Коди стану HTTP (наприклад, кількість помилок клієнта 4xx та помилок сервера 5xx за хвилину).

– Події автентифікації (наприклад, кількість спроб входу, невдалих входів, скидання паролів).

– Швидкість трафіку (запити за секунду, або IP адреси що відвідують сайт за годину).

– Використання ресурсів інфраструктури (процесор, пам’ять, дисковий ввід/вивід на сервері).

– Метрики компонентів безпеки (наприклад, кількість запитів, заблокованих брандмауером вебдодатку, або частота сповіщень від датчика виявлення вторгнень).

Метрики корисні для виявлення аномалій, оскільки вони забезпечують високорівневе уявлення про ситуацію в режимі реального часу. Раптові зміни в шаблонах метрик можуть бути раннім показником атаки. Наприклад, сплеск помилок HTTP 401 (Несанкціоновано) чи 403 (Заборонено) може свідчити про спробу брут форс атаки або атаку з перехопленням облікових даних, особливо якщо лише одна IP-адреса викликає багато невдалих входів. Різке зростання помилок 404 (Не знайдено) для незвичайних URL-адрес може сигналізувати про те, що хтось сканує систему на наявність прихованих або вразливих файлів (проводить розвідку). Сплески частоти помилок сервера 5xx можуть виникати, коли дії зловмисника призводять до винятків або збоїв програм (можливо, під час спроби експлуатації). Навіть показники інфраструктури, такі як завантаження процесора або пам’яті, якщо вони несподівано досягають максимуму, можуть свідчити про зловмисну діяльність, таку як споживання ресурсів шкідливим програмним забезпеченням для майнінгу криптовалюти.

Ключовою перевагою метрик є швидкість та агрегація – вони збираються через регулярні проміжки часу та можуть швидко сповістити за допомогою алертів. Організації часто встановлюють порогові значення для сповіщень або використовують статистичні моделі для виявлення відхилень. Наприклад, Grafana Labs обговорювала процедуру сповіщення базуючись на аномаліях в метриках які збирав Prometheus, використовуючи стандартні розрахунки відхилень або використовуючи методи машинного навчання (див. Рис. 1.3) [6].

Рисунок 1.3 – Приклад візуалізації відхилень у вигляді графіку.

Системи виявлення аномалій метрик можуть повідомляти команди безпеки, коли щось, наприклад, коефіцієнт спроб входу або коефіцієнт помилок, виходить далеко за межі нормальних показників. У таблиці 1.1 наведено приклади метрик, що стосуються безпеки веб-застосунків, та як їх можна використовувати.

Таблиця 1.1 – Приклади простих метрик та їх використання

Мітка Метрики	Джерело	Релевантність до безпеки	Приклад сповіщення
Кількість помилок HTTP 404	Веб сервер або Проксі сервер	Високі показники 404 на незвичайних URL-адресах можуть свідчити про сканування на наявність вразливостей	Сповіщати, якщо кількість помилок 404 з однієї IP-адреси або ASN перевищує поріг протягом 5 хвилин
Кількість помилок HTTP 500	Веб-додаток	Велика кількість помилок 5xx можуть виникати під час спроб використання експлойтів, що призводить до винятків або збоїв.	Сповіщення, якщо рівень помилок сервера подвоюється порівняно з базовим рівнем (потенційна експлуатація або атака DoS/DDoS [27])
Невдалі спроби входу	Сервіс авторизації або вебдодаток	Повторні невдалі входи можуть означати підбір пароля методом перебору або підтасування облікових даних.	Сповіщати, якщо > X невдалих входів для одного облікового запису або IP-адреси за Y хвилин.

На практиці, метрики забезпечують швидкий відгук. Вони не розкажуть повну історію інциденту, але можуть швидко показати, що щось не так і де шукати. Однак, самі показники не містять деталей подій – сплеск також може бути пов’язаний із реальним сплеском користувачів (наприклад, рекламною

подією) або навіть через помилку, яка спричиняє велику кількість спроб повторити невдалий запит. Ось чому показники необхідно поєднувати з іншими даними для підтвердження, але вони безцінні для раннього виявлення та аналізу тенденцій.

1.2.2 Логи

Логи – це необроблені записи подій, що відбуваються в програмному забезпеченні та системах, з позначками часу. Вони можуть бути структурованими (наприклад, події у форматі JSON) або неструктурованими (звичайний текст) і походити з багатьох джерел:

- журнали доступу до веб-сервера;
- журнали налагодження або помилок програм;
- журнали аудиту баз даних;
- журнали операційної системи (наприклад, журнали автентифікації);
- журнали інструментів безпеки (IDS, WAF тощо).

У сфері безпеки логи часто є основним матеріалом для аналізу, оскільки вони містять багато деталей. Кожен запис у журналі може відповісти на питання – хто і що зробив, коли і звідки це сталося, та який був результат.

Наприклад, один запис у журналі доступу HTTP може показувати, що IP-адреса клієнта 203.16.113.5 запросила GET /admin у момент часу X і отримала відповідь 404. Лог програми може показувати, що користувач «alice@example.com» намагався отримати доступ до ресурсу та отримав відмову через недостатні дозволи у момент часу Y. Журнал бази даних може записати, що обліковий запис «alice» виконав запит SELECT до таблиці Customers у момент часу Z. Шляхом агрегації та співвіднесення цих журналів аналітик може реконструювати підозрілі шляхи користувачів або виявити несанкціонований доступ до даних.

Аналітики кібербезпеки та розслідувачі інцидентів значною мірою опираються на логи. На відміну від метрик, журнали надають конкретність та

описовий характер. Вони важливі для відповіді на питання, що саме сталося. Наприклад, якщо метрики сповіщають про сплеск невдалих входів, журнали показуватимуть імена користувачів, на які було здійснено атаку, вихідні IP-адреси та точні позначки часу, що дозволить відрізнити помилку у роботі сервісу аутентифікації (багато різних облікових записів, що зазнають невдачі один чи кілька раз) від цілеспрямованого перебору (один обліковий запис з багатьма спробами). Як інший приклад журнали веб-доступу в поєднанні з логами налагодження програми можуть показувати послідовність: підозрілий вхід, отриманий веб сервером, помилка, викинута в програмі, та журнал помилок бази даних показує неправильно сформований запит, що вказує на спроби SQL-ін’єкції [28].

Через свою важливість, збої в веденні логів та моніторингу самі по собі є ризиком для безпеки. У рейтингу OWASP Top 10 (2021) «Збої в веденні журналу та моніторингу безпеки» перераховані як основний ризик, зазначаючи випадки, коли відсутність моніторингу дозволяла порушенням залишатися непоміченими протягом місяців або років [2]. Поширені помилки включають відсутність логування ключових подій (входу в систему, дій адміністратора), відсутність моніторингу логів на наявність ознак атаки або зберігання логів лише локально (де зловмисники можуть легко їх змінити) [2]. Ефективне ведення журналів безпеки означає не лише створення логів, а й їх захист та регулярний перегляд. Стандарти відповідності підтверджують це – PCI DSS вимагає ведення журналів аудиту для всіх систем, що обробляють конфіденційні дані, та передбачає регулярний перегляд логів (наприклад, щоденний перегляд журналів подій безпеки) для виявлення аномалій [7]. ISO 27001 також включає засоби контролю для захисту цілісності логів та забезпечення реєстрації та моніторингу дій системного адміністратора для запобігання зловживанням [4].

Логи, будучи настільки детальними, виграють від централізації та кореляції. Агрегація журналів у централізованій системі (часто SIEM – системі управління інформацією та подіями безпеки – або платформі аналітики журналів) є критично важливою для єдиного подання. Як зазначається в блозі про безпеку Elastic,

ефективне використання журналів означає перетворення їх з простого тексту на проактивні засоби моніторингу шляхом їх аналізу в сукупності та виявлення аномалій або закономірностей [8].

Централізоване керування логами дозволяє організаціям налаштовувати правила виявлення для всіх своїх журналів (наприклад, правило для пошуку «20-ти невдалих спроб входу з однієї IP-адреси за 5 хвилин» або «повідомлення про помилку SQL, що з'являється в лозі програми»). Це також гарантує, що журнали зберігаються у захищеному від несанкціонованої зміни способі. Сучасні платформи ведення журналів (комерційні чи з відкритим кодом, такі як ELK/Elastic Stack, Grafana Loki тощо) індексують та структурують дані журналів, що значно пришвидшує пошук та кореляцію. Це особливо корисно під час розслідування інциденту – ви можете швидко переходити між різними джерелами журналів за спільним полем (наприклад, IP-адресою або ідентифікатором сеансу), щоб відстежити дії зловмисника. Приклад такої системи зображено на Рисунку 1.4.

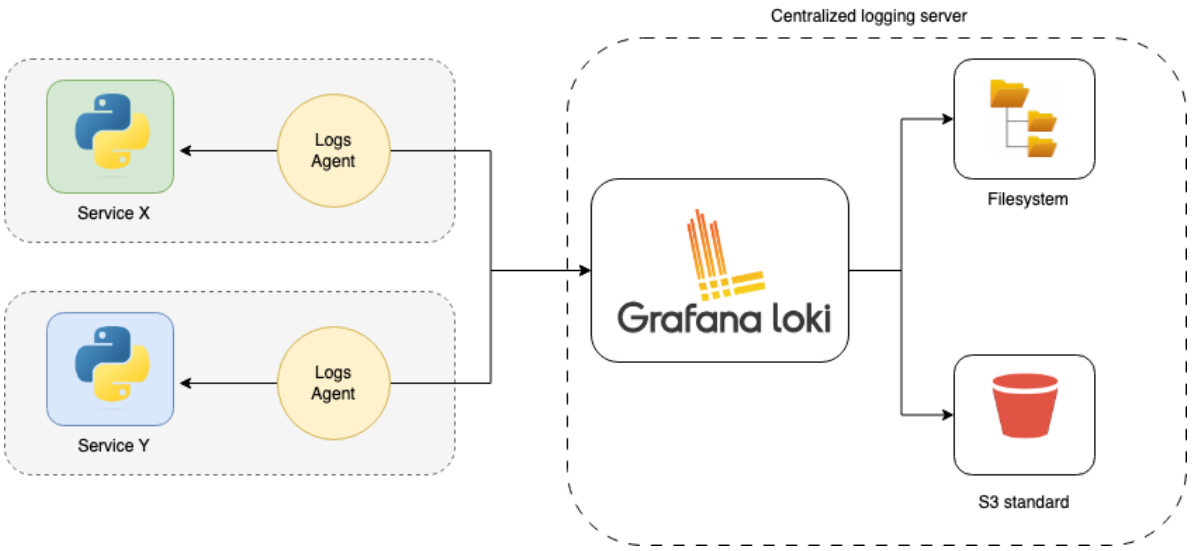

Рисунок 1.4 – Приклад Системи логування, де різні сервіси надсилають логи до централізованого середовища

Grafana Loki виступає в ролі основного сервісу по обробці журналів з зовнішніх сервісів, після чого система зберігатиме їх на фізичному або зовнішньому носії для майбутнього відображення.

1.2.3 Трайси (traces)

Трайси представляють третій стовп спостережуваності – вони відстежують шлях однієї транзакції або запиту, коли він проходить через розподілену систему. У мікросервісах або багаторівневому веб-застосунку одна дія користувача може викликати каскад викликів між сервісами. Розподілене трасування об’єднує їх, призначаючи кожному запиту унікальний ідентифікатор трасування та поширюючи його через усі компоненти. Потім трасування складається з проміжків – сегментів роботи в кожному сервісі, – які разом показують повний шлях цього запиту.

З точки зору продуктивності, трасування використовуються для пошуку сегментів з низькою продуктивністю. Але з точки зору безпеки, трасування може бути надзвичайно корисним для аналізу впливу та проведення розслідувань. Трасування надає причинно-наслідковий контекст, якого може бракувати маючи лише метрики та логи. Розглянемо конкретний приклад: сповіщення вказує на те, що певний HTTP-запит є підозрілим (можливо, він мав дивне наповнення, або надійшов з підозрілої IP-адреси). Якщо ввімкнено розподілене трасування, ви можете взяти ідентифікатор трасування для цього запиту та побачити, що саме сталося всередині програми в результаті цього запиту – які внутрішні служби були викликані, які запити до бази даних були зроблені, скільки часу зайняв кожен крок і де виникли помилки.

Трасування дає змогу відновити повний шлях підозрілого запиту в системі та перевірити, чи не дістався він до чутливих компонентів, доступ до яких не мав би отримати. За трасуванням видно, чи викликався внутрішній мікросервіс лише для адміністратора або чи відбулося пряме звернення до бази даних в обхід очікуваних API.

Також трасування показує, які служби та залежності були задіяні, що допомагає оцінити радіус потенційної загрози, коли запит мав би обмежитися веб-інтерфейсом, але через помилку спричинив виклики внутрішніх сервісів. Окремо можна перевірити, чи проходив запит автентифікацію та авторизацію на

кожному етапі, оскільки в трайсах фіксуються метадані на кшталт ідентифікатора користувача та статусів доступу.

Крім цього, трасування дозволяє виміряти тривалість кроків і помітити аномальні затримки або помилки, що інколи вказує на навмисне навантаження чи збої. Дані про IP-адресу, User-Agent і результати на кожному етапі допомагають прив’язати подію до джерела та зрозуміти, де саме запит був зупинений або завершився успішно.

У публікації «Security Observability: Why Tracing?» пояснюється, що самі лише журнали та метрики часто дають неповну картину безпеки – вони показують, що сталося, але не чому [9]. Розподілене трасування, шляхом додавання контексту та причинно-наслідкового зв’язку, може допомогти визначити, чи була послідовність подій частиною звичайної операції, чи атакою, що тривала [9]. Трасування – це щось на кшталт сюжетної лінії для кожної дії користувача. Вони можуть показати, наприклад, що певний вхідний сигнал спричинив виклики неочікуваної служби, що є тривожним сигналом. Трасування також природним чином пов’язані з журналами. Часто до записів журналу додається ідентифікатор трасування, тому аналітик може зібрати всі логи для цьому трасування, щоб побачити нову інформацію.

Існують конкретні атрибути трасування, що допомагають у сортуванні даних для забезпечення безпеки. Одним з таких, до прикладу, є ідентифікатор трасування. Це унікальний ідентифікатор, що пов’язує всі події однієї транзакції. Він відіграє ключову роль у співвіднесенні між журналами та системами. Іншими важливими атрибутами є користувач або сервіс, IP-адреса, послідовність викликів служб, помилки і коди стану, інформація про час тощо.

Багато систем трасування дозволяють прив’язувати ідентифікатори користувачів до трас (наприклад, який обліковий запис користувача ініціював запит), а списки викликаних мікросервісів можна перевіряти на відповідність очікуванням для цього типу запиту (у випадку, якщо трасування показує виклик платіжної служби, коли користувач просто переглядав сторінку продукту, це підозріло).

Помилки та коди стану можуть свідчити про спробу експлойту (наприклад, примусове виявлення помилок для перевірки на вразливість), а інформація про час може повідомляти про важкі обчислення (можливо, атаку алгоритмічної складності) або тайм-аути (спробу використати атаку стану гонки).

Під час реагування на інциденти, трасування надають, упорядковану за часом, карту шляху зловмисника через систему. Наприклад, припустимо, що виявлено підозрілий HTTP-запит, який намагається здійснити SQL-ін'єкцію. Трасування може показувати, що він звернувся до Служби А (веб-інтерфейс), потім викликав Службу В (API), яка здійснила запит до бази даних, що завершився помилкою. Це повідомляє аналітику, які саме компоненти були задіяні та де проявилася атака (помилка в БД).

Інший сценарій – сповіщення про «витік даних», що о 3:00 ранку відбувся великий експорт даних. Трасування цього запиту на експорт може показати, які нижче стоячі служби використовувалися для збору даних і чи були пропущені належні перевірки. Згідно з підходом одного постачальника послуг по обробці трайсів, кореляція шаблонів атак з розподіленим контекстом трасування може навіть запускати автоматизовані сповіщення безпеки [10]. Наприклад, APM від Datadog може позначати трасування, що відповідають відомим сигнатурам атак (наприклад, шаблону SQL-ін'єкції в запиті), і виводити їх як сповіщення безпеки, поєднуючи дані виконання трасування з правилами виявлення [10].

1.3 Ключові аспекти безпеки на різних рівнях моніторингу вебдодатків

Під час проектування системи моніторингу задля цілей кібербезпеки корисно враховувати різні цілі або рівні, які є підконтрольними, оскільки кожен з них може відповідати на різні питання безпеки. У контексті веб-застосунку можна загалом розрізнити три рівні моніторингу:

– Рівень хоста та системи.

– Стек веб-застосунку (інфраструктура та проміжне програмне забезпечення).

– Рівень застосунку та бізнес-логіки.

Кожен рівень моніторингу надає унікальні дані та допомагає виявляти певні категорії шкідливої активності.

1.3.1 Рівень хоста та ОС

Рівень хоста та ОС охоплює моніторинг фізичних або віртуальних серверів, операційних систем і хостових служб, що лежать в основі веб-застосунку. На цьому рівні аналізуються показники використання системних ресурсів, зокрема навантаження процесора, споживання пам’яті, активність диска та мережевого інтерфейсу. Додатково використовуються журнали операційної системи, такі як:

– Системні логи (наприклад, Linux /var/log/syslog або журнали подій Windows).

– Журнали автентифікації (спроби входу, SSH-з’єднання [26], використання RDP).

– Записи, пов’язані з процесами.

Важливою складовою є контроль запущених процесів і служб, щоб зрозуміти, які саме процеси активні, як вони споживають ресурси, чи відбуваються збої або перезапуски (наприклад, за даними менеджерів процесів). Окремо враховуються сповіщення безпеки на рівні хоста, зокрема повідомлення антивіруса або агента контролю над підозрілою активністю.

З погляду безпеки моніторинг на рівні хоста є критично важливим для виявлення атак, спрямованих безпосередньо на сервер, або для фіксації нецільового використання системних ресурсів. Наприклад, повторні спроби несанкціонованого входу (атаки методом перебору для SSH) відображаються в журналах автентифікації, а різке зростання кількості невдалих входів чи підозрілі блокування облікових записів можуть вказувати на спроби компрометації сервера. Так само раптове високе завантаження процесора на веб-сервері, який зазвичай працює під помірним навантаженням, може бути ознакою

запуску крипто-майнінгу або іншого ресурсозатратного шкідливого процесу після компрометації вебдодатка.

Ще одним важливим напрямом є виявлення незвичних процесів або служб. Якщо зловмисник запускає нову службу чи відкриває підозрілий мережевий порт, це може бути зафіксовано засобами моніторингу хоста, зокрема через системи виявлення вторгнень або аналіз процесів ОС. Показовою ознакою є нетипова поведінка процесів, наприклад коли процес веб-сервера створює інтерактивну shell-сесію або ініціює з'єднання з невідомим зовнішнім хостом, що є підозріло. Додатково про компрометацію можуть свідчити ознаки порушення цілісності хоста, такі як вимкнення програмного забезпечення безпеки, зміни системних бінарних файлів або неочікуване перезавантаження системи.

На практиці для цього рівня використовуються агенти виявлення вторгнень на основі хоста (HIDS) та моніторингу. Такі інструменти, як OSSEC/Wazhuh, auditd у Linux або Sysmon у Windows, створюють журнали про системні виклики та поведінку, які можуть розкрити дії зловмисника.

Рисунок 1.5 – Приклад дашборду моніторингу Хоста

1.3.2 Рівень інфраструктури веб-застосунку

Рівень стеку веб-застосунків (інфраструктура та проміжне програмне забезпечення) охоплює компоненти, які безпосередньо беруть участь в обробці

веб-запиту. Зазвичай це зворотній проксі, який приймає та маршрутизує HTTP-трафік, прикладний рівень, де виконується бізнес-логіка, а також підсистеми даних і інтеграції. На цьому рівні важливо бачити не лише факт надходження запиту, а й те, як він проходить через проксі, застосунок, базу даних і зовнішні сервіси, та де саме виникають помилки або затримки.

До цього рівня зазвичай належать такі компоненти:

– Зворотний проксі, балансувальник навантаження або вхідний обробник (Nginx, Apache, HAProxy або хмарні балансувальники).

– Середовище виконання та сервер застосунків (Java application server, Python Flask/FastAPI, Node.js тощо).

– Підсистеми даних (SQL або NoSQL бази даних, кеші, системи зберігання файлів та інші сховища, що використовуються застосунком).

– Зовнішні інтеграції (виклики зовнішніх API, мікросервісів або сторонніх сервісів, якщо такі є в архітектурі).

Моніторинг цього стеку передбачає збір і кореляцію різних типів даних, щоб відслідковувати працездатність і безпеку кожного компонента та всього ланцюжка обробки запиту. У практичному сенсі це включає журнали, метрики та події, які дозволяють швидко з’ясувати, що саме відбулося, де сталася помилка і чи є ознаки зловмисної активності.

Основні джерела спостережуваності на цьому рівні наступні:

– Журнали проксі і доступу (кожен запит, код відповіді, байти, IP-адреса клієнта, агент користувача тощо).

– Журнали програм (помилки, попередження, користувацькі події безпеки, зареєстровані кодом програми).

– Журнали бази даних та показники продуктивності (повільні запити, невдалі входи до бази даних тощо).

– Метрики з усіх цих компонентів (частота HTTP-запитів, частота помилок, кількість запитів до бази даних тощо).

З погляду безпеки веб-стек часто є першим і найбільш помітним місцем взаємодії зловмисника із системою, тому значна частина атак проявляється саме

в журналах і метриках цього рівня. Під час сканування та перерахування ресурсів зловмисники зазвичай генерують підвищену кількість помилок 404 або 400, звертаючись до неіснуючих сторінок чи надсилаючи некоректно сформовані запити. Це можна помітити в логах проксі або застосунку, зокрема коли фіксується багато запитів до типових шляхів на кшталт /admin, /phpmyadmin, /wp-login.php навіть тоді, коли відповідні компоненти в системі не використовуються.

Спроби ін’єкцій також нерідко залишають характерні сліди в телеметрії. У разі SQL-ін’єкції в логах застосунку або БД можуть з’являтися синтаксичні помилки, а в рядках запитів або параметрах можуть траплятися підозрілі шаблони на кшталт ' OR '1'='1. Аналогічно атаки XSS [30] можуть проявлятися як нетипові фрагменти вхідних даних, зокрема HTML-теги або підозрілі конструкції, що потрапляють у журнали запитів чи подій валідації. В окремих випадках зловмисник може навмисно провокувати винятки в застосунку, надсилаючи неочікувані або граничні значення, а кореляція таких помилок із аномальними параметрами запитів може вказувати на спроби експлуатації вразливостей.

Окрему категорію складає зловживання автентифікацією та API. Масові спроби підбору часто відображаються як багато входів до одного й того ж облікового запису з різних IP-адрес або географічно несумісних локацій. Якщо на рівні проксі або застосунку увімкнено обмеження швидкості, то агресивне використання API чи автоматизована активність може супроводжуватися збільшенням кількості відповідей HTTP 429 і відповідних записів у журналах, що є корисним індикатором бот-активності або спроб виснаження ресурсів.

Наведені приклади узгоджуються з підходами, описаними в OWASP Web Security Testing Guide, де розглядаються типові класи веб-атак (зокрема SQLi, XSS, CSRF) і підкреслюється роль належного журналювання та моніторингу як практичного механізму їх виявлення [2]. У цьому контексті важливими маркерами можуть бути не лише коди помилок і винятки застосунку, а й аномальні шаблони URL, параметрів і запитів, які можуть бути зафіксовані веб-

сервером або проксі [2].

1.3.3 Рівень застосунку та бізнес-логіки

Рівень застосунку та бізнес-логіки є найбільш деталізованим шаром моніторингу, який зазвичай налаштовують під конкретну функціональність системи. Якщо нижчі рівні переважно фіксують технічні факти на кшталт HTTP-запитів, кодів відповіді або звернень до бази даних, то тут акцент зміщується на зміст і контекст дій у домені застосунку. Для цього в коді або конфігурації застосунку передбачають інструментування, яке генерує події та записи журналів, пов'язані з безпекою і поведінкою користувачів, у формі зрозумілих семантичних подій.

На цьому рівні найчастіше фіксуються:

– Дії користувача (наприклад, зміна пароля, виконання переказу коштів, оновлення контактних даних).

– Бізнес-транзакції (оформлення замовлення, створення рахунку, підтвердження платежу).

– Безпекові операції в межах застосунку (зміна ролей і прав доступу, відхилені спроби доступу до обмежених функцій, підозрілі сценарії автентифікації).

– Журнали аудиту, які показують, хто і що зробив у домені застосунку, що часто потрібно для дотримання вимог у фінтех, медицині та інших регульованих сферах.

Щоб такий моніторинг працював, застосунок має створювати події для значущих дій і доповнювати їх контекстом – ідентифікаторами користувачів, ролями, атрибутами сесії, результатом операції, а інколи й додатковими бізнес-параметрами. На відміну від загальних журналів веб-сервера, ці події є специфічними для домену. Наприклад, сайт електронної комерції може зареєструвати «USER alice@example.com PLADED ORDER ID 987654, TOTAL $123.45». З точки зору безпеки, це саме по собі є нормальним явищем, але маючи

більший контекст завдяки моніторингу, можна співвідносити такі події, та виявити аномалії, такі як раптове розміщення Алісою 100 замовлень за хвилину (що може свідчити про автоматизовану атаку або зловживання).

З погляду кібербезпеки саме моніторинг бізнес-логіки допомагає виявляти зловживання легітимною функціональністю та проблеми з авторизацією, які на нижчих рівнях можуть не виглядати як явна атака. Технічно запити можуть бути коректними і проходити без помилок, але їхній зміст і частота, невідповідність ролям, нетипові послідовності дій або порушення бізнес-правил створюють ознаки інциденту, які видно лише на рівні доменних подій.

Наведемо приклад зловживання функціональністю. Припустимо, що зловмисник знаходить спосіб багаторазово ініціювати грошовий переказ у банківському додатку (без досягнення обмежень) – технічно кожен переказ є дійсною дією, тому нижчі рівні бачать відповіді «200 OK», але рівень бізнес-логіки виявляє, що один користувач ініціює надзвичайно великий обсяг або вартість транзакцій за короткий час. Це може сигналізувати про шахрайство або зловживання.

Інший приклад з захопленням облікового запису та зловживання привілеями. Якщо зловмисник скомпрометує обліковий запис користувача, його поведінка в додатку може змінитися. Бізнес-журнали можуть показати, що обліковий запис отримує доступ до даних або виконує дії, нетипові для цього користувача (наприклад, завантажує всю історію транзакцій, коли зазвичай вони переглядають її лише на екрані). Якщо ми реєструємо «користувач X переглянув 100 записів клієнтів», це можна позначити як відхилення від звичайного використання. Аналогічно, якщо звичайний обліковий запис користувача раптово намагається виконати функцію лише для адміністратора (і, можливо, зазнає невдачі), це варте уваги.

Також потрібно звертати увагу на багатоетапні атаки. Деякі логічні недоліки вимагають покрокового виконання дій. Моніторинг дій користувача послідовно може виявити такі речі, як спроба примусового перегляду (доступ до неавторизованої URL-адреси) або обхід робочого процесу. Наприклад, якщо

застосунок очікує, що користувачі переглянуть сторінку 1, потім 2, а потім 3 у робочому процесі, але зловмисник переходить одразу до 3 (пропускаючи перевірки на 2), спеціальний журнал може зафіксувати аномальну послідовність.

Рисунок 1.6 – Приклад дашборду з патернами логів веб додатку

Для дослідження такої поведінки аналітики кібербезпеки часто звертаються до журналів подій, які можна відфільтрувати по окремому користувачу, чи сервісу. На рисунку 1.6 наведено приклад інтерфейсу Grafana для перегляду логів для окремого сервісу.

РОЗДІЛ 2 ІНСТРУМЕНТИ, АРХІТЕКТУРИ ТА МОДЕЛІ РОЗГОРТАННЯ ПЛАТФОРМ МОНІТОРИНГУ

2.1 Загальна архітектура платформ моніторингу вебдодатків

Сучасна платформа моніторингу та логування зазвичай будується як набір взаємопов'язаних компонентів, які разом забезпечують збір, доставку, зберігання, аналіз і представлення телеметрії. Розуміння цієї архітектури важливе не лише з точки зору експлуатації, а й для виконання вимог безпеки, зокрема щодо надійності, цілісності даних і стійкості до несанкціонованого доступу. Як правило, система починається зі збору телеметрії безпосередньо на джерелах, далі дані транспортуються через конвеєр доставки, потрапляють у спеціалізовані сховища, після чого стають доступними для запитів, візуалізації та автоматизованого сповіщення.

На першому етапі використовуються збирачі даних або агенти, які встановлюються на серверах, у контейнерах або інтегруються в застосунки. Вони відповідають за отримання метрик, логів і трайс. Для метрик це можуть бути Prometheus Node Exporter або клієнт StatsD, для журналів часто застосовують Filebeat або Grafana Promtail, а для трасування – інструментування на основі OpenTelemetry SDK чи APM-агенти, вбудовані в код. Залежно від реалізації агенти або самі надсилають дані в центральну систему, або центральна система періодично забирає їх, наприклад шляхом опитування endpoint-ів метрик.

Зберігання телеметрії зазвичай розділяють за типами даних і використовують спеціалізовані бекенди сховищ:

– TSDB на кшталт Prometheus або Graphite для метрик.

– Elasticsearch/OpenSearch або Loki для логів.

– Jaeger чи Tempo для трайс.

Над ними працює рівень обробки та аналізу, який формує правила сповіщень (Prometheus Alertmanager, Grafana alerting, ElastAlert), а за потреби виконує кореляцію подій на рівні SIEM і виявлення аномалій.

Доступ до даних забезпечує рівень візуалізації (Grafana, Kibana), де користувачі будують дашборди та проводять розслідування, а підсистема оповіщень доставляє алерти в пошту, месенджери, Slack, PagerDuty тощо. Оскільки телеметрія може містити чутливі відомості, платформа повинна підтримувати керування доступом і RBAC, обмежуючи перегляд і зміни дашбордів та правил сповіщень, зазвичай через вбудовані механізми Grafana або Kibana.

Що стосується архітектури, багато систем використовують центральний «кластер моніторингу», куди надсилаються всі дані. Такий централізований підхід важливий для моніторингу безпеки. Агреговане сховище логів та показників необхідне для захисту від несанкціонованого доступу. Якщо зловмисник скомпрометує один сервер, він може спробувати стерти локальні логи, але якщо ці логи були відправлені на віддалену систему в режимі реального часу, зловмисник не зможе легко приховати свої сліди. Таким чином, один із принципів полягає в тому, що журнали слід надсилати з хоста якомога швидше для безпечного зберігання. ISO 27001 підкреслює важливість захисту інформації журналів від несанкціонованого доступу або зміни – централізована система реєстрації зі суворим контролем доступу допомагає досягти цього [4]. Аналогічно, стандарт PCI DSS рекомендує централізоване ведення журналу. Це треба для того, що у разі порушення роботи системи, журнал аудиту зберігався в іншому місці (а також щоб можна було співвідносити події між системами).

Захист від несанкціонованого доступу можна додатково підвищити за допомогою таких заходів, як одноразове зберігання даних або перевірка криптографічної цілісності журналів. Деякі системи додають спеціальні дані до файлів логів, які потім відправляються. Інші системи надсилають подію за подією безпосередньо до індексу. Все це забезпечує те, що після потрапляння даних до системи моніторингу зловмисник не може їх ретроактивно змінити (принаймні, не без виявлення). Такі технології, як «immutability flags» або логи, доступні лише для доповнення, подібні до блокчейну, використовуються у середовищах з високим рівнем безпеки для захисту цілісності журналів.

Ще один аспект – це висока доступність і масштабованість – система моніторингу повинна обробляти навантаження (потенційно мільйони подій на день) і залишатися доступною навіть під час інцидентів (коли навантаження може різко зрости). Як не парадоксально, але під час інциденту безпеки працівники та аналітики найбільше покладається на свою систему моніторингу, тому вона має бути надійною та відмомостійкою.

З точки зору моніторингу безпеки, можна інтегрувати платформу спостереження зі спеціалізованим робочим процесом SIEM або SOC. Наприклад, сповіщення від системи моніторингу можна пересилати до системи видачі заявок SOC, або журнали можна паралельно надсилати як до інструменту спостереження, так і до інструменту аналітики безпеки. Багато сучасних платформ розмивають межу між спостереженням та безпекою (Elastic Security, Splunk тощо, обробляють усі види телеметрії). Архітектура повинна враховувати, хто використовує дані – розробники, оператори та команди безпеки можуть використовувати одні й ті ж дані, але по-різному.

Доступ базуючись на ролі користувача системи в межах централізованої системи також є ключовим, якщо платформою користуються велика кількість осіб. Не доцільно, щоб кожен розробник бачив журнали безпеки, які йому не слід бачити, або щоб скомпрометований обліковий запис розробника перейшов у систему моніторингу та зібрав конфіденційну інформацію. Реалізація дозволів на читання або запис для кожного індексу або типу даних є поширеною. Наприклад, Elastic та Grafana можуть обмежувати доступ до певних індексів та джерел даних і редагування панелі інструментів за допомогою ролей.

Контроль A.12.4.2 стандарту ISO 27001 вимагає захисту інформації журналу, що в термінології архітектури означає захист центрального сховища журналів, контроль доступу та запобігання несанкціонованому втручанню [4].

2.2 Моніторинг на основі SaaS

Багато організацій зараз використовують пропозиції «програмне забезпечення як послуга» (SaaS) для своїх потреб моніторингу та ведення журналів. До них належать такі сервіси, як Datadog, Splunk Cloud, New Relic, Grafana Cloud, Elastic Cloud тощо. Рішення для моніторингу SaaS означає, що постачальник надає платформу (сховище, інтерфейс користувача, механізми аналізу) як хмарний сервіс, а користувачеві потрібно лише встановити агентів або надіслати свою телеметрію до хмари.

Використання платформи моніторингу у форматі SaaS зазвичай дає змогу швидкого розгортання та зменшення витрат на супровід, оскільки не потрібно самостійно адмініструвати складний стек зі сховищами, обробкою та масштабуванням. Постачальник бере на себе оновлення, підтримку інфраструктури та керування продуктивністю, а команда може одразу перейти до налаштування дашбордів, алертів і політик спостережуваності.

Додатковою перевагою є широкий набір інтеграцій і готових сценаріїв. SaaS-рішення часто мають попередньо зібрані панелі, типові правила сповіщень і автоматичне виявлення сервісів у хмарі, що скорочує час до отримання корисних результатів. У багатьох таких платформах доступні й більш серйозні можливості, які складно підтримувати невеликим командам власними силами, наприклад алгоритми виявлення аномалій або ML-функції для аналізу телеметрії. Як приклад, Grafana Cloud надає вбудовані можливості для виявлення аномалій у метриках (зокрема через функції на кшталт “Asserts”), які можуть автоматично відмічати відхилення в поведінці системи [11].

Окремо варто відзначити зручність масштабування та доступу. Хмарний сервіс зазвичай легше розширюється під великі обсяги даних, а команда може безпечно працювати з дашбордами та сповіщеннями з будь-якого місця. Крім того, SaaS-постачальники регулярно додають нові функції, включно з інструментами на базі ШІ для допомоги в пошуку першопричин за логами і

трайсами, і ці оновлення не потребують від клієнта складних міграцій або ручного оновлення компонентів [12].

Водночас у середовищах із чутливими даними важливо зважати на ризики та обмеження такого підходу. Ключові моменти, які зазвичай потребують оцінки, наступні:

– Логи й метрики можуть містити конфіденційну інформацію, а передавання в хмару постачальника підвищує вимоги до безпеки.

– Регуляторні норми можуть вимагати зберігання даних у конкретних регіонах [13].

– Модель оплати за зберігання може різко дорожчати, особливо для логів, які швидко ростуть до великих обсягів.

– Прив’язка до специфічних функцій, форматів чи мов запитів ускладнює міграцію, що спричиняє прив’язку до постачальника.

– Обмеження кастомізації і проблеми з інтеграціями нестандартних open-source компонентів.

Для високочутливих або засекречених середовищ (наприклад, військової чи критичної інфраструктури) багато хто не погоджується на SaaS для логів. Вони віддають перевагу локальним рішенням, щоб гарантувати відсутність розкриття даних. Навіть у межах підприємств часто виникають дилеми: використовувати хмарний сервіс моніторингу чи розгортати власний стек.

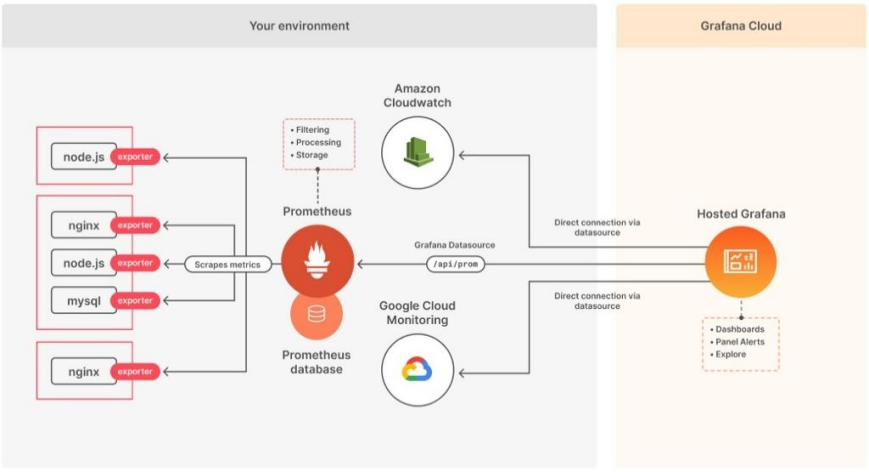

Рисунок 2.1 – Приклад архітектури з SaaS продуктом за межами підконтрольного середовища

Постачальники SaaS усвідомлюють ці проблеми. Багато хто з них наголошує на своїх заходах безпеки (шифрування, сертифікати відповідності, такі як SOC2, ISO 27001 для власних операцій). Деякі пропонують гібридні варіанти (наприклад, локальний збірник даних, який надсилає лише агреговані дані).

Підсумовуючи, моніторинг SaaS може значно пришвидшити впровадження належних практик моніторингу та часто забезпечує складні можливості виявлення «з коробки». Для багатьох компаній, особливо стартапів або тих, що не мають важкого регуляторного тягаря, переваги переважають ризики. Але для інших ідея передачі всіх логів програм третій стороні є неприйнятною. Ці організації схиляються до самостійно розміщених рішень, які буде розглянуто далі.

2.3 Стек з відкритим вихідним кодом та власним хостингом (фокус на екосистемі Grafana)

Для організацій, які обирають самостійне розміщення, існує багатий ландшафт інструментів з відкритим кодом для створення платформи спостереження. Однією з популярних комбінацій є екосистема Grafana разом з іншими проектами CNCF (Cloud Native Computing Foundation), які часто називають стеком «LGTM»:

– Loki для журналів;

– Grafana для панелей інструментів;

– Tempo для трасування;

– Prometheus для метрик (власна назва Prometheus не відповідає абревіатурі, іноді «P» замінюється на «Graphite» у старіших термінах, але Prometheus є фактичною системою метрик OSS).

Prometheus використовується для збору, зберігання та аналізу метрик, а також для побудови правил сповіщень. Це система моніторингу з відкритим кодом, яка працює як база даних часових рядів і періодично опитує

інструментовані сервіси через HTTP, отримуючи від них показники продуктивності та стану. Для роботи з цими даними Prometheus має мову запитів PromQL, яка дозволяє виконувати агрегації, фільтрацію та обчислення похідних метрик, що робить її придатною не лише для експлуатаційних задач, а й для безпекових сценаріїв, де важливо відслідковувати відхилення та різкі зміни поведінки.

В контексті безпеки Prometheus може збирати метрики, такі як кількість HTTP-запитів, кількість помилок, завантаження процесора тощо. У Alertmanages можна визначити правила сповіщень, такі як «коефіцієнт помилок 5xx > N протягом 5 хвилин» або «Завантаження процесора > 90% протягом 10 хвилин», і Alertmanager оброблятиме надсилання сповіщень. Prometheus може масштабуватися за допомогою віддаленого сховища даних, але зазвичай для різних робочих навантажень або для шардування даних запускаються окремі екземпляри сервісу. Для більшого масштабування (або тривалого зберігання) використовуються такі проекти, як Cortex або Mimir (від Grafana Labs) – вони, по суті, є розподіленими серверними частинами Prometheus.

2.3.1 Loki (Журнали)

Loki – це система агрегації журналів від Grafana Labs, розроблена для економії та простоти використання. Її ще часто називають «Prometheus для журналів», оскільки вона спрямована на індексацію журналів за мітками, а не за повним текстом, щоб зменшити розмір індексу. Loki складається з інжестерів, дистриб’юторів, запитувачів та сховища індексів. Вона працює з Promtail, агентом для збору журналів із серверів, та надсилає їх до Loki. Loki зберігає журнали та дозволяє обробляти їх за допомогою мови запитів LogQL, яка виглядає як PromQL, але для журналів. Одна велика перевага полягає в тому, що існує можливість додавання до журналів тих же міток, що й до метрик (наприклад, назва програми, сервер тощо), що спрощує кореляцію. Loki за замовчуванням не індексує повний вміст журналів, зосереджуючись на мітках

(наприклад, джерело, ім’я файлу тощо), що робить її дешевшою для роботи у великих масштабах. Для моніторингу безпеки Loki може централізувати журнали з веб-серверів, програм, ОС тощо. Потім можна використовувати запити або сповіщення на Loki, де Grafana може розглядати Loki як джерело даних і навіть сповіщати про нього. У блогах Grafana Labs було показано використання Loki для виявлення загроз безпеці, наприклад, поєднання Loki з форматом правил Sigma для виявлення відомих загроз у даних журналів [14]. Легкість використання Loki та його відкритий вихідний код роблять його зручним для тих, хто уникає Splunk або Elastic через вартість або складність.

2.3.2 Tempo (Трасери)

Tempo – це розподілений сервер трасування Grafana, який є масштабованим і може приймати трайси у форматах, таких як Jaeger, Zipkin або OpenTelemetry. Він розроблений для дешевого зберігання великої кількості трайс шляхом індексації лише ідентифікаторів трайс (не всіх даних span). Суть полягає у знаходженні ідентифікаторів трайс за допомогою журналів або метрик з подальшим отриманням повної інформації. Tempo дозволяє пов’язувати дані трасування з Grafana. Як приклад, користувач може клацнути запис у журналі, який має ідентифікатор трасування, щоб побачити повну інформацію про запит.

У сфері безпеки Tempo можна використовувати для зберігання трасування, що пізніше може бути використано під час аналізу інцидентів (наприклад, отримання трасування підозрілого запиту). Його рідше безпосередньо використовують для сповіщень у реальному часі, але частіше для глибокого аналізу.

2.3.3 Grafana – інтерфейс візуалізації та сповіщень

Grafana – це панель інструментів, де можна створювати діаграми з метрик Prometheus, таблиці з журналів Loki та розподілені трасування з Tempo. Grafana

також має систему сповіщень (уніфіковані сповіщення), яка може оцінювати запити до будь-якого джерела даних та надсилати сповіщення. Grafana надає операторам уніфіковане джерело інформації та може інтегруватися з панелями інструментів безпеки. Користувач може створити панель, що відображає найпопулярніші IP-адреси джерел за кількістю помилок, або панель, що відображає останні сповіщення безпеки з журналів. Grafana підтримує керування доступом на основі ролей та може інтегруватися з системами автентифікації наприклад з OAuth. Це дозволяє визначати, хто і які панелі інструментів може бачити. Проте, точніший RBAC на рівні даних може залежати від дозволів базового джерела даних.

2.3.4 Експортери та агенти

Для отримання даних у Prometheus та Loki використовуються експортери та агенти. Серед експортерів варто виділити наступні:

– node_exporter, що використовується для системних метрик;

– blackbox_exporter для збирання інформації з кінцевих ендпоінтів;

– експортери для конкретних програм, наприклад різних баз даних як от MongoDB exporter, тощо.

Для журналів можна встановити Promtail на кожному хості або всередині кожного контейнера який буде зчитувати файли логів, позначати їх (наприклад, назва програми, хост) та надсилаити до Loki.

Для трайс в код програми інтегрується OpenTelemetry SDK, яка експортує діапазони до колектора, та надсилає до Tempo. У самостійно розміщеному стеку користувач має повний контроль над розгортанням цих агентів та їх налаштуванням за потреби. Користувач також може використовувати Elastic Beats (Filebeat/Metricbeat) при інтеграції з Elastic або Fluentd/FluentBit для пересилання журналів – існує багато варіантів з відкритим кодом, і їх часто можна змішувати. Наприклад, надсилати журнали як до Loki, так і до Elastic, якщо потрібно.

Самостійно розміщений стек Grafana/Prom/Loki/Tempo охоплює метрики, журнали та трасування – три основні складові – за допомогою відкритих компонентів. Зазвичай їх запускають як контейнери Docker або поди Kubernetes, або на віртуальних машинах. Prometheus може зберігати дані на локальному диску або віддаленому сховищі, а Loki може використовувати об'єктне сховище, таке як S3, для фрагментів даних. Усі компоненти піддаються масштабуванню. До прикладу, Loki та Tempo створені для кластеризації для високого масштабування, Prometheus можна шардувати, або ж можна використовувати Grafana Mimir (бекенд метрик з відкритим кодом на основі Cortex) для масштабованих метрик.

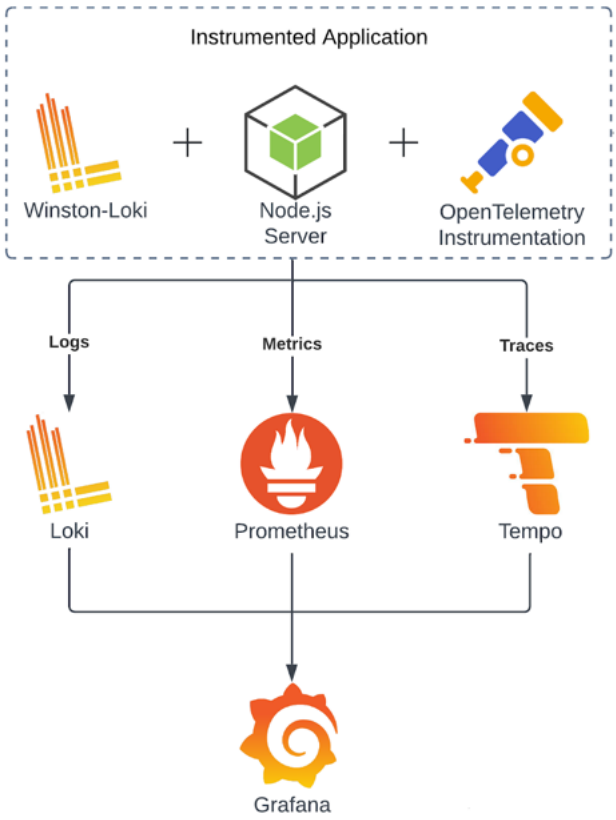

Рисунок 2.2 – Візуалізація самостійно розміщеного стеку з відкритим вихідним кодом

У продемонстровану на рисунку 2.2 систему за потреби можна додатково інтегрувати Elasticsearch або OpenSearch для індексації журналів, навіть якщо

базовим сховищем логів є Loki. Такий підхід корисний, коли потрібен повнотекстовий пошук, складніші запити або коли в організації вже використовується стек ELK. У цьому випадку Elasticsearch може виконувати роль довготривалого сховища та “озера даних” для логів, а Grafana може підключатися до нього як до джерела даних для візуалізації. Практики застосування Elasticsearch для аналітики безпеки, включно з правилами виявлення та ML-задачами для пошуку загроз, описуються в матеріалах Elastic [15]. Часто використовується комбінований сценарій, де Loki призначається для швидких і дешевших інфраструктурних логів, а Elasticsearch для критичних журналів безпеки з глибшим аналізом.

Окремим напрямом є інтеграція з процесами SIEM/SOC, коли сповіщення з Prometheus або Grafana передаються в SIEM чи SOAR для централізованої кореляції та керування інцидентами. Наприклад, алерт може надсилатися у вебхук, який далі використовується SOAR-платформою для автоматичного створення заявки, збагачення контекстом і запуску стандартного сценарію реагування.

Також до стеку спостережуваності логічно підключати WAF та IDS, оскільки ці засоби генерують журнали і сповіщення про підозрілу активність. Їхні події можна збирати в Loki або Elastic і відображати в Grafana, щоб отримати єдине поле огляду для операційної та безпекової телеметрії. Наприклад, журнали ModSecurity можна корелювати з логами застосунку та проксі, що допомагає точніше відтворювати ланцюжок подій і швидше розуміти причину спрацювання.

Акцент на самостійному розміщенні, особливо в контексті таких стандартів, як PCI DSS та ISO 27001, полягає в тому, щоб зберігати конфіденційні журнали в приміщенні та під суворим контролем. Самостійне розміщення дозволяє дотримуватися вимог щодо контролю доступу до логів та інших даних, та гарантувати, що журнали не потрапляють до неавторизованих хмарних сервісів. Це також може сприяти створенню середовища з обмеженим доступом (без Інтернету). PCI DSS, наприклад, сприятиме архітектурі, де інформація про дані

власників карток знаходиться у внутрішній системі з обмеженим доступом. ISO 27001 вимагатиме, щоб сама централізована система моніторингу була захищена та контролювалася на предмет несанкціонованого доступу, оскільки вона стає ціллю – зловмисники можуть спробувати отримати доступ до журналів або спробувати видалити їх.

Самостійно розміщений стек за потреби може бути повністю ізольованим, тобто працювати офлайн без зовнішніх підключень, наприклад у закритих військових мережах. Компоненти Grafana, Prometheus, Loki та Tempo підтримують роботу в автономному режимі. У такому підході можуть бути відсутні можливості на кшталт керованих оновлень або хмарних функцій на базі штучного інтелекту, однак зберігається повний суверенітет даних.

Такий варіант передбачає компроміси, оскільки організація бере на себе розгортання та підтримку стеку, включно з оновленнями й питаннями масштабування. Водночас перевагою залишається повний контроль над даними та потенційна економія за великих обсягів, оскільки витрати зводяться переважно до інфраструктури без оплати ліцензій або тарифів.

На практиці поширеним є гібридний підхід, коли рішення з відкритим кодом використовуються для частини журналів або метрик з метою збереження контролю та оптимізації витрат, а окремі функції спостережуваності реалізуються через SaaS. Така модель дозволяє поєднати автономність і гнучкість self-hosted розгортання з перевагами керованих сервісів там, де це виправдано.

Для самостійного хостингу критичною є безпечна конфігурація компонентів і каналів обміну даними. Доцільним є застосування TLS для веб-інтерфейсів (наприклад, Grafana або Kibana, якщо використовується Elastic), налаштування автентифікації та сегментація мережі моніторингу, щоб лише авторизовані системи могли передавати телеметрію і зменшувався ризик ін’єкції підроблених журналів. Додатково важливим є моніторинг самої платформи спостережуваності, зокрема метрик Prometheus і журналів системи логування, що дає змогу вчасно виявляти збої або ознаки компрометації.

2.4 Гібридні моделі

Багато організацій застосовують гібридний підхід до моніторингу, поєднуючи простоту SaaS із контролем, який забезпечує власний хостинг. Така модель зазвичай реалізується через розділення потоків телеметрії або за типом даних, або за середовищами та системами. Поширеним прикладом є передавання метрик у хмарний сервіс, оскільки вони часто менш чутливі та мають відносно менший обсяг, тоді як журнали зберігаються локально через їхню конфіденційність або високу вартість передачі великих масивів даних. Інший варіант полягає у використанні SaaS для менш критичних середовищ або корпоративних IT-систем, а для виробничих систем із даними клієнтів застосовується self-hosted розгортання.

Окремо зустрічаються схеми з повним дублюванням, коли телеметрія надсилається одночасно до локальної системи та до хмарної копії. Це підвищує стійкість у разі відмови одного з сервісів і може спрощувати залучення зовнішніх партнерів, наприклад керованих сервісів безпеки для цілодобового моніторингу, при збереженні локального контролю. Також використовується підхід, за якого дані зберігаються в хмарі, але шифруються ключами, що залишаються під контролем організації, що частково зменшує ризики та покращує відповідність вимогам, не відмовляючись від хмарної обробки.

Попри переваги, гібридний підхід створює додаткові виклики. Зростає складність керування двома системами та підтримки інтеграцій, а також виникає ризик фрагментації, коли дані розподілені між різними інструментами і втрачається цілісне уявлення без активної кореляції. Ускладнюється й розслідування інцидентів, оскільки контекст у хмарних метриках може вимагати підтвердження у локальних журналах, що не завжди відбувається без впливу. Часто гібридність диктується оптимізацією витрат, коли дорогі хмарні платформи використовуються лише для найбільш важливих даних або функцій аналітики, а великі масиви логів зберігаються в дешевших власних сховищах. Водночас така модель може допомагати виконувати внутрішні політики контролю даних, наприклад коли журнали з персональними даними не

залишають периметр, а менш конфіденційні операційні метрики обробляються в хмарі.

З точки зору безпеки, гібридні моделі вимагають чіткої політики щодо того, які дані можуть потрапляти в хмару, та забезпечення того, щоб усе, що потрапляє в хмару, було належним чином анонімізовано, якщо це необхідно. Наприклад, можна видалити ідентифікаційну інформацію з метрик і замість надсилання повних URL-адрес можна надсилати лише назви кінцевих точок. Деякі агенти моніторингу мають функції для редагування або фільтрації даних перед надсиланням до SaaS. Наприклад, не надсилати номери кредитних карток у журналах.

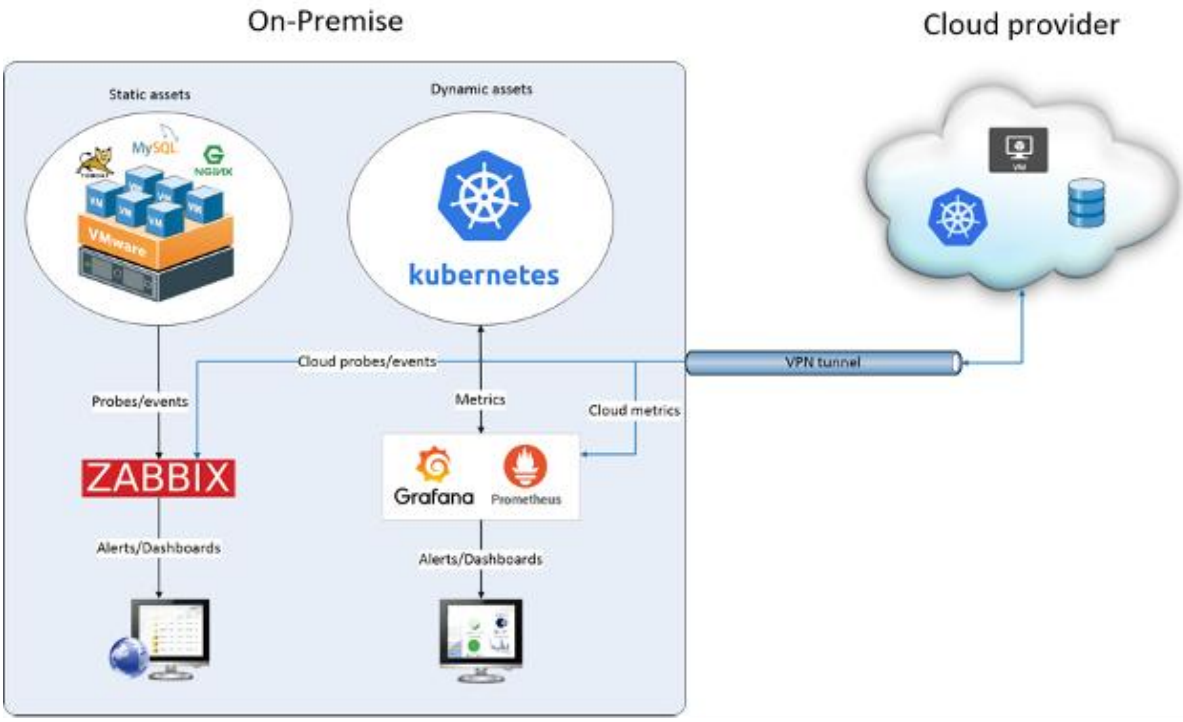

Рисунок 2.3 – Приклад гібридної архітектури моніторингу [16]

Вибір гібридного хостингу часто є також перехідним станом – компанії можуть почати використовувати власний хостинг, а потім почати використовувати деякі хмарні функції, або навпаки. Його також можна використовувати як стратегію міграції і залишати обидва, доки не буде повна впевнені в одному.

2.5 Методи розгортань систем моніторингу з акцентом на безпеку

Щоб система моніторингу вважалася безпечною в контексті вебдодатків та кібербезпеки, вона повинна відповідати набору загальних вимог, що виходять за рамки базової спостережуваності. По-перше, необхідно зберегти цілісність та автентичність даних моніторингу, щоб логи, метрики та трайси можна було використовувати, як докази під час розслідувань та аудитів. По-друге, необхідно підтримувати конфіденційність, оскільки дані моніторингу часто містять конфіденційну технічну та бізнес-інформацію, включаючи ідентифікатори користувачів або часткові чутливі дані. По-третє, доступність та стійкість є важливими, адже система повинна продовжувати збирати та представляти дані навіть у разі несправності або під час атаки, інакше персонал служби безпеки фактично не бачить нічого. Нарешті, архітектура моніторингу повинна підтримувати відстеження та підзвітність, наприклад, забезпечуючи узгодженість усіх відповідних подій у часі та пов’язаність з ідентифікованими суб’єктами, а також відповідати застосовним системам відповідності, таким як рекомендації ISO 27001, PCI DSS та OWASP. Наведені нижче практики перетворюють ці загальні вимоги на конкретні варіанти проектування та конфігурації для безпечного моніторингу вебдодатків.

До ключових тем безпеки під час розгортання моніторингу належать цілісність і незмінність журналів, оскільки вони використовуються під час реагування на інциденти та аудитів. Це досягається моделлю зберігання з режимом лише додавання, жорстким розмежуванням прав на запис і, за потреби, додатковою верифікацією на основі ланцюгового хешування.

Коректна кореляція подій вимагає точної синхронізації часу, тому інфраструктура має використовувати спільне джерело NTP для серверів, контейнерів і мережевих пристроїв. Доступ до даних моніторингу повинен бути обмежений через SSO, MFA та RBAC, а передавання метрик і журналів між агентами та бекендами має виконуватися через TLS з автентифікацією, щоб зменшити ризики перехоплення або ін’єкції підроблених записів.

Щоб знизити витоки через телеметрію, доцільними є мінімізація та маскування чутливих даних у логах, зокрема секретів, токенів і платіжних реквізитів. Надійність підвищується моніторингом самої платформи спостережуваності, включно з алертами на збої агентів, падіння обсягу логів або інші ознаки деградації, а також налаштуванням сповіщень із пріоритизацією та приглушенням шуму. Додатково враховуються політики й правові вимоги до зберігання, а працездатність конфігурації періодично перевіряється контрольованими симуляціями атак або тестовими скануваннями.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ МОНІТОРИНГУ ВЕБДОДАТКІВ

3.1 Використані технології, засоби та утиліти

Дана робота використовує Docker та Docker Compose для реалізації інтегрованого середовища моніторингу, в якому збір, транспортування, зберігання та візуалізація розглядаються як єдина система. Стек всієї системи розгортається як набір контейнерів, з'єднаних двома виділеними мережами: одна для зовнішнього маршрутизованого доступу до сервісів (monitoring-network) та одна для внутрішнього обміну телеметрією (telemetry-network). Таке розділення зменшує зв'язок між вхідним трафіком та трафіком моніторингу та забезпечує чітке розділення зв'язку на площині даних та на площині керування.

Стек моніторингу побудований навколо широко поширених, сумісних сервісних компонентів які згадувались раніше. Prometheus забезпечує сховище часових даних та модель збору на основі витягування, в якій спостережувані об'єкти надають метрики через протокол HTTP та періодично очищуються за фіксованим графіком. Loki служить сховищем логів, індексованим мітками, який приймає та безперервно обробляє логи, а також дозволяє виконувати пошукові запити, що зберігаються в корисних даних журналів JSON. Tempo забезпечує зберігання та пошук розподілених трайсів, причому дані представлені як проміжки на різних рівнях та співвідносяться за допомогою ідентифікаторів поширеного контексту. Grafana розташована над цими компонентами як єдиний інтерфейс запитів та візуалізації, дозволяючи операторам виконувати запити одразу до всіх доступних джерел.

Також присутній новий раніше не згаданий компонент – Grafana Alloy. Він функціонує як шлюз телеметрії та точка нормалізації, надаючи доступ приймачам OpenTelemetry Protocol (OTLP) через HTTP та gRPC. Приймаючи телеметрію, що надсилається додатками, Alloy пересилає трайси до Tempo, а журнали до Loki. Alloy також підтримує збір метрик шляхом парсингу вибраних цілей та пересилання зразків до Prometheus через інтерфейс приймача

віддаленого запису. Приблизну схему взаємодії сервісів можна побачити на рисунку 3.1.

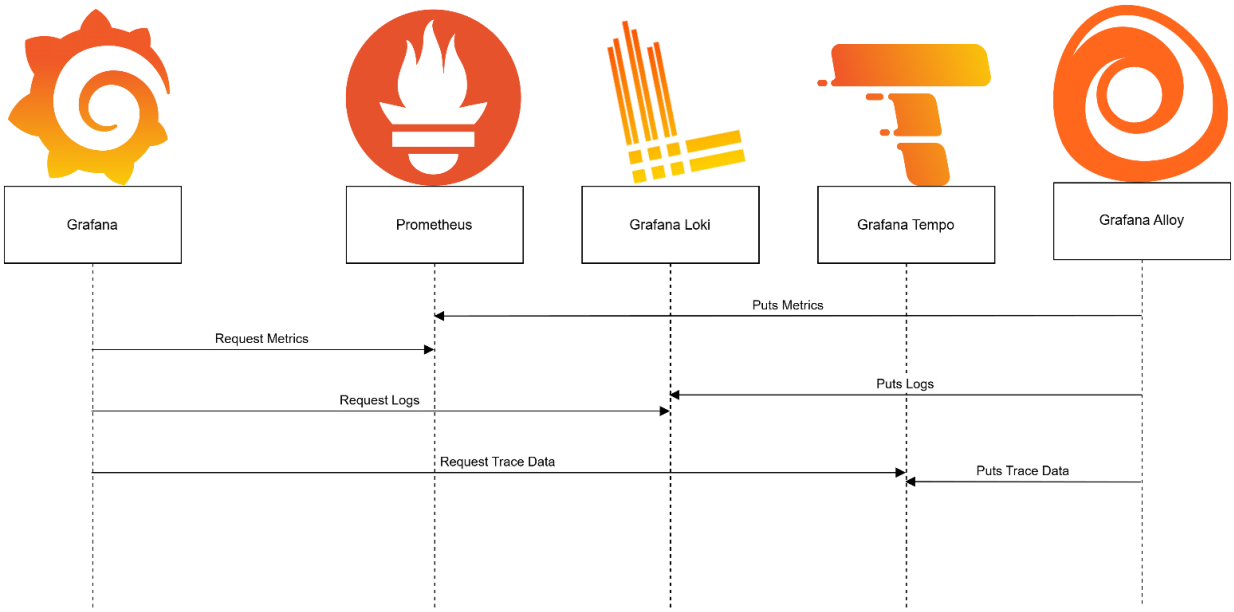

Рисунок 3.1 – Схема взаємодії сервісів моніторингу в запропонованій системі

У архітектурі даної роботи Alloy зменшує навантаження на конфігурацію для кожної програми, централізуючи задачу транспортування, включаючи пакетну обробку та повторні спроби отримання даних при невдачах. Таким чином Alloy надає єдину чітко визначену службу для інтеграції різнорідних джерел даних, тоді як сервери зберігання залишаються спеціалізованими для відповідних типів сигналів.

3.1.1 Демонстраційний додаток як інструментоване робоче навантаження

Дана робота також передбачає сервіс – вебдодаток на основі FastAPI (demo-app), призначений для роботи як реалістичне, інструментоване веб-навантаження до якого буде підключено моніторинг. Він буде упакований як образ контейнера, побудований на мові Python та обслуговується Uvicorn, із залежностями, що

включають бібліотеки FastAPI, prometheus-client та OpenTelemetry для інструментації FastAPI та експорту OTLP.

Програма генерує три класи телеметрії:

– метрики через кінцеву точку HTTP /metrics;

– логи як структурований JSON до stdout;

– трасування як діапазони OTLP.

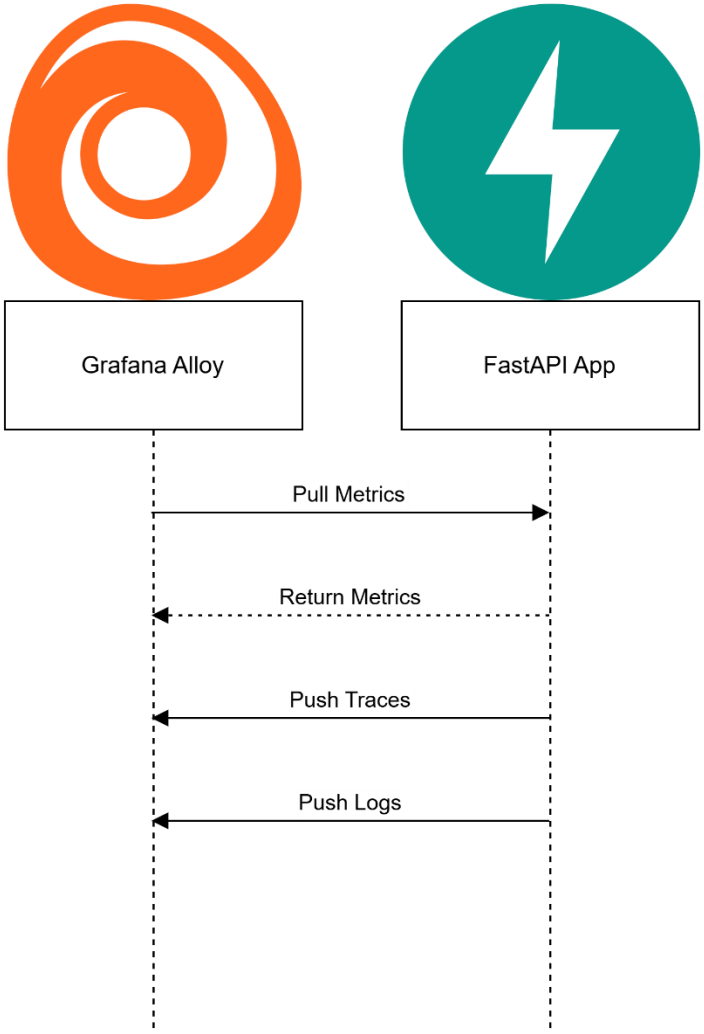

Рисунок 3.2 – Схема взаємодії Grafana Alloy та FastAPI додатку

На рисунку 3.2 можна побачити як Grafana Alloy взаємодіє з іншими сервісами, підтримуючи різноманітні протоколи та методи передачі даних. Метрики збираються за допомогою методу витягування – сервіс надає спосіб отримати данні метрик, і сервіс Alloy збирає їх через HTTP. Натомість, трасування експортуються програмою до Alloy через HTTP OTLP, після чого

Alloy пересилає їх до Tempo. Логи виводяться на stdout та дістаються з середовища виконання контейнера за допомогою Docker Log Intake в Alloy, а потім записуються до Loki.

Також в програмній реалізації передбачено навмисно налаштовувану затримку для емуляції роботи з базою даних, що створює вимірювані коливання в гістограмах тривалості запитів та часі діапазону трасування.

3.1.2 Зворотний проксі-сервер та HTTPS

В системі передбачено додатковий сервіс Traefik, який забезпечує вхідний рівень для маршрутизації HTTP запитів. Він інтегрується з метаданими сервісів Docker для динамічного налаштування маршрутів та надає доступ до сервісів моніторингу та демонстраційного додатку через HTTPS без необхідності налаштування проксі-сервера для кожного сервісу, приклад роботи такого проксі-серверу зображено на рисунку 3.3.

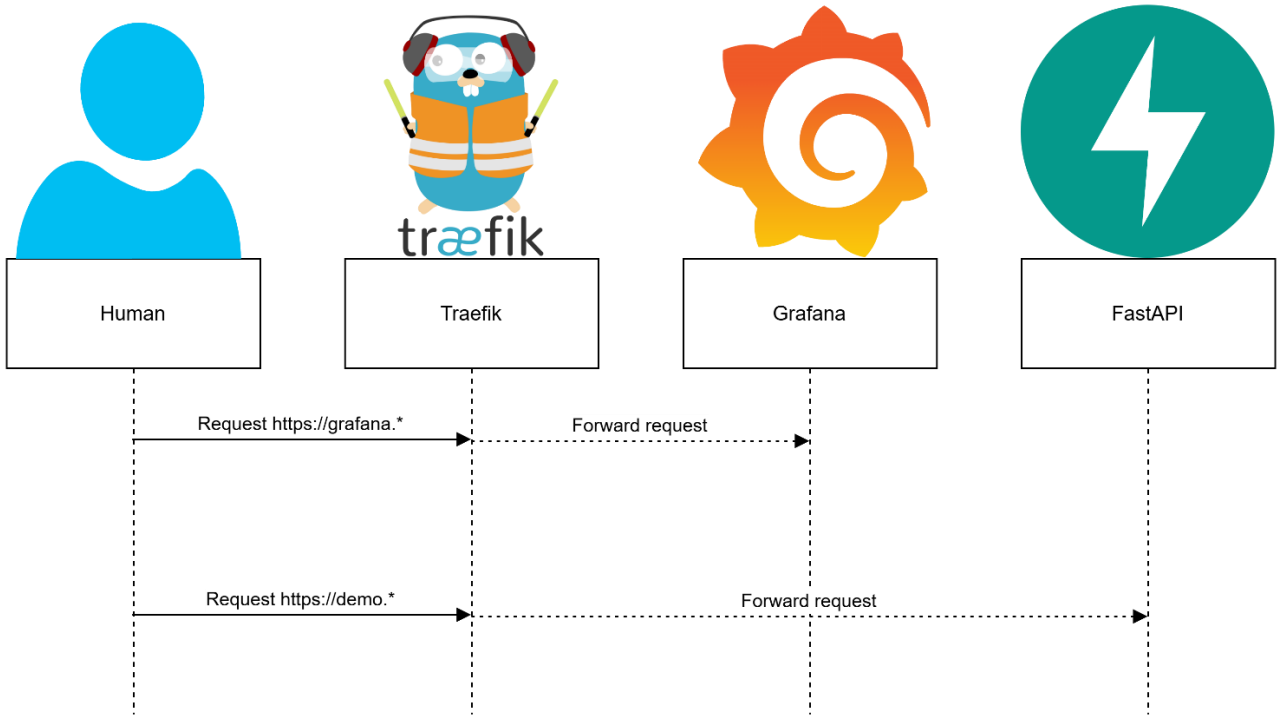

Рисунок 3.3 – Принцип взаємодії зворотного проскі-серверу

В результаті стек забезпечує узгоджену межу безпеки на периферії, залишаючи обмін телеметрією ізольованим від внутрішніх мереж.

3.2 Реалізація системи моніторингу веб додатків

Стек моніторингу збирається шляхом вираження кожного компонента як сервісу Docker Compose, підключення його до відповідної мережі, а потім явного підключення сервісів між собою для передачі та обробки артефактів. Практично це виглядає наступним чином – зворотний проксі-сервер працює як точка доступу для звичайних користувачів, мережа телеметрії як безпечна мережа передачі даних, а сервіси зберігання та обробки даних як внутрішні залежності, які не підключені безпосередньо до Інтернету, окрім випадків, коли це необхідно для адміністрування.

3.2.1 Налаштування Traefik для вхідного HTTPS-трафіку

Traefik визначається як контейнер, який взаємодіє з Docker, оголошує точки доступу HTTP/HTTPS запитів та налаштовує отримання сертифікатів. Уривок нижче ілюструє основні елементи – виявлення сервісів на основі Docker, резолвер ACME та захищену панель інструментів (див. лістинг 3.1).

Лістинг 3.1 – Налаштування сервісу Traefik в системі Docker Compose

```
services:
  traefik:
    image: traefik:v2.11
    command:
- --providers.docker=true
- --providers.docker.exposedbydefault=false
- --entrypoints.web.address=:80
- --entrypoints.websecure.address=:443
- --entrypoints.web.http.redirections.entrypoint.to=websecure
- --entrypoints.web.http.redirections.entrypoint.scheme=https
```

Продовження лістингу 3.1

```
- --certificatesresolvers.letsencrypt.acme.email=${ACME_EMAIL}
certificatesresolvers.letsencrypt.acme.storage=/letsencrypt/ac
me.json
- --certificatesresolvers.letsencrypt.acme.httpchallenge=true
certificatesresolvers.letsencrypt.acme.httpchallenge.entrypoin
t=web
volumes:
- /var/run/docker.sock:/var/run/docker.sock:ro
labels:
- traefik.enable=true
- traefik.http.routers.dashboard.rule=Host(`traefik.${DOMAIN}`)
- traefik.http.routers.dashboard.entrypoints=websecure
- traefik.http.routers.dashboard.tls.certresolver=letsencrypt
- traefik.http.routers.dashboard.middlewares=dashboard-auth
-                          traefik.http.middlewares.dashboard-
auth.basicauth.users=${DASHBOARD_AUTH}
```

Наведена вище конфігурація передбачає мінімальний набір параметрів середовища, які зазвичай надаються через файл налаштування env, приклад яких наведено у лістингу 3.2. Вони є обов’язковими для конфігурації, для забезпечення безпечного HTTPS з’єднання [25].

Лістинг 3.2 – Приклад env конфігурації для сервісу Traefik

```
DOMAIN=example.com
ACME_EMAIL=admin@example.com
DASHBOARD_AUTH=admin:$apr1$...
```

Після встановлення Traefik будь-який додатковий HTTP-сервіс можна буде відкрити, додавши невеликий набір міток. У лістингу 3.3 продемонстровано шаблон для Grafana.

Лістинг 3.3 – Приклад конфігурації зворотного проксі-сервера для HTTP доступу до сервісу Grafana

```
services:
  grafana:
    labels:
      - traefik.enable=true
      -
traefik.http.routers.grafana.rule=Host(`grafana.${DOMAIN}`)
      - traefik.http.routers.grafana.entrypoints=websecure
      -
traefik.http.routers.grafana.tls.certresolver=letsencrypt
      -
traefik.http.services.grafana.loadbalancer.server.port=3000
```

Дана конфігурація забезпечить доступ по HTTPS та домену описаному в опції rule=Host, до веб панелі Grafana яка за замовчуванням налаштована використовувати порт 3000.

3.2.2 Налаштування сервісів сховища та відображення артефактів моніторингу

Кожен сервіс-сховище розгортається як внутрішній сервіс з конфігурацією, змонтованою у вигляді файлу. Loki та Tempo навмисно не маршрутизуються через Traefik у цьому стеку, натомість сервіси Grafana та Alloy використовуватимуть окрему мережу telemetry-network як безпечний канал передачі даних.

Tempo налаштовано на прийом OTLP через HTTP та gRPC:

Лістинг 3.4 – Налаштування протоколів отримання даних у Tempo

```
distributor:
  receivers:
    otlp:
      protocols:
```

Продовження лістингу 3.4

```
        http:
          endpoint: 0.0.0.0:4318
        grpc:
          endpoint: 0.0.0.0:4317
```

Loki налаштовано як локальне сховище логів з одним екземпляром даних та короткотривалим збереженням даних.

Лістинг 3.5 – Налаштування Loki на просте збереження логів протягом 7 днів

```
schema_config:
  configs:
    - from: 2024-01-01
      store: tsdb
      object_store: filesystem
limits_config:
  retention_period: 168h
```

Prometheus налаштовано на парсинг даних метрик на веб додатку та прийом віддаленого запису, з сервісу як-от Alloy.

Лістинг 3.6 – Просте налаштування Prometheus на парсинг метрик з веб ресурсу

```
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: demo-app
    static_configs:
      - targets: ['demo-app:8000']
```

Початкова конфігурація Grafana вимагає підключення попередньо згаданих сервісів як джерел даних для відображення їх в подальшому на дашбордах.

Налаштовувати Grafana можна як за допомогою графічного інтерфейсу веб додатку так і використовуючи файли конфігурацій.

Лістинг 3.7 – Конфігурація джерел даних у Grafana

```
datasources:
  - name: Tempo
    type: tempo
    uid: tempo
    jsonData:
      tracesToLogsV2:
        datasourceUid: loki
      tracesToMetrics:
        datasourceUid: prometheus
  - name: Prometheus
    type: prometheus
    uid: prometheus
    jsonData:
      exemplarTraceIdDestinations:
        - name: trace_id
          datasourceUid: tempo
```

3.3 Інтеграція демонстраційного вебдодатку та E2E тестування

Сервіс demo-app реалізовано на Python з використанням FastAPI, оскільки цей фреймворк має компактний набір залежностей, швидко розгортається та добре підходить для створення простого вебінтерфейсу на базі HTML-сторінок.

FastAPI забезпечує HTTP-маршрутизацію та обробку запитів, prometheus-client надає кінцевий запит /metrics з даними метрик, а бібліотеки OpenTelemetry забезпечують автоматичну інтеграцію з FastAPI та експорт OTLP для трасування. Список всіх використаних залежностей можна побачити у лістингу 3.8.

Лістинг 3.8 – Список використаних бібліотек у демонстраційному вебдодатку

```
fastapi==0.115.5
uvicorn[standard]==0.32.1
prometheus-client==0.21.0
opentelemetry-sdk==1.28.2
opentelemetry-instrumentation-fastapi==0.49b2
opentelemetry-exporter-otlp-proto-http==1.28.2
```

Для локального виконання застосунок можна запустити як звичайний сервіс Python за допомогою команд з лістингу 3.9:

Лістинг 3.9 – Команди для запуску вебдодатку

```
cd demo-app
python -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
ubicorn app:app --host 0.0.0.0 --port 8000
```

При відвідуванні веб сторінки http://localhost:8000 повинен відобразитися інтерфейс зображений на рисунку 3.4

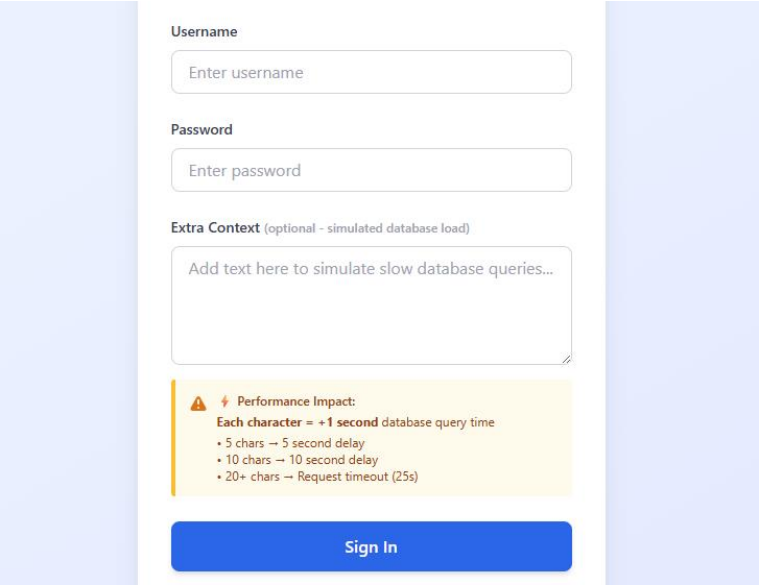

Рисунок 3.4 – Графічний інтерфейс вебдодатку

Після запуску повного стеку сервісів можна провести E2E перевірку, яка зосереджена на перевірці того, що кожен артефакт моніторингу досягає свого призначеного сервісу, і що кореляція між даними збережена.

У веб інтерфейсі Grafana перевірку артефактів можна виконати за допомогою розділу Explore, та виконуючи певні запити, а саме:

– Логи (Loki) ({container="demo-app"} | json).
– Метрики (Prometheus) (http_requests_total або login_attempts_total).
– Трасування (Tempo) (пошук service.name = "demo-app", а потім перехід до журналів/метрик через налаштовані посилання).

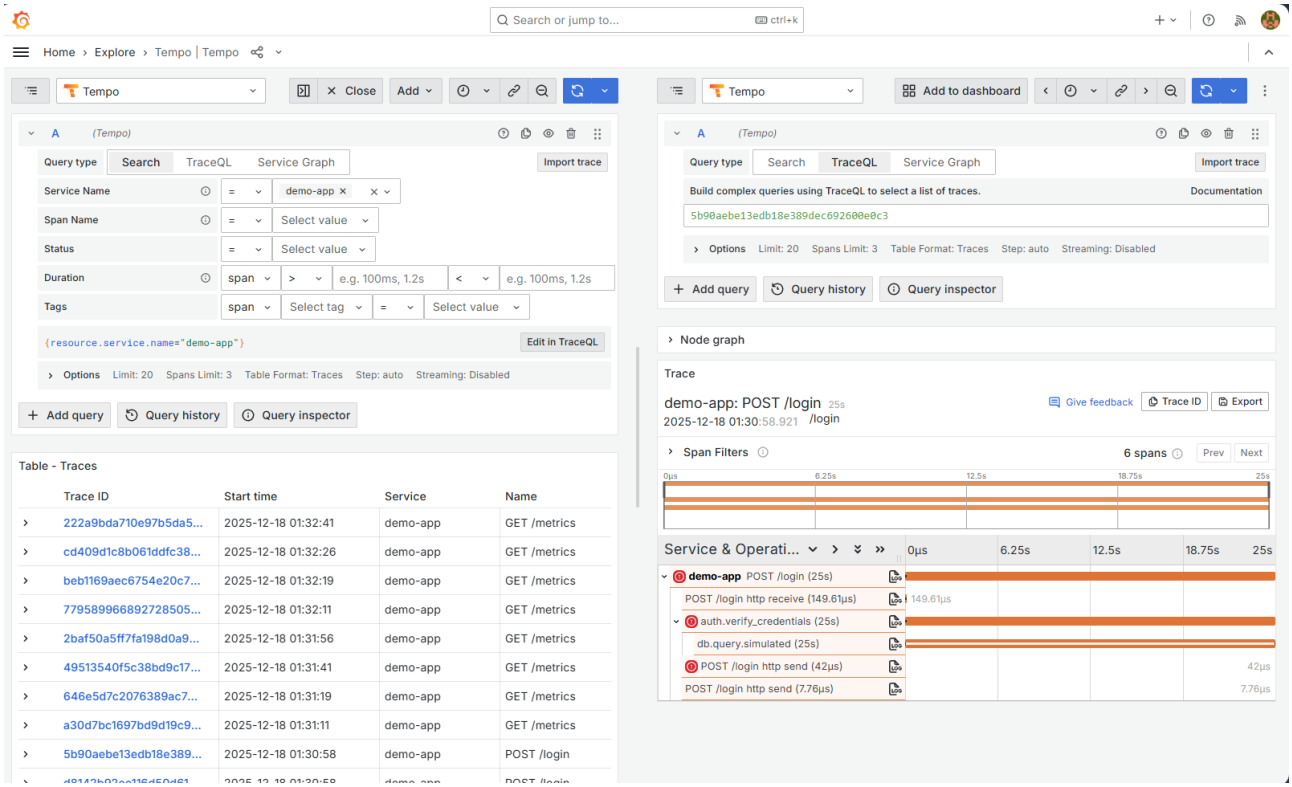

Рисунок 3.5 – Приклад артефактів трасування Tempo з запитом авторизації

На рисунку 3.5 можна побачити, що звернення з ідентифікатором, який починається на 5b90…, завершилося передчасно після штучної операції до бази даних, яка тривала понад 25 секунд. Натиснувши на графіки справа, можна отримати детальнішу інформацію про інцидент і переглянути відповідні логи, що зберігаються в іншому джерелі, а саме в Loki.

3.4 Дашборди для централізованого моніторингу та сповіщення

Дашборди Grafana дають змогу об’єднати на одній сторінці дані з кількох джерел, поєднуючи числові метрики з Prometheus, журнали подій з Loki та детальні трасування з Tempo. Метрики першими сигналізують про відхилення або деградацію, логи додають контекст і дозволяють швидко зрозуміти, що саме відбулося та в якому компоненті, а трасування показує повний ланцюжок обробки і допомагає визначити, яка операція або обробник спричинили затримку чи помилку. У сукупності це прискорює пошук першопричини та підвищує ефективність розслідування інцидентів. Приклад реалізованого дашборду наведено на рисунку 3.6.

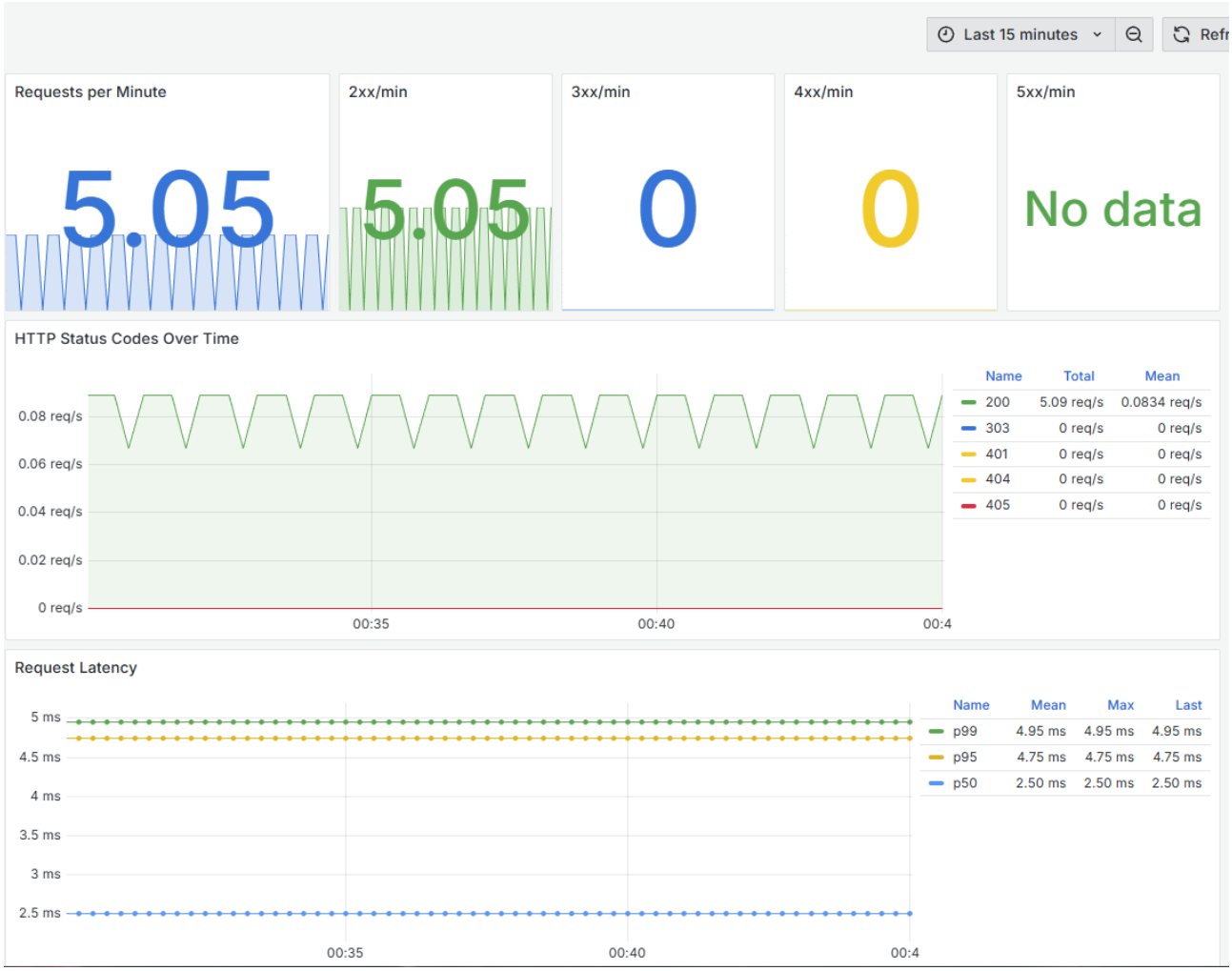

Рисунок 3.6 – Простий дашборд з інформацією отриманою з демонстраційного вебдодатку

Для багатопанельних дашбордів поширеним методом є визначення невеликого набору змінних інформаційної панелі (наприклад, сервіс, контейнер, чи шлях запиту) та використання їх подалі в запитах PromQL та LogQL. Це зменшує розбіжності між панелями та робить перехресне дослідження сигналів систематичним: панель метрик може бути обмежена однією службою, тоді як суміжна панель журналу застосовує те саме обмеження через мітки та поля JSON, а панель трасування фільтрує за назвою сервісу.

Також поширеною є розробка та підтримка кількох дашбордів для одного й того ж самого сервісу, кожен з яких оптимізований для різної перспективи розслідування та часового простору. Команди роблять це, тому що цілі моніторигу, під час звичайного робочого процесу, не збігаються з цілями та завданнями, що постають під час розслідування інциденту.

Дашборди можна створювати як у графічному інтерфейсі так і використовуючи файли JSON. Дашборд який раніше було зображено можна імпортувати завдяки конфігурації Grafana описаної у лістингу 3.10.

Лістинг 3.10 – Налаштування імпорту дашборду з JSON у Grafana

```
providers:
  - type: file
    updateIntervalSeconds: 10
    options:
      path: /etc/grafana/provisioning/dashboards
```

Таким чином, практика підтримки кількох дашбордів для одного сервісу дозволяє розділити повсякденний контроль стану системи та роботу під час інцидентів, коли потрібна інша деталізація й інший часовий масштаб. Це спрощує супровід системи, пришвидшує реагування та підвищує узгодженість процесів розслідування.

Алерти або сповіщення доповнюють дашборди і вирішують іншу задачу: якщо дашборди потрібні для аналізу та пояснення, що саме сталося, то алерти потрібні для швидкого виявлення проблеми в момент її виникнення. На практиці

команди зазвичай будують сповіщення на основі невеликого набору стабільних джерел даних з невеликою кількістю варіантів значень, щоб уникнути шуму та дублювання подій. Саме тому для сповіщень доцільно використовувати узагальнені метрики на кшталт росту кількості помилок сервісу, деградації затримок або аномалій автентифікації, тоді як деталізація причин та контексту вже виконується через дашборди.

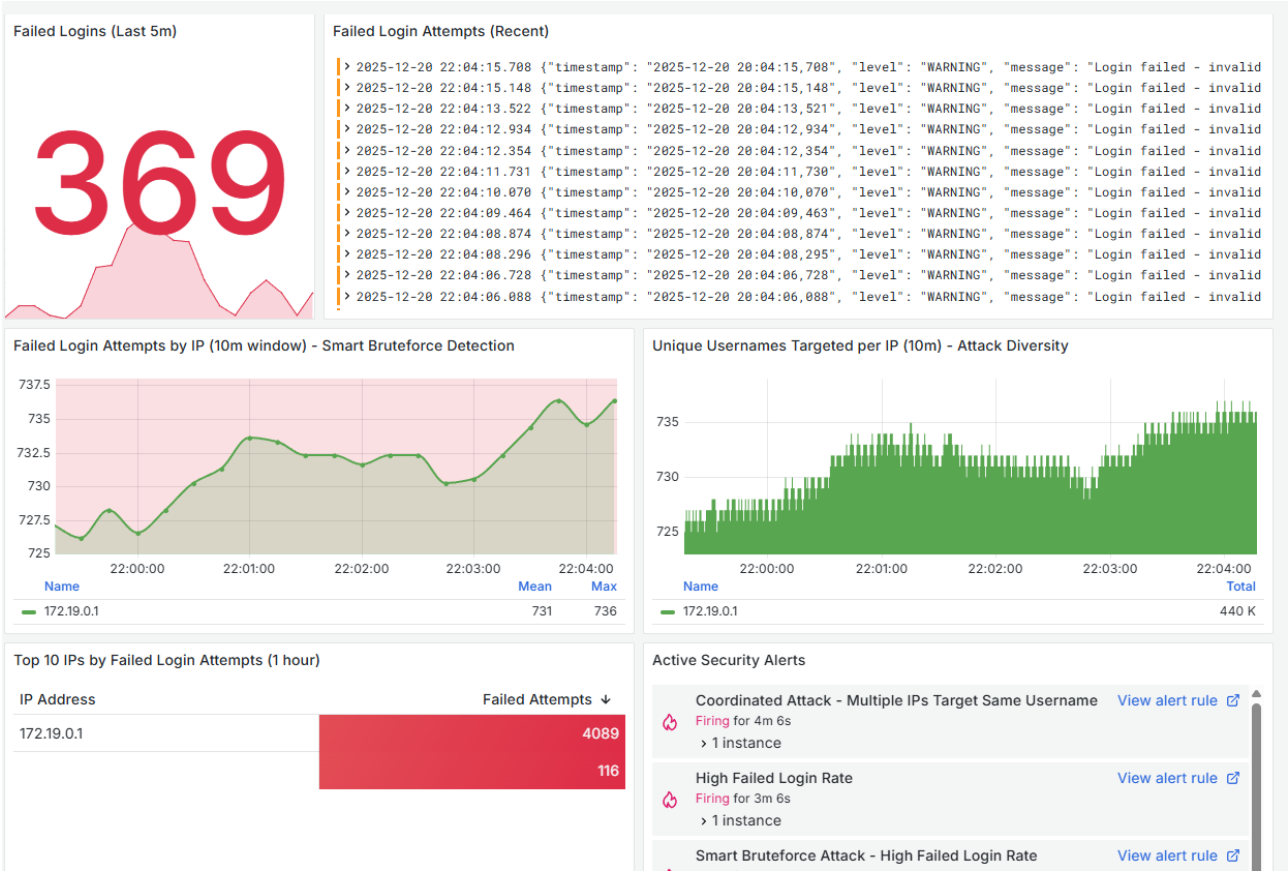

Рисунок 3.7 – відображення алертів на дашборді при спробі брутфорсу

На Рисунку 3.7 зображено елементи дашборду які показують в реальному часі велику кількість невдалих запитів авторизації. У реалізованому підході правила алертингу оцінюються на основі запитів до Prometheus, оскільки метрики є найкращим джерелом для надійних і повторюваних умов спрацювання. Крім цього реалізовано й алерт на базі журналів з Loki.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Охорона праці під час професійної діяльності, пов’язаної з розробкою та експлуатацією системи моніторингу безпеки вебдодатків, спрямована на збереження здоров’я працівника та запобігання професійним ризикам, що виникають при тривалій роботі з персональним комп’ютером і офісним електрообладнанням. Характерні умови такої роботи включають інтенсивну взаємодію з інформаційними панелями дашбордами, журналами подій, аналітичними інтерфейсами, роботу з інцидентами та сповіщеннями в режимі обмеженого часу. Це формує поєднання фізичних і і психофізіологічних чинників (висока відповідальність, дефіцит часу, багатозадачність), що потребує системних заходів з організації безпечних умов праці. Загальні правові засади забезпечення безпечних і здорових умов праці в Україні визначені Законом України «Про охорону праці», який закріплює обов’язки роботодавця щодо створення належних умов, профілактики травматизму та організації навчання з питань охорони праці [17].

Важливою організаційною основою є навчання та інструктажі з охорони праці, що формують у працівника практичні навички безпечної роботи та дій у нестандартних ситуаціях. Порядок проведення навчання і перевірки знань регламентується положенням (НПАОП 0.00-4.12-05) [18], яке визначає види інструктажів, вимоги до перевірки знань і документування результатів. Для робіт за ПК це охоплює правила безпечної експлуатації електроприладів і подовжувачів, вимоги до організації робочого місця, дотримання режиму праці та відпочинку, а також порядок повідомлення про несправності обладнання або ознаки небезпечних ситуацій (перегрів, запах горілої ізоляції, іскріння).

Санітарно-гігієнічні умови праці в офісному приміщенні включають нормування мікроклімату та забезпечення комфортних параметрів температури, вологості і швидкості руху повітря. Порушення мікроклімату підвищує втому, знижує працездатність і може посилювати прояви головного болю та

дискомфорту під час інтенсивної роботи. В Україні поняття мікроклімату виробничих приміщень та підходи до його нормування визначені санітарними нормами мікроклімату, де мікроклімат розглядається як сукупність факторів внутрішнього середовища, що впливають на тепловий обмін організму з оточенням [19]. Для робочих місць за ПК підтримання нормативних параметрів мікроклімату є одним із базових профілактичних заходів з охорони праці.

Електробезпека при роботі з комп'ютерною технікою охоплює безпечне використання електромережі, подовжувачів, джерел безперебійного живлення, мережевого обладнання та зарядних пристроїв. Профілактика електротравматизму в офісних умовах базується на використанні справного та сертифікованого обладнання, недопущенні перевантаження електромережі, своєчасному виявленні пошкоджених кабелів та розеток, а також на забороні експлуатації обладнання з ознаками несправності. Нормативну основу для безпечної експлуатації електрообладнання споживачів визначають «Правила безпечної експлуатації електроустановок споживачів», які встановлюють вимоги безпеки під час експлуатації електроустановок та електрообладнання [20].

Пожежна безпека є обов'язковою складовою охорони праці для робочих місць, де одночасно працює багато електроприладів. У практичному вимірі це означає підтримання евакуаційних шляхів у вільному стані, дотримання режимних вимог у приміщенні, контроль справності електропроводки й обладнання, а також наявність організаційних інструкцій щодо дій персоналу у разі пожежі або задимлення. Загальні вимоги пожежної безпеки в Україні встановлені «Правилами пожежної безпеки в Україні», які визначають базові вимоги до об'єктів, приміщень та експлуатації обладнання, і застосовуються також до офісних приміщень з комп'ютерною технікою [21].

Розроблене рішення для моніторингу безпеки вебдодатків придатне до впровадження в офісному середовищі та відповідає вимогам охорони праці, техніки безпеки, електро- і протипожежної безпеки за умови їх дотримання під час експлуатації.

4.2 Ергономічні вимоги до робочого місця користувача персональним комп’ютером (ПК)

Ергономіка робочого місця для роботи за персональним комп’ютером напряму впливає на безпеку та ефективність професійної діяльності. Для спеціалістів, які займаються моніторингом безпеки вебдодатків, характерні тривалі періоди зосередженої роботи, аналіз логів, метрик, інцидентів та підготовка звітів. За таких умов незручна поза, неправильне розташування монітора або ввідних пристроїв, а також недостатня організація перерв підвищують ризики зорової втоми, болю в шиї та спині, перенавантаження кистей і передпліч, що може знижувати точність рішень та збільшувати ймовірність помилок.

На практиці вимоги до організації роботи з дисплейним обладнанням формуються поєднанням трудового законодавства, стандартів ергономіки та рекомендацій з охорони праці.

Робоча поза має підтримувати нейтральне положення тулуба та кінцівок, щоб зменшити статичне навантаження і напруження м’язів [22]. Бажані характеристики наступні:

– Крісло має бути з можливістю регулюванням висоти, підтримкою попереку, можливістю налаштувати глибину сидіння, з підлокітниками для часткової підтримки рук.

– Положення ніг - стопи мають бути повністю на підлозі або на підставці, а коліна орієнтовно під кутом близько 90 градусів.

– Висота стола має бути така, щоб плечі були розслаблені, лікті біля тулуба, а передпліччя приблизно паралельні підлозі під час роботи з клавіатурою та мишею.

– Відповідні вимоги також застосовуються і до монітора, за яким відбувається робота. Ключовими вимогами до монітора є наступні:

– Верхня межа екрану зазвичай налаштовується на рівні очей або трохи нижче, щоб не підіймати обличчя доверху.

– Відстань до екрана підбирається індивідуально (часто орієнтиром виступає довжина руки) щоб не нахилятися вперед і не напружувати зір.

– Екран бажано розташовувати так, щоб уникати прямих відблисків від вікна або світильників.

Також є певний ряд вимог до периферійних пристроїв. Клавіатуру треба розмістити так, щоб кисті були в нейтральному положенні, без надмірного згинання вгору або вниз, а лікті залишались поруч з тулубом.

Комп’ютерна мишка має бути на тому ж рівні, що й клавіатура, і максимально близько до користувача, щоб не тягнутися рукою.

Якщо робота відбувається з ноутбука тривалий час, можуть виникати типові ергономічні проблеми. До прикладу, екран може бути надто низько. Можливим рішенням є підставка під ноутбук та окрема клавіатура і комп’ютерна мишка.

Не менш важливою є ергономіка робочого столу. Стіл має забезпечувати достатню глибину для розміщення монітора на комфортній відстані, наявність вільного простору для ніг, а також можливість гнучкого розташування ввідних пристроїв без постійного витягування рук вперед. У професійних стандартах ергономіки для роботи з візуальними дисплейними терміналами підкреслюється, що компонування робочої станції має підтримувати прийнятні робочі пози і зменшувати статичне навантаження, а під час проєктування або підбору столу доцільно орієнтуватися на принципи компонування робочого місця та постуральні вимоги, описані в ISO 9241-5 [23].

Останнім часом стала популярною практика ходьби під час роботи. Вона реалізується через так званий “active workstation”, де користувач виконує задачі за ПК під час повільної ходьби низької інтенсивності на біговій доріжці. Дані систематичних оглядів і метааналізу свідчать, що такі робочі станції підвищують енерговитрати і зменшують час сидіння, тобто ефективні саме як інструмент боротьби з тривалою малорухомістю.

Водночас питання ефективності для продуктивності зазвичай зводиться до компромісу, адже під час ходьби швидкість і точність окремих дрібномоторних задач можуть дещо погіршуватися (наприклад у наборі тексту), але базові

когнітивні показники часто не демонструють суттєвого погіршення при повільній ходьбі, а деякі дослідження навіть відмічають відсутність негативного впливу на робочу результативність у типових офісних сценаріях.

Для моніторингу безпеки вебдодатків роботу стоячи або з легкою ходьбою можна використовувати як додатковий спосіб зменшити сидячий час і підтримати самопочуття. Водночас для задач, де важливі набір тексту та висока точність, особливо під час пікових інцидентів, краще працювати сидячи.

Важливою складовою ергономіки є режим праці та відновлення. Директива 90/270/ЕЕС [22] вимагає організовувати щоденну роботу за дисплеєм так, щоб вона була розбавлена перервами або зміною діяльності, зменшуючи навантаження від безперервного перегляду екрана.

У практичних матеріалах з безпеки праці й ергономіки також підкреслюється цінність коротких пауз і періодичних перерв, які можна поєднувати з простими діями на місці (зміна пози, легка розминка, переключення фокуса зору).

Це має прямий зв’язок з темою моніторингу безпеки вебдодатків, адже під час тривалих чергувань або розслідування інцидентів зростає когнітивне навантаження, а регулярні короткі паузи допомагають підтримувати стабільну працездатність і уважність без істотної втрати робочого часу.

Результати досліджень показують, що ергономічний ефект рідко досягається одним втручанням, наприклад тільки інструкцією “тримати правильну позу”. Водночас метааналізи та огляди щодо коротких перерв свідчать, що мікропаузи загалом можуть покращувати самопочуття, зменшувати втому і в окремих умовах підтримувати або навіть покращувати продуктивність, що робить їх практично корисним організаційним заходом у роботі за ПК.

Для зорової втоми важливо зазначити, що популярні правила на кшталт “20-20-20” широко рекомендуються, але наукова база щодо їх універсальної ефективності є обмеженою. Водночас дослідження підтверджують, що регулярні перерви як такі можуть мати позитивний ефект на окремі параметри зорового комфорту.

ВИСНОВКИ

У кваліфікаційній роботі розглянуто та обґрунтовано підхід до побудови системи моніторингу вебдодатків з акцентом на кібербезпеку. Показано, що класичні методи спостереженння, а саме контроль доступності й продуктивності, є недостатніми без безпекового контексту, а ефективне виявлення інцидентів потребує системного збору та аналізу телеметрії з різних рівнів – застосунку, інфраструктури, даних тощо.

У межах теоретичної частини визначено контекст застосування та сформовано узгоджені вимоги до моніторингу вебдодатків з точки зору безпеки. Описано ключові артефакти спостереження – метрики та логи, їх роль у виявленні атак, діагностиці аномалій, розслідуванні інцидентів та підтримці реагування. Систематизовано ключові аспекти безпеки на різних рівнях моніторингу та досліджено інструменти, архітектури й моделі розгортання платформ моніторингу, включно з SaaS і гібридними підходами.

У практичній частині роботи розроблено прототип системи моніторингу і демонстраційного вебзастосунку, інтегрованого зі збором телеметрії. Реалізовано базову структуру збору, агрегації та візуалізації даних, а також підхід до створення шаблонів виявлення на основі узгодження метрик, логів і трасування. Проведено E2E тестування, яке підтвердило працездатність інтеграції компонентів і можливість відтворюваної перевірки сценаріїв, що імітують підозрілі події та потенційні інциденти безпеки.

Отримані результати підтверджують доцільність використання спостережуваності як джерела подій безпеки для вебдодатків, а також демонструють, що поєднання різних типів телеметрії підвищує якість виявлення інцидентів та зменшує час на первинну діагностику. Запропонований підхід і реалізований прототип можуть бути використані як основа для подальшого розширення набору правил і кореляцій, впровадження оцінювання ризику подій, підключення додаткових джерел (WAF, IAM, хмарні журнали, CI/CD), а також автоматизації відповідей на інциденти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Microsoft. (2023, November 15). Architecture strategies for designing and creating a monitoring system. Microsoft Learn. https://learn.microsoft.com/en-us/azure/well-architected/operational-excellence/observability. (дата звернення: 29.10.2025)

[2] OWASP Top 10 Team. (2021). A09:2021 - Security logging and monitoring failures. OWASP. https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/. (дата звернення: 29.10.2025)

[3] PCI Security Standards Council. (2024, June). Payment Card Industry Data Security Standard (PCI DSS) v4.0.1 [PDF]. https://docs-prv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI-DSS-v4_0_1.pdf (дата звернення: 29.10.2025).

[4] Trunc. (n.d.). Log management and ISO 27001 Annex A.12.4. https://trunc.org/iso-27001 (дата звернення: 01.11.2025).

[5] Wadhwa, P. (2025, September 30). ISO 27001 logging and monitoring policy: Requirements, objectives, and best practices. Sprinto. https://sprinto.com/blog/iso-27001-logging-and-monitoring-policy/ (дата звернення: 30.10.2025).

[6] Tan, M. (2024, July 2). Identify anomalies, outlier detection, forecasting: How Grafana Cloud uses AI/ML to make observability easier. Grafana Labs. https://grafana.com/blog/2024/07/02/identify-anomalies-outlier-detection-forecasting-how-grafana-cloud-uses-ai-ml-to-make-observability-easier/ (дата звернення: 31.10.2025).

[7] RSI Security. (2022, October 13). Breaking down the PCI logging requirements. https://blog.rsisecurity.com/breaking-down-the-pci-logging-requirements/ (дата звернення: 31.10.2025).

[8] Bekiares, T. (2023, November 19). Why do customers choose Elastic for logs? Elastic. https://www.elastic.co/blog/why-do-customers-choose-elastic-for-logs (дата звернення: 31.10.2025).

[9] Schreyer, T. (2020, November 17). Security observability: Why tracing? Traceable. https://www.traceable.ai/blog-post/security-observability-why-tracing (дата звернення: 31.10.2025).

[10] Datadog. (n.d.). How App and API Protection works in Datadog. Datadog Documentation. https://docs.datadoghq.com/security/application_security/how-it-works/ (дата звернення: 31.10.2025).

[11] Boland, G., & Rochau, M. (2025, June 16). Adaptive alerting: Faster, better insights with the new metrics forecasting UI in Grafana Cloud. Grafana Labs. https://grafana.com/blog/2025/06/16/adaptive-alerting-faster-better-insights-with-the-new-metrics-forecasting-ui-in-grafana-cloud/ (дата звернення: 03.11.2025).

[12] Business Wire. (2025, October 8). Grafana Labs revolutionizes AI-powered observability with GA of Grafana Assistant and introduces Assistant Investigations. Yahoo Finance. https://finance.yahoo.com/news/grafana-labs-revolutionizes-ai-powered-090000693.html (дата звернення: 03.11.2025).

[13] Dhillon, J. (2023, March 28). Reduce compliance TCO by using Grafana Loki for non-SIEM logs. Grafana Labs. https://grafana.com/blog/2023/03/28/reduce-compliance-tco-by-using-grafana-loki-for-non-siem-logs/ (дата звернення: 03.11.2025).

[14] Moore, N. (2022, December 15). A guide to cyber threat hunting with Promtail, Grafana Loki, Sigma, and Grafana Cloud. Grafana Labs. https://grafana.com/blog/2022/12/15/a-guide-to-cyber-threat-hunting-with-promtail-grafana-loki-sigma-and-grafana-cloud/ (дата звернення: 04.11.2025).

[15] Modée, A. M., Adler, L., & Ngoma, M. (2024, July 24). Log it like you mean it: Best practices for security. Elastic. https://www.elastic.co/blog/best-practices-security-log (дата звернення: 15.11.2025).

[16] MetricFire Blogger. (2021, March 9). Sample approaches of hybrid cloud monitoring models. MetricFire. https://www.metricfire.com/blog/examples-of-hybrid-cloud-deployment-models/ (дата звернення: 29.11.2025).

[17] Верховна Рада України. (1992, 14 жовтня). Про охорону праці (Закон України № 2694-XII, редакція від 12.09.2025). Законодавство України. https://zakon.rada.gov.ua/laws/main/2694-12 (дата звернення: 10.12.2025)

[18] Держнаглядохоронпраці України. (2005, 26 січня). Про затвердження Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці та Переліку робіт з підвищеною небезпекою (Наказ № 15, редакція від 25.10.2024). Законодавство України. https://zakon.rada.gov.ua/laws/main/z0231-05 (дата звернення: 10.12.2025)

[19] Міністерство охорони здоров'я України. (1999, 1 грудня). Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. Законодавство України. https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення: 10.12.2025)

[20] Міністерство праці та соціальної політики України. (1998, 9 січня). Про затвердження Правил безпечної експлуатації електроустановок споживачів (Наказ № 4). Законодавство України. https://zakon.rada.gov.ua/laws/main/z0093-98 (дата звернення: 10.12.2025)

[21] Міністерство внутрішніх справ України. (2014, 30 грудня). Про затвердження Правил пожежної безпеки в Україні (Наказ № 1417, редакція від 14.08.2024). Законодавство України. https://zakon.rada.gov.ua/laws/show/z0252-15 (дата звернення: 10.12.2025)

[22] Council of the European Communities. (1990, May 29). Council Directive 90/270/EEC on the minimum safety and health requirements for work with display screen equipment. EUR-Lex. https://eur-lex.europa.eu/eli/dir/1990/270/oj/eng (дата звернення: 10.12.2025)

[23] International Organization for Standardization. (1998). ISO 9241-5:1998 Ergonomic requirements for office work with visual display terminals (VDTs) - Part 5: Workstation layout and postural requirements (1st ed.). https://cdn.standards.iteh.ai/samples/16877/c63fa5bf2d9c4b05a08e9432912c8751/ISO-9241-5-1998.pdf (дата звернення: 10.12.2025)

[24] National Institutes of Health. (Дата не виявлена). Computer Workstation Ergonomics: Self-Assessment Checklist [PDF]. https://ors.od.nih.gov/sr/dohs/Documents/checklist-ergonomics-computer-workstation-self-assessment.pdf (дата звернення: 10.12.2025)

[25] Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. (2014). Комп'ютерні мережі. Книга 2 (навчальний посібник). Львів: Магнолія 2006. pp. 174. (дата звернення: 10.12.2025)

[26] А.Г. Микитишин, М.М. Митник, П.Д. Стухляк, В.В. Пасічник. (2013). Комп'ютерні мережі. Книга 1 (навчальний посібник). Львів: Магнолія 2006. pp. 130. (дата звернення: 10.12.2025)

[27] Tymoshchuk, D., Yasniy, O., Mytnyk, M., Zagorodna, N., & Tymoshchuk, V. (2024). Detection and classification of DDoS flooding attacks by machine learning method. CEUR Workshop Proceedings, 3842, pp. 184. (дата звернення: 10.12.2025)

[28] Revniuk, O. A., Zagorodna, N. V., Kozak, R. O., Karpinski, M. P., & Flud, L. O. (2024). The improvement of web-application SDL process to prevent insecure design vulnerabilities. Applied Aspects of Information Technology, 7(2), pp. 166. https://doi.org/10.15276/aait.07.2024.12 (дата звернення: 10.12.2025)

[29] Revniuk, O. A., & Zahorodna, N. V. (2024). Методологія кількісної оцінки захищеності вебдодатку електронної комерції на етапі експлуатації. Scientific Bulletin of Ivano-Frankivsk National Technical University of Oil and Gas, (2(57)), pp. 108. (дата звернення: 10.12.2025)

[30] Tryhubets, B., Tryhubets, M., & Zahorodna, N. (2024). Аналіз ефективності використання безкоштовних та комерційних сканерів вразливостей для веб-застосунків електронної комерції. Вісник Тернопільського національного технічного університету, 116(4), pp. 28. (дата звернення: 10.12.2025)

ДОДАТКИ

Додаток А

ITTAP 2021

The 1st International Workshop on Information Technologies: Theoretical and Applied Problems 2021

The Development of automated system for monitoring the information leakage into the Darknet

Oleksandr Pokydko[a], Olena Karelina[a], Liliana Dzhydzhora[a]

[a] *Ternopil Ivan Pulyuj National Technical University, Ruska, 56, Ternopil, 46001, Ukraine*

Abstract

The system for monitoring the corporate information leakage of into the darknet is developed. The web documents collected from darknet are compared with the documents of the company-customer of monitoring and conclusion about the information leakage is made on the basis of documents similarity. The system consists of the following modules: source search module in darknet, text mining module, cryptographic module, document similarity assessment module. The following mathematical methods are used to assess the degree of similarity between corporate documents and Darknet documents: vector space model and information interaction model. Software implementation of the tool for automatic monitoring of information leakage by Python is described. The system testing results are given.

Keywords 1

Dataleakage, monitoring, Python, Tor, Darknet.

1. Introduction

Digital assets play an important or even crucial role in today's business. In all medium and large companies, payment information about transactions, agreements, customers personal data, financial documentation is formed and stored in the information system. Leakage of such information carries severe reputational and monetary losses for the company. Taking into account the peculiarities of modern business, hackers illegally access the companies files and demand a huge ransom for their unlocking and non-disclosure. It is worth mentioning the recent attacks on Colonial Pipeline, Acer, Apple, when ransoms amounted to millions of dollars.

Corporate information leakage can also be caused by insiders - employees of the company. Sometimes insiders are financially interested, sometimes the leakage is due to incompetence, inattention or mistakes in the business process. The reasons are different, but the result is the same - losses for the company. Therefore, the task of developing an automated tool for network monitoring, which would detect corporate information leakage is of great importance. If the loss of information is detected as early as possible, there are more opportunities to minimize its negative consequences. As hackers often publish information in the Darknet, the monitoring tool is developed for this purpose.

2. Background

Different aspects of information leakage monitoring are studied in scientific discourse. The solutions for cloud computing are developed in papers [1, 2]. In paper [3] the functionality of the leak detection system is based on the capabilities of the virtual machines hypervisor. In papers [4, 5] the development of information leakage monitoring systems from Android devices is revealed. Monitoring information leakage in the Darknet is an important and unresolved problem.

CEUR-WS.org/Vol-3039/paper29.pdf

ITTAP'2021: 1nd InternationalWorkshop on Information Technologies: Theoretical and Applied Problems, November 16–18, 2021, Ternopil, Ukraine

EMAIL: oleksandr_pokydko2507@tntu.edu.ua (A. 1); karelina@tntu.edu.ua (A. 2), lilyadzhydzhora1970@gmail.com (A 3)

ORCID: 0000-0001-7105-7942 (A. 1); 0000-0002-5628-9048 (A. 2), 0000-0002-3672-4807 (A 3)

© 2021 Copyright for this paper by its authors.

Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

3. Characteristics of the data leakage detection system

The specification of the system is Darknet monitoring and collection of information about web documents in accordance with customer requirements. The collected web data sources are compared with confidential user documents. If the document that is semantically similar to users' confidential documents appears on the Internet, the system indicates possible data leakage.

Typically, the similarity of documents is determined by solid similarity metric based on repetition. This approach neglects all potential semantic correlations between different words. The system does not compare pure content, but the value of web documents and user documents. The system consists of modules presented in Figure 1.

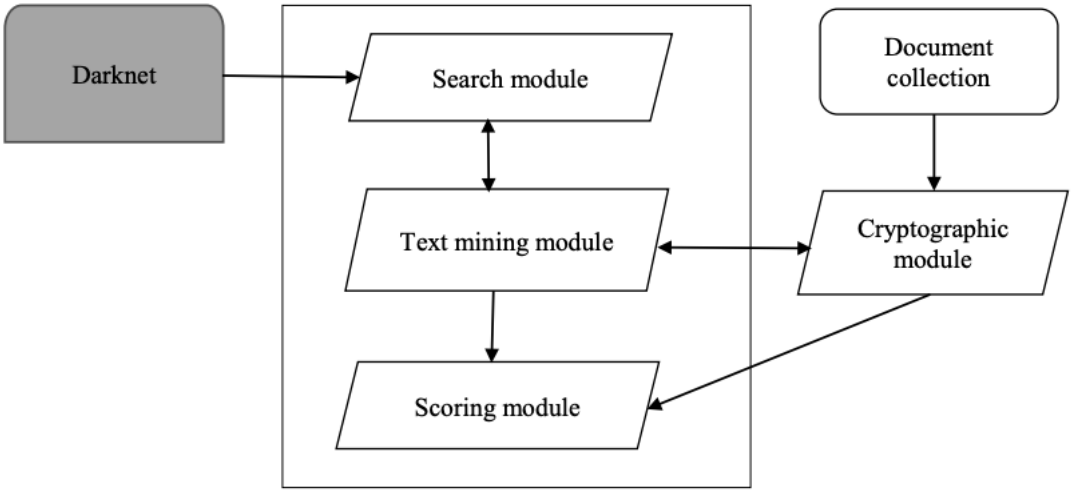

Figure 1: Modules of the system

The documents collection contains protected or confidential information. Encryption is required to protect these documents. The cryptographic module is responsible for preparing the encrypted version of the documents. To ensure the document content confidentiality, the cryptographic module is located on the client server. All other services are located in the cloud. The search module is responsible for detecting web pages that indicate data leakage. The search module includes the scanner module that examines the structure of websites, identifies those pages of websites that contain relevant data, and indexes those pages using keywords. The text extraction module converts web documents into the corresponding mathematical image. The evaluation module corresponds to the mathematical representation of web documents and confidential user documents.

4. Mathematical methods used to detect data leakage

Mathematical methods are used in the modules of text mining, cryptography and evaluation. Taking into account the search query, received web documents and confidential documents of the users, the calculation module calculates the conformity assessment, which measures the similarity of these documents. The scoring module uses various mathematical documents representations.

Depending on the type of information that prevails in the document, different mathematical methods are used.

4.1. Vector space model

Vector space model (VSM) is widely used as mathematical model of the systems. In VSM, documents are represented by the vector in n-dimensional vector space, where n is the number of terms of the keyword or index [6].

In data leakage detection system the VSM is proposed to be used as a basis for cryptographic and text analysis module. As a result, the scoring module can match the mathematical representation of web documents and user confidential documents based on vector space.

Presentation of VSM documents is as follows.

On the one hand, a set of keywords $C = \{c_1, \dots, c_i\}$ is obtained from confidential documents during indexing. On the other hand, another set of keywords $W = \{w_1, \dots, w_j\}$ is derived from web documents during indexing. Web documents and confidential user documents are represented by VSM over $C \cup W$ in the following way: under the condition of finite set $T = C \cup W$ of index terms $T = \{t_1, \dots, t_n\}$, any web document D_j is assigned vector v_j of real numbers, as shown below:

$$v_j = (w_{ij})_{i=1,\dots,n} = (w_{1j}, \dots, w_{ij}, \dots, w_{nj}) \quad (1)$$

Confidential user documents U_k should also be presented as vector v_k of real numbers, as shown below:

$$v_k = (w_{ik})_{i=1,\dots,n} = (w_{1k}, \dots, w_{ik}, \dots, w_{nk}) \quad (2)$$

Weight w_{ij} is interpreted as the value to which the index term t_i characterizes the document. Web document vectors and customer documents vectors are compared to some degree of similarity. The disadvantage of this presentation is that due to the potential diversity of web documents, the dimension of the vector space can be quite high requiring significant computing power to determine the degree of similarity. Web document D_j is presented to the customer who has confidential document U_k, if they are quite similar, i.e. the degree of similarity S_{jk} between the web document vector v_j and the confidential user vector v_k exceeds a certain threshold, i.e.

$$S_{jk} = s(v_j, v_k) > K \quad (3)$$

Term weight indices express how important the term or keyword is to describe the content of the document. The simplest weighing scheme is binary, which indicates the presence or absence of the keyword in the document. This scheme is usually insufficient, as it does not distinguish documents with frequent and infrequent keyword inclusions. The frequency with which keywords is met in the document is called term-frequency weight. The logarithm-based weighing scheme is used to adjust the frequency within the document. This is done because the terms that are frequently met in the document are not necessarily more important than special terms that are met in the document only once.

Documents sizes can vary greatly. In order to compensate this fact, the length normalization is used, i.e. the document rank is divided by its length. Taking into account the weighted vector space of the documents submission, they can be compared by calculating the distance between the points representing the documents. In other words, the similarity indicator is used in such a way as the documents with the highest scores will be the most similar to each other.

Three typical normalized similarity indicators are defined as follows [7]:

Similarity of cosines

$$S_{jk} = \frac{(v_j \cdot v_k)}{(\|v_j\| \cdot \| v_k\|)} \quad (4)$$

Jaquard similarity

$$S_{jk} = \frac{(v_j \cdot v_k)}{\frac{\sum_{i=1}^{n} w_{ij} + w_{ik}}{2^{w_{ij} w_{ik}}}} \quad (5)$$

The system's conclusion concerning documents similarity can be changed by both changing the weighing scheme and the degree of similarity.

4.2. Information interaction model

Information interaction and search (I2R) model is also chosen to implement the information leakage monitoring system. In data leakage detection system, I2R model can be implemented in the following way. Any web document is represented by an object. Each object $o_i \; i = 1, 2, \dots, M$ is associated with the vector of predefined index terms $t_i = (t_{ik}), k = 1, 2, \dots, n_i$. Connections

are established between any pair (o_i, o_j) $i \neq j$. Links are weighted and targeted. Two pairs of links can be identified in each direction. One directional link shows the relative frequency w_{ijp} of the index polynomial [8]. The relative frequency of the keyword in the document is defined as follows:

$$w_{ijp} = \frac{r_{ijp}}{n_i}, p = 1, \dots, n_j \quad (6)$$

where r_{ijp} indicates the relevance of the index term t_{jp} in the object of the web document o_i, n_i is the number of index terms in the web document object o_i.

Relationship between pairs of objects are schematically shown in Fig. 2.

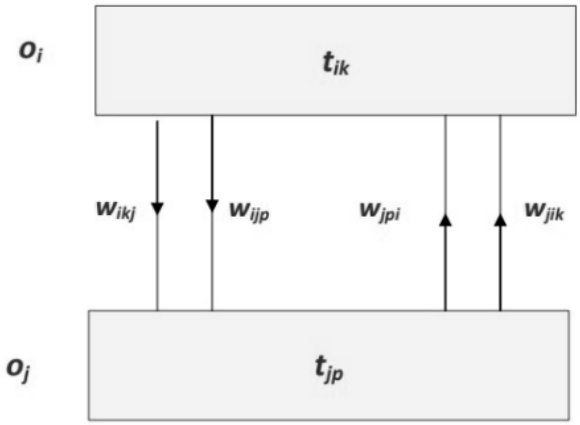

Figure 2: Relationship between pairs of objects for documents similarity determination

Another directional link is the reverse frequency of the document w_{ikj}. It is defined as follows:

$$w_{ikj} = r_{ijp} \log \frac{2M}{df_{ik}} \quad (7)$$

where r_{ijp} indicates the correspondence of the index term t_{ik} in o_j, df_{ik} is the number of web documents containing the keyword t_{ik}, and M is the number of objects of the web document. As it is shown in Figure 3, the other direction also has two links with the same value. The investigated web documents are presented in the form of complete graph of objects. The user's confidential document is also represented by the object. It is linked to other web documents that are in the full column. Linking this new object to the graph, the weights of the existing links will change. The weight of the link between any pair of objects $(o_j, o_i), i \neq$ i, and therefore between the confidential document and another object of the web document o_i, is defined as the sum of the corresponding weights, as shown below:

$$K_{ij} = \sum_{p=1}^{n_j} w_{jpi} + \sum_{k=1}^{n_j} w_{jik} \quad (8)$$

This complete graph of objects can be considered as an artificial neural network. In this network, the neurons activation is subjected to the dominant, which accepts all strategies. Activation begins with the user's confidential document and extends to the most active neuron. After several steps, the activation reaches the affected object. These web documents are similar to the confidential documents of users in this group. These documents can indicate the data leakage. The advantages of the interaction model are that this method avoids expensive calculations. The complexity of calculating weights is polynomial. The method of finding the interaction makes it possible to obtain relatively high accuracy within the range of 75% -85% on test collections.

5. Software implementation of the tool for automatic monitoring of information leakage

The set task of automatic monitoring of information leakage includes search, collection, analysis and classification of information from the Internet, which is very cumbersome work. The system should be able to analyze a large number of data of different formats, which come from a variety of sources on the network. Implementation of this system is presented in the form of modules:

- file analyzer module, which is responsible for reading useful data from a wide range of file types, as well as for searching for key words in these files to create a prediction about the file involvement in the source;
- modules for searching and collecting information - the number of modules is limited by the number of sources of information. Their task is to find and save files for further analysis.

Such modularity of this solution makes it possible to: perform them on different servers; conduct selection and analysis in Docker containers and Kubernetes clusters; use the hybrid approach to the location of servers; failure of one component does not affect the work of other components.

All these opportunities make it possible to maintain the maximum stability of work, give the chance to be expanded both vertically and horizontally, and enable to save money during work in cloud environments.

The scheme of operation principle of the monitoring system for information leakage is shown in Figure 3.

During the development it was decided to implement minimum two required components, which together build the system for collecting and analyzing information. The obligatory component is the file analyzer module, which should analyze the files in particular directory. For the second component the module for collecting information from Darknet is chosen for implementation.

Darknet is the global network by which users can access web resources that are not accessible through conventional search engines. Information in the Darknet is usually not available to the general public, and such information is deliberately hidden for the ordinary Internet, known as Clearnet.

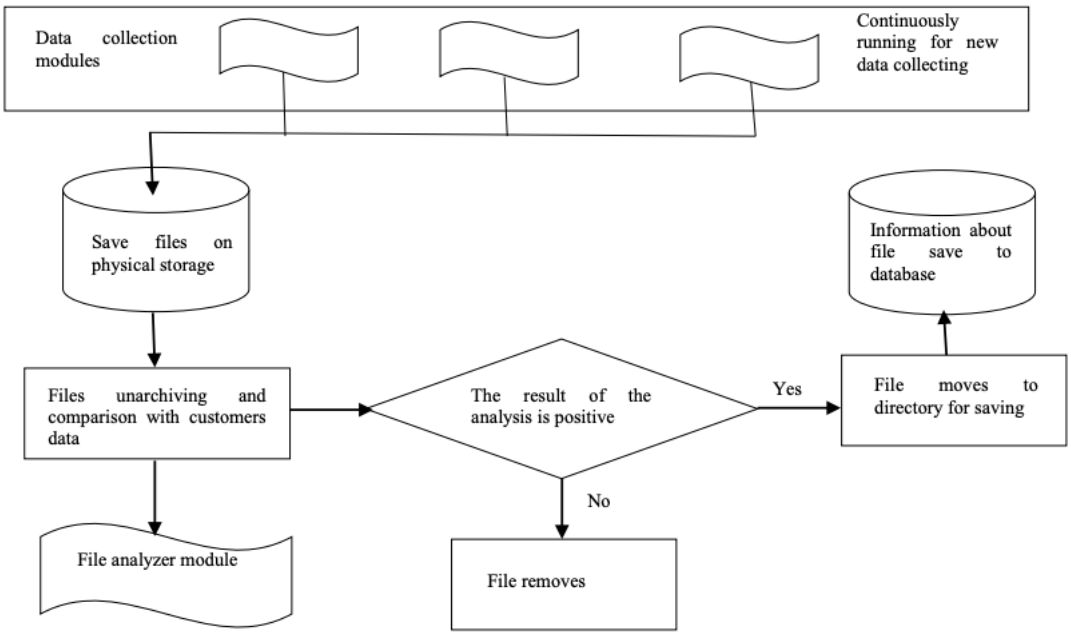

Figure 3. The scheme of operation principle of the information leakage monitoring system

The decision to choose this source is due to the sharp increase in the number of hacker attacks on corporate data. If companies do not pay ransom, all collected information becomes available to the public on the Darknet resource. To analyze possible sources of corporate information leakage, blogs of several ransomware-groups are selected.

To access the information on the Darknet page, you must use Tor software. When you run Tor utility, socks5h proxy is automatically created by means of which the program receive information. It is not necessary to run Tor in the browser, it is sufficient to run Tor as a service and allow it to run in the background. Thus, the implementation of this module is not limited to desktop computers, but can be run on servers where there is no graphical interface. For the Python programming language, there is a special Stem library [9], which enables you to control the operation of Tor service in the background. In order not to create problems with already running Tor services, the new launch will take place in specially created temporary directory, which will be deleted automatically at the end of the work.

The Deep_Web_Parser class is created to work with information on sites in the darknet network. This is the parent class and defines only the interfaces, the implementation of which is in the children's class ContiNews_Parser (for parsing the blog ransomware-group Conti). This unifies the method the data is accessed between different page parsers, making it possible to add easily a new source without breaking the program.

Undocumented API was found during the development of ContiNews parser. After getting acquainted with the principle of queries, the parser implementation was greatly simplified. There are two main methods of information parsing from the darknet web page:

- get – allows you to get information about the article;
- files – allows you to get information about files in the article.

After the information about all previously unprocessed publications has been processed and stored in the database, the information is passed over the socket to the file download module which is implemented in Watcher class. The part of the information retrieval server implementation can be seen in Listing 1. This module operates in multithreading mode, which makes it possible to receive information about new publications and download multiple files at once. This has a positive effect on performance, as the download speed of a single file over Tor network is very slow, and the queue for files to be downloaded can always be supplemented with new files.

Listing 1 - Data retrieval server implementation

```
    def _server_connection(self):
        """                         Methodisrunninginbackgroundinfinitelyandwaitsfornewqueueupdate,
afterreceivingvalidjson-dataifwillexecuteself.update_queue
        """
        # Create a TCP/IP socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # Bindthesockettotheport
    server_socket.bind((self.ip, self.port))
        # Listenforincomingconnectionsserver_socket.listen(1)
    whileTrue:
            # Generateemptydecodeddataforeveryconnection
    data_decoded = dict()           # Waitfor a connection
    connection, _ = server_socket.accept()          data = ''           try:
                # Receivethedatainsmallchunksandappendit whileTrue:                                  chunk =
connection.recv(1024).decode(encoding="utf-8")                  iflen(chunk) > 0:                   data
+= chunkif "END_CONVERSATION" indata:                                                               data =
data.replace("END_CONVERSATION", "")                    breakelse:
    break
    try:                data_decoded = json.loads(data)
    resp          =          f"DataReceived!          :          {data_decoded}END_CONVERSATION"
connection.sendall(resp.encode(encoding="utf-8"))           exceptException:                    resp =
f"Couldnotdecodedata, pleasedoublecheck:{data}"             logger.warning(resp)
```

```
    connection.sendall(resp.encode(encoding="utf-8"))
    finally:
            # Cleanuptheconnection connection.close()
    ifdata_decoded:                    ifdata_decoded.get("action", None) == "stop_server":
returnifself._download_thread_started:          self._stop_download()
    self._queue = self._generate_new_queue(data_decoded)          self._start_download()
```

During downloading, a separate Tor-proxy is created for each file, in order not to overload the same Tor access nodes. As the result, the download speed is limited only by the capabilities of the equipment on which the files are downloaded and the speed provided by Internet-provider. The function for file downloading is shown in Listing 2.

Listing 2 - The function of file downloading from darknet

```
    def _download_by_link(self, _file: dict, download_status: dict):
        # PrepearigTorproxies
    Tor_Session = Tor_Connector(socks_port=socks_port, control_port=control_port,
disable_init_msgs=True) sleep(2)
    proxy = Tor_Session.get_proxy()          # Creatingrequestssessionwithparams session =
requests.Session() session.proxies.update(proxy)
    save_path = f"{DOWNLOAD_BASE_PATH}{source_parser}/{company_name}/{file_name}"
whiledownloaded<total_length:          resume_header = {'Range': f'bytes={downloaded}-'}
session.headers.update(resume_header)

    res = session.get(url=url, allow_redirects=True)
    try:                                forchunkinres.iter_content(chunk_size=2048):
ifself._download_stop_flag:
    Tor_Session.stop_tor()
    return 0          try:          withopen(save_path, "ab+") as f:
    f.write(chunk)
    f.flush()
    downloaded += len(chunk)
                # XXX maybeeaddctrl + c signalcheckto
                # verifyexitortoproperlysavedata
    withopen(f"{save_path}.metadata", "w") asf_meta:                              try:
f_meta.write(str(downloaded))          f_meta.flush()
    exceptExceptionas e:                    logger.error("Cannotwritelenofdownloaded"
"datatometadatafile"                    f"Error: {e}")          exceptExceptionas e:
logger.error(f"CannotwritedatatofileError: {e}")          return 1
    exceptExceptionas e:
    logger.warning(f"Erroroccurredwhiledownloadingfile {file_name}, probablyconnection "
    f"crashed, tryingtocontinuein a momentError: {e}")
    Tor_Session.build_new_circuit()          sleep(5)          continue

    download_status.pop(file_name, None)
    os.remove(f"{save_path}.metadata")

    Tor_Session.stop_tor()

        # Deletingfinisheddownloadfromqueue. Byebye ;)
    q_download_links = [_q_obj.get("file", {}).get("remote_location", "") for _q_objinself._queue]
    self._queue.pop(q_download_links.index(url))
```

5.1. Software implementation of the file analyzer module

The implementation of the module starts with the definition of data types that will be analyzed. Due to the fact that the task is to find keywords in corporate documents, it is necessary to be able to read data from such formats as:

- doc, docb, docm та docx;
- xls,xlsb, xlsm та xlsx; - ppt, pptb та pptx.

In addition, support for popular data formats is implemented. These formats are:

- pdf; txt; png; jpg та jpeg; log; json.

In order to standardize the work, Analyzer class which contains implemented data access interfaces and algorithms for searching for keywords in the text, titles and metadata files is created. Textract library is used while retrieving from plain text formats, as well as retrieving metadata from files. The function of retrieving text from files is shown in Listing 3.

Listing 3 - Function for retrieving file contents

```
defget_content(self): try: returntextract.process(self.file_path).decode('utf-8').lower()
exceptUnicodeDecodeError: try:
withopen(self.file_path, 'rb') as f:
returnf.read().decode('utf-16').lower() exceptUnicodeDecodeError: try:
withopen(self.file_path, 'rb') as f: returnf.read().decode('utf-32').lower()
exceptUnicodeDecodeError:
logger.error(f'[Analyzing] {self.file_path} cannotbedecodein
anyformat (utf-8/utf-16/utf-32)')
self.extracting_error = "cannotbedecodeinanyformat (utf8/utf-16/utf-32)"
excepttextract.exceptions.ExtensionNotSupported: self.extracting_error = 'isnotsupported'
logger.warning(f'[Analyzing] {self.file_path} isnotsupported') exceptExceptionas e:
self.extracting_error = f'generatethefollowingexception {e}' logger.error(f'[Analyzing]
{self.file_path} hasgeneratethefollowing
exception {e}')
```

Due to the fact that pdf and word files often contain images that cannot be retrieved as separate text, the approach of text analysis from images is implemented. In order to do this, Office format files are converted to pdf using Libreoffice [25]. Then pdf files are converted into images using pdf2image library. Google's tesseract tool, i.e. pytesseract [26], is used to retrieve text from images. The example of text retrieval is given in Listing 4.

Listing 4 - Text retrieval from images

```
forpage_pathinpages:
text = str((pytesseract.image_to_string(Image.open(page_path)))) text = text.replace('-\n', '')
os.remove(page_path) content += text
```

Further, the file is analyzed for the content of keywords, domains, email addresses and bank card numbers, on the basis of this analysis the prediction if the document refers to data leakage is made. The implementation fragment is shown in Listing 5.

Listing 5 - Implementation of data leakage prediction

```
defpredict_breach(report, customer_keywords): prediction = 0
customer_keywords_size = len(customer_keywords)
content_keywords_size = len(report["event"]["content"]["keywords"]) title_keywords_size =
len(report["event"]["title"]["keywords"]) metadata_keywords_size =
len(report["event"]["metadata"]["keywords"])
content_keywords_value = 0.9 * (content_keywords_size/customer_keywords_size) prediction
+= content_keywords_value
```

```
    title_keywords_value = 0.05 * (title_keywords_size/customer_keywords_size)      prediction +=
title_keywords_value
    metadata_keywords_value    =    0.05    *    (metadata_keywords_size/customer_keywords_size)
prediction += metadata_keywords_value
    iflen(report["event"]["content"]["emails"]) > 5:
    prediction += len(report["event"]["content"]["emails"]) * 0.02
    iflen(report["event"]["content"]["credit_card"]) > 2:
    prediction += len(report["event"]["content"]["credit_card"]) * 0.02
        prediction        =        content_keywords_value        +        title_keywords_value    +
    metadata_keywords_value
    returnprediction
```

The result of the entire analysis is stored in the database, where the information can be obtained for further application.

5.2. Testing the functionality of the information leakage monitoring system

After the development and implementation of the new module in software product, functional testing is carried out.

Functional testing is carried out in the following areas:

- regression testing. Testing of the developed module after changes in it is carried out;
- modular testing. Testing of separate modules after their development;
- integration testing. Testing the correct modules interaction for information processing correctness.

In the process of testing the module for collecting and storing information by logging the successful scanning, information storage to Elasticsearch database, and successful transfer of information about publications to the download module are confirmed. The results are shown in Figure 4.

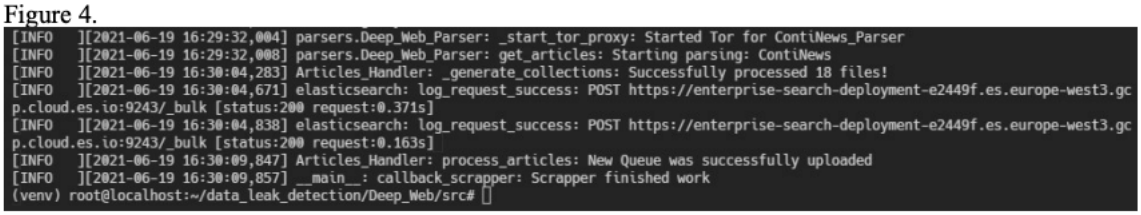

Figure 4: The results of collecting information about the article

The test of file download module, after receiving the list of information about new articles, successfully obtains additional information about files from the database, generates the download queue and starts downloading, this information is displayed in the execution logs, which is presented in Figure 5.

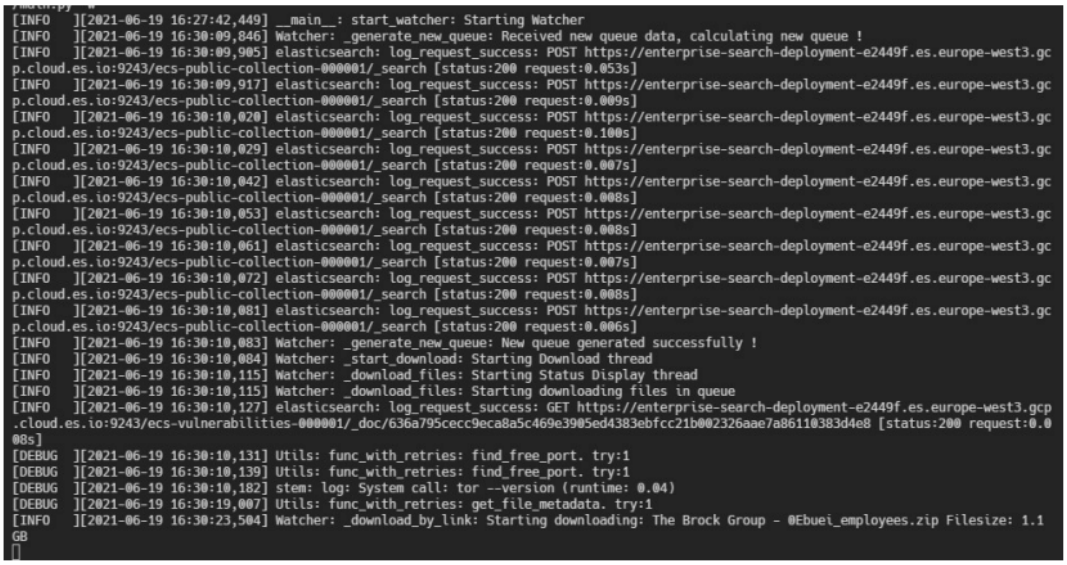

Figure 5: The results of file download module performance

In order to simulate the real example of information leakage, the keywords of one of the customers are added to random file. After this file scanning, the corresponding message is displayed in the logs. It is shown in Figure 6.

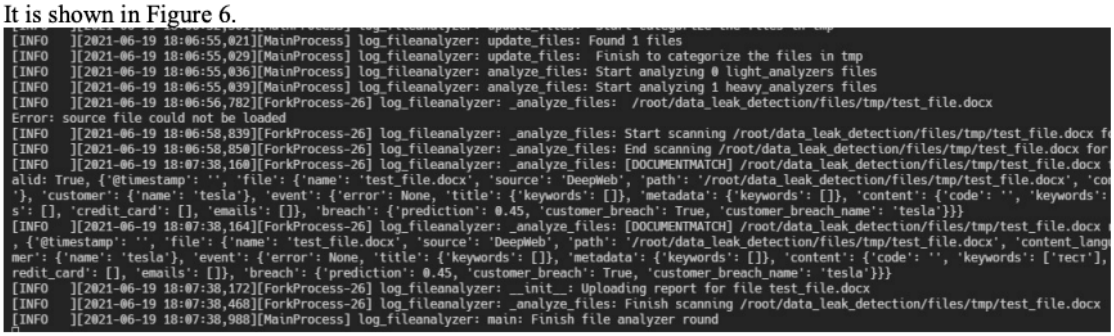

Figure 6: The example of scanning the file containing customer's keywords

Modular and integration testing of the system components showed excellent results and did not reveal any problems.

6. Conclusions and directions for further investigations

The developed system of automatic monitoring of corporate information leakage makes it possible to reveal the corporate information leakage as soon as possible and to minimize its consequences. At present the problem of hacker attacks on companies is very important all over the world. Different countries are taking measures to protect businesses from attacks by intruders in the digital space. However, the companies should take care of data security themselves.

The proposed development will be expanded in order to cover new sources of information (hacker forums, shadow Telegram channels, etc.). For some companies (design, design bureaus) graphic information is of primary importance. In further research it is necessary to investigate mathematical methods of similarity detection for graphic, audio, video files.

7. References

[1] R. Kirdat, N. Mokal, J. Mokal, A. Parkar, R. Shahabade. “Data Leakage Detection and File Monitoring in Cloud Computing” Internetional Journal of Advance Research, Ideas and Innovations in Technology. Volume 4, Issue 2, 2018. pp. 859-866.

[2] R. Naik and M. N. Gaonkar, "Data Leakage Detection in cloud using Watermarking Technique," 2019 International Conference on Computer Communication and Informatics (ICCCI), 2019, pp. 1-6, doi: 10.1109/ICCCI.2019.8821894.

[3] Chang, S.-H., Mallissery, S., Hsieh, C.-H., & Wu, Y.-S. (2018). Hypervisor-Based Sensitive Data Leakage Detector. 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS). pp. 155-162. doi:10.1109/qrs.2018.00029

[4] Tuan, L.H., Cam, N.T. & Pham, VH. Enhancing the accuracy of static analysis for detecting sensitive data leakage in Android by using dynamic analysis. Cluster Comput 22, 1079–1085 (2019). https://doi.org/10.1007/s10586-017-1364-8

[5] G. Kul, S. Upadhyaya and V. Chandola, "Detecting Data Leakage from Databases on Android Apps with Concept Drift," 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2018, pp. 905-913, doi: 10.1109/TrustCom/BigDataSE.2018.00129.

[6] R. Baeza-Yates, B. Ribeiro-Neto. Modern information retrieval: The Concepts and Technology behind Search (2nd Edition). ACM Press Books, 2011. pp. 53-55.

[7] C. T. Meadow, R. B.Bert, H. K.Donald. TextInformationRetrievalSystems, 2017. Print.

[8] S. Dominich. Connectionist interaction information retrieval. In: Information processing & management. 2003. pp. 167-193.

[9] Welcome to Stem! URL: https://stem.torproject.org/

Додаток Б Лістинг файлу docker-compose.prod.yaml

```
services:
  traefik:
    image: traefik:v2.11
    container_name: traefik
    restart: unless-stopped
    command:
      - --providers.docker=true
      - --providers.docker.exposedbydefault=false
      - --entrypoints.web.address=:80
      - --entrypoints.websecure.address=:443
      - --entrypoints.web.http.redirections.entrypoint.to=websecure
      - --entrypoints.web.http.redirections.entrypoint.scheme=https
      - --api.dashboard=true
      - --log.level=INFO
      - --accesslog=true
      - --certificatesresolvers.letsencrypt.acme.email=${ACME_EMAIL}
      - --certificatesresolvers.letsencrypt.acme.storage=/letsencrypt/acme.json
      - --certificatesresolvers.letsencrypt.acme.httpchallenge=true
      - --certificatesresolvers.letsencrypt.acme.httpchallenge.entrypoint=web
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - traefik-certificates:/letsencrypt
    networks:
      - monitoring
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.dashboard.rule=Host(`traefik.${DOMAIN}`)"
      - "traefik.http.routers.dashboard.entrypoints=websecure"
      - "traefik.http.routers.dashboard.tls.certresolver=letsencrypt"
      - "traefik.http.routers.dashboard.service=api@internal"
      - "traefik.http.routers.dashboard.middlewares=dashboard-auth"
      - "traefik.http.middlewares.dashboard-auth.basicauth.users=${DASHBOARD_AUTH}"
  grafana:
    image: grafana/grafana:11.3.1
    container_name: grafana
    restart: unless-stopped
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_PASSWORD:-admin}
      - GF_PATHS_PROVISIONING=/etc/grafana/provisioning
      - GF_LOG_LEVEL=info
    volumes:
      - grafana-data:/var/lib/grafana
      - ./grafana/provisioning:/etc/grafana/provisioning
    networks:
      - monitoring
      - telemetry
    labels:
      - "traefik.enable=true"
```

```
   - "traefik.http.routers.grafana.rule=Host(`grafana.${DOMAIN}`)"
    - "traefik.http.routers.grafana.entrypoints=websecure"
    - "traefik.http.routers.grafana.tls.certresolver=letsencrypt"
    - "traefik.http.services.grafana.loadbalancer.server.port=3000"
  loki:
    image: grafana/loki:3.3.1
    container_name: loki
    restart: unless-stopped
    command: -config.file=/etc/loki/config.yml
    volumes:
      - ./loki/config.yml:/etc/loki/config.yml
      - loki-data:/loki
    networks:
      - telemetry
  tempo:
    image: grafana/tempo:2.6.1
    container_name: tempo
    restart: unless-stopped
    command: -config.file=/etc/tempo/config.yml
    volumes:
      - ./tempo/config.yml:/etc/tempo/config.yml
      - tempo-data:/var/tempo
    networks:
      - telemetry
  prometheus:
    image: prom/prometheus:v3.0.1
    container_name: prometheus
    restart: unless-stopped
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/usr/share/prometheus/console_libraries'
      - '--web.console.templates=/usr/share/prometheus/consoles'
      - '--web.enable-remote-write-receiver'
    volumes:
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-data:/prometheus
    networks:
      - monitoring
      - telemetry
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.prometheus.rule=Host(`prometheus.${DOMAIN}`)"
      - "traefik.http.routers.prometheus.entrypoints=websecure"
      - "traefik.http.routers.prometheus.tls.certresolver=letsencrypt"
      - "traefik.http.services.prometheus.loadbalancer.server.port=9090"
      - "traefik.http.routers.prometheus.middlewares=dashboard-auth"
  alloy:
    image: grafana/alloy:v1.5.1
    container_name: alloy
    restart: unless-stopped
    command:
```

```
      - run
      - --server.http.listen-addr=0.0.0.0:12345
      - --storage.path=/var/lib/alloy/data
      - /etc/alloy/config.alloy
    volumes:
      - ./alloy/config.alloy:/etc/alloy/config.alloy
      - alloy-data:/var/lib/alloy/data
      - /var/run/docker.sock:/var/run/docker.sock:ro
    networks:
      - telemetry
  demo-app:
    build:
      context: ./demo-app
      dockerfile: Dockerfile
    container_name: demo-app
    restart: unless-stopped
    working_dir: /app
    command: ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000", "--no-access-log"]
    volumes:
      - ./demo-app:/app
    networks:
      - monitoring
      - telemetry
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.demo-app.rule=Host(`demo.${DOMAIN}`)"
      - "traefik.http.routers.demo-app.entrypoints=websecure"
      - "traefik.http.routers.demo-app.tls.certresolver=letsencrypt"
      - "traefik.http.services.demo-app.loadbalancer.server.port=8000"

networks:
  monitoring:
    driver: bridge
    name: monitoring-network
  telemetry:
    driver: bridge
    name: telemetry-network

volumes:
  traefik-certificates:
    name: traefik-certificates
  grafana-data:
    name: grafana-data
  loki-data:
    name: loki-data
  tempo-data:
    name: tempo-data
  prometheus-data:
    name: prometheus-data
  alloy-data:
    name: alloy-data
```

Додаток В Лістинг файлу demo-app.py

```
"""
Demo FastAPI application with observability and simulated authentication
"""
import asyncio
import json
import logging
import os
import random
import sys
import time
import uuid
from contextlib import asynccontextmanager
from typing import Optional

from fastapi import FastAPI, Form, Request, Response, status
from fastapi.responses import HTMLResponse, RedirectResponse
from itsdangerous import BadSignature, URLSafeTimedSerializer
from prometheus_client import Counter, Histogram, generate_latest, CONTENT_TYPE_LATEST
from opentelemetry import trace
from opentelemetry.exporter.otlp.proto.http.trace_exporter import OTLPSpanExporter
from opentelemetry.instrumentation.fastapi import FastAPIInstrumentor
from opentelemetry.sdk.resources import Resource
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.trace import Status, StatusCode

# ============================================================================
# Configuration
# ============================================================================

SERVICE_NAME = os.getenv("SERVICE_NAME", "demo-app")
APP_SECRET = os.getenv("APP_SECRET", "change-me-in-production-please")
DEMO_PASSWORD = os.getenv("DEMO_PASSWORD", "password123")
COOKIE_SECURE = os.getenv("COOKIE_SECURE", "false").lower() == "true"
LOGIN_TIMEOUT_SECONDS = int(os.getenv("LOGIN_TIMEOUT_SECONDS", "25"))
DB_BASE_MS = int(os.getenv("DB_BASE_MS", "100"))
DB_PER_CHAR_MS = int(os.getenv("DB_PER_CHAR_MS", "1000"))  # 1 second per character
DB_MAX_MS = int(os.getenv("DB_MAX_MS", "25000"))
OTEL_ENDPOINT = os.getenv("OTEL_EXPORTER_OTLP_ENDPOINT", "http://alloy:4318")

# Simple user database (in-memory)
USERS = {
    "admin": DEMO_PASSWORD,
    "user": DEMO_PASSWORD,
}

# ============================================================================
# Logging Setup (JSON structured)
# ============================================================================

class JSONFormatter(logging.Formatter):
    """Custom JSON formatter for structured logging"""
```

```
def format(self, record):
    log_data = {
        "timestamp": self.formatTime(record, self.datefmt),
        "level": record.levelname,
        "message": record.getMessage(),
        "service.name": SERVICE_NAME,
    }

    # Add extra fields if present
    if hasattr(record, "request_id"):
        log_data["request_id"] = record.request_id
    if hasattr(record, "trace_id"):
        log_data["trace_id"] = record.trace_id
    if hasattr(record, "span_id"):
        log_data["span_id"] = record.span_id
    if hasattr(record, "client_ip"):
        log_data["client_ip"] = record.client_ip
    if hasattr(record, "method"):
        log_data["method"] = record.method
    if hasattr(record, "path"):
        log_data["path"] = record.path
    if hasattr(record, "status_code"):
        log_data["status_code"] = record.status_code
    if hasattr(record, "duration_ms"):
        log_data["duration_ms"] = record.duration_ms
    if hasattr(record, "user_agent"):
        log_data["user_agent"] = record.user_agent
    if hasattr(record, "event"):
        log_data["event"] = record.event
    if hasattr(record, "username"):
        log_data["username"] = record.username
    if hasattr(record, "result"):
        log_data["result"] = record.result
    if hasattr(record, "payload_size_bytes"):
        log_data["payload_size_bytes"] = record.payload_size_bytes
    if hasattr(record, "db_sleep_ms"):
        log_data["db_sleep_ms"] = record.db_sleep_ms
    if hasattr(record, "http_referer"):
        log_data["http_referer"] = record.http_referer
    if hasattr(record, "note_length"):
        log_data["note_length"] = record.note_length
    if hasattr(record, "password_length"):
        log_data["password_length"] = record.password_length
    if hasattr(record, "username_length"):
        log_data["username_length"] = record.username_length
    if hasattr(record, "user_exists"):
        log_data["user_exists"] = record.user_exists
    if hasattr(record, "timeout_seconds"):
        log_data["timeout_seconds"] = record.timeout_seconds
    if hasattr(record, "has_session_cookie"):
        log_data["has_session_cookie"] = record.has_session_cookie
    if hasattr(record, "endpoint"):
```

```
            log_data["endpoint"] = record.endpoint
        if hasattr(record, "note"):
            log_data["note"] = record.note

        return json.dumps(log_data)

# Configure root logger
logger = logging.getLogger()
logger.setLevel(logging.INFO)
handler = logging.StreamHandler(sys.stdout)
handler.setFormatter(JSONFormatter())
logger.handlers = [handler]

app_logger = logging.getLogger(__name__)

# ============================================================================
# Prometheus Metrics
# ============================================================================

http_requests_total = Counter(
    "http_requests_total",
    "Total HTTP requests",
    ["method", "path", "status"]
)

http_request_duration_seconds = Histogram(
    "http_request_duration_seconds",
    "HTTP request duration in seconds",
    ["method", "path"]
)

login_attempts_total = Counter(
    "login_attempts_total",
    "Total login attempts",
    ["result"]
)

authz_denied_total = Counter(
    "authz_denied_total",
    "Total authorization denials"
)

db_simulated_query_duration_seconds = Histogram(
    "db_simulated_query_duration_seconds",
    "Simulated DB query duration"
)

db_simulated_query_payload_bytes = Histogram(
    "db_simulated_query_payload_bytes",
    "Simulated DB query payload size",
    buckets=[100, 500, 1000, 5000, 10000, 50000, 100000, 500000, 1000000]
)
```

```
# ================================================================================
# OpenTelemetry Setup
# ================================================================================

def setup_tracing():
    """Initialize OpenTelemetry tracing"""
    resource = Resource.create({"service.name": SERVICE_NAME})

    provider = TracerProvider(resource=resource)

    # OTLP HTTP exporter
    # Explicitly specify the full traces endpoint
    traces_endpoint = f"{OTEL_ENDPOINT}/v1/traces" if not OTEL_ENDPOINT.endswith("/v1/traces") else
OTEL_ENDPOINT
    otlp_exporter = OTLPSpanExporter(
        endpoint=traces_endpoint
    )

    provider.add_span_processor(BatchSpanProcessor(otlp_exporter))
    trace.set_tracer_provider(provider)

    app_logger.info("OpenTelemetry tracing initialized", extra={
        "event": "startup.tracing",
        "endpoint": traces_endpoint
    })

tracer = trace.get_tracer(__name__)

# ================================================================================
# Session Management
# ================================================================================

serializer = URLSafeTimedSerializer(APP_SECRET)

def create_session_cookie(username: str) -> str:
    """Create a signed session cookie"""
    return serializer.dumps({"username": username})

def verify_session_cookie(cookie: Optional[str]) -> Optional[str]:
    """Verify session cookie and return username if valid"""
    if not cookie:
        return None
    try:
        data = serializer.loads(cookie, max_age=3600)  # 1 hour expiry
        return data.get("username")
    except BadSignature:
        return None

# ================================================================================
# Simulated Database
# ================================================================================

async def simulate_db_query(username: str, password: str, note: str) -> bool:
```

```
    """
    Simulate a database query with latency that grows with character count.
    Each character in the note adds DB_PER_CHAR_MS (default 1 second).
    Returns True if credentials are valid.
    """
    span = trace.get_current_span()

    # Calculate payload size (bytes for metrics, chars for latency)
    payload_bytes = len(username.encode("utf-8")) + len(password.encode("utf-8")) + len(note.encode("utf-8"))
    total_chars = len(note)  # Only count 'note' field for dramatic effect

    # Calculate sleep time based on character count: 1 char = +1 second (dramatic for demos)
    jitter = random.randint(-50, 50)
    sleep_ms = DB_BASE_MS + (total_chars * DB_PER_CHAR_MS) + jitter
    sleep_ms = min(sleep_ms, DB_MAX_MS)
    sleep_ms = max(sleep_ms, 0)

    # Record metrics
    db_simulated_query_payload_bytes.observe(payload_bytes)

    # Create a child span for DB query
    with tracer.start_as_current_span("db.query.simulated") as db_span:
        db_span.set_attribute("db.system", "simulated")
        db_span.set_attribute("db.operation", "SELECT")
        db_span.set_attribute("payload.size_bytes", payload_bytes)
        db_span.set_attribute("payload.chars", total_chars)
        db_span.set_attribute("sleep.ms", sleep_ms)

        start_time = time.time()

        # Simulate async DB query
        await asyncio.sleep(sleep_ms / 1000.0)

        duration = time.time() - start_time
        db_simulated_query_duration_seconds.observe(duration)

        # Check credentials
        is_valid = username in USERS and USERS[username] == password
        db_span.set_attribute("auth.result", "success" if is_valid else "fail")

        return is_valid

# ============================================================================
# HTML Templates
# ============================================================================

LOGIN_PAGE_TEMPLATE = """
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login - Demo App</title>
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-gradient-to-br from-blue-50 to-indigo-100 min-h-screen flex items-center justify-center p-4">
    <div class="bg-white rounded-lg shadow-xl p-8 max-w-md w-full">
        <div class="text-center mb-8">
            <h1 class="text-3xl font-bold text-gray-800 mb-2">Demo Application</h1>
            <p class="text-gray-600">Observability Stack Login</p>
        </div>

        {error_banner}

        <form method="POST" action="/login" class="space-y-6">
            <div>
                <label for="username" class="block text-sm font-medium text-gray-700 mb-2">Username</label>
                <input
                    type="text"
                    id="username"
                    name="username"
                    required
                    class="w-full px-4 py-2 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-
transparent"
                    placeholder="Enter username"
                >
            </div>

            <div>
                <label for="password" class="block text-sm font-medium text-gray-700 mb-2">Password</label>
                <input
                    type="password"
                    id="password"
                    name="password"
                    required
                    class="w-full px-4 py-2 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-
transparent"
                    placeholder="Enter password"
                >
            </div>

            <div>
                <label for="note" class="block text-sm font-medium text-gray-700 mb-2">
                    Extra Context <span class="text-gray-400 text-xs">(optional - simulated database load)</span>
                </label>
                <textarea
                    id="note"
                    name="note"
                    rows="4"
                    class="w-full px-4 py-2 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-
transparent"
                    placeholder="Add text here to simulate slow database queries..."
```

Продовження Додатку В

```
            ></textarea>
            <div class="mt-2 p-3 bg-amber-50 border-l-4 border-amber-400 rounded">
              <div class="flex items-start">
                <svg class="w-5 h-5 text-amber-600 mt-0.5 mr-2 flex-shrink-0" fill="currentColor" viewBox="0 0 20 20">
                  <path fill-rule="evenodd" d="M8.257 3.099c.765-1.36 2.722-1.36 3.486 0l5.58 9.92c.75 1.334-.213
2.98-1.742 2.98H4.42c-1.53 0-2.493-1.646-1.743-2.98l5.58-9.92zM11 13a1 1 0 11-2 0 1 1 0 012 0zm-1-8a1 1 0 00-1
1v3a1 1 0 002 0V6a1 1 0 00-1-1z" clip-rule="evenodd"></path>
                </svg>
                <div class="text-xs text-amber-800">
                  <p class="font-semibold">⚡ Performance Impact:</p>
                  <p class="mt-1"><strong>Each character = +1 second</strong> database query time</p>
                  <p class="mt-1">• 5 chars → 5 second delay</p>
                  <p>• 10 chars → 10 second delay</p>
                  <p>• 20+ chars → Request timeout (25s)</p>
                </div>
              </div>
            </div>
          </div>

          <button
            type="submit"
            class="w-full bg-blue-600 hover:bg-blue-700 text-white font-semibold py-3 px-4 rounded-lg transition duration-
200 shadow-md hover:shadow-lg"
          >
            Sign In
          </button>
        </form>

        <div class="mt-6 text-center text-sm text-gray-600">
          <p>Demo credentials: <code class="bg-gray-100 px-2 py-1 rounded">admin</code> / <code class="bg-gray-100
px-2 py-1 rounded">password123</code></p>
        </div>
      </div>
    </body>
    </html>
    """

    PROTECTED_PAGE_TEMPLATE = """
    <!DOCTYPE html>
    <html lang="en">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Protected Area - Demo App</title>
      <script src="https://cdn.tailwindcss.com"></script>
    </head>
    <body class="bg-gradient-to-br from-green-50 to-emerald-100 min-h-screen flex items-center justify-center p-4">
      <div class="bg-white rounded-lg shadow-xl p-8 max-w-md w-full">
        <div class="text-center mb-8">
          <div class="inline-flex items-center justify-center w-16 h-16 bg-green-100 rounded-full mb-4">
            <svg class="w-8 h-8 text-green-600" fill="none" stroke="currentColor" viewBox="0 0 24 24">
              <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M5 13l4 4L19 7"></path>
            </svg>
```

Продовження Додатку В

```
            </div>
            <h1 class="text-3xl font-bold text-gray-800 mb-2">Welcome, {username}!</h1>
            <p class="text-gray-600">You have successfully authenticated</p>
        </div>

        <div class="bg-blue-50 border-l-4 border-blue-500 p-4 mb-6">
            <div class="flex">
                <div class="flex-shrink-0">
                    <svg class="h-5 w-5 text-blue-500" fill="currentColor" viewBox="0 0 20 20">
                        <path fill-rule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7-4a1 1 0 11-2 0 1 1 0 012 0zM9 9a1
1 0 000 2v3a1 1 0 001 1h1a1 1 0 100-2v-3a1 1 0 00-1-1H9z" clip-rule="evenodd"></path>
                    </svg>
                </div>
                <div class="ml-3">
                    <p class="text-sm text-blue-700">
                        This is a protected area. Access is logged and monitored for security purposes.
                    </p>
                </div>
            </div>
        </div>

        <div class="space-y-4">
            <div class="bg-gray-50 rounded-lg p-4">
                <h3 class="font-semibold text-gray-700 mb-2">🔍 Observability Features:</h3>
                <ul class="text-sm text-gray-600 space-y-1">
                    <li>✓ Structured JSON logging</li>
                    <li>✓ Distributed tracing (OpenTelemetry)</li>
                    <li>✓ Prometheus metrics</li>
                    <li>✓ Request correlation</li>
                </ul>
            </div>

            <form method="POST" action="/logout" class="w-full">
                <button
                    type="submit"
                    class="w-full bg-gray-600 hover:bg-gray-700 text-white font-semibold py-3 px-4 rounded-lg transition
duration-200"
                >
                    Sign Out
                </button>
            </form>
        </div>
    </div>
</body>
</html>
"""

def render_login_page(error: Optional[str] = None) -> str:
    """Render the login page with optional error message"""
    error_banner = ""
    if error:
        error_banner = f"""
        <div class="bg-red-50 border-l-4 border-red-500 p-4 mb-6">
```

Продовження Додатку В

```
        <div class="flex">
          <div class="flex-shrink-0">
            <svg class="h-5 w-5 text-red-500" fill="currentColor" viewBox="0 0 20 20">
              <path fill-rule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zM8.707 7.293a1 1 0 00-1.414
1.414L8.586 10l-1.293 1.293a1 1 0 101.414 1.414L10 11.414l1.293 1.293a1 1 0 001.414-1.414L11.414 10l1.293-
1.293a1 1 0 00-1.414-1.414L10 8.586 8.707 7.293z" clip-rule="evenodd"></path>
            </svg>
          </div>
          <div class="ml-3">
            <p class="text-sm text-red-700 font-medium">{error}</p>
          </div>
        </div>
      </div>
      """
    return LOGIN_PAGE_TEMPLATE.replace("{error_banner}", error_banner)

def render_protected_page(username: str) -> str:
    """Render the protected page"""
    return PROTECTED_PAGE_TEMPLATE.replace("{username}", username)

# ============================================================================
# Middleware
# ============================================================================

@asynccontextmanager
async def lifespan(app: FastAPI):
    """Application lifespan events"""
    setup_tracing()
    app_logger.info("Application starting", extra={"event": "startup"})
    yield
    app_logger.info("Application shutting down", extra={"event": "shutdown"})


app = FastAPI(title="Demo App", lifespan=lifespan)

# Instrument FastAPI with OpenTelemetry
FastAPIInstrumentor.instrument_app(app)

def get_client_ip(request: Request) -> str:
    """Get real client IP, accounting for reverse proxy headers"""
    # Check for X-Forwarded-For header (set by reverse proxies like Traefik)
    forwarded_for = request.headers.get("X-Forwarded-For")
    if forwarded_for:
        # X-Forwarded-For can contain multiple IPs, take the first one (original client)
        return forwarded_for.split(",")[0].strip()

    # Fallback to X-Real-IP
    real_ip = request.headers.get("X-Real-IP")
    if real_ip:
        return real_ip

    # Last resort: direct connection IP
    return request.client.host if request.client else "unknown"
```

```
@app.middleware("http")
async def logging_middleware(request: Request, call_next):
    """Add request ID, logging, and metrics to all requests"""
    request_id = str(uuid.uuid4())
    start_time = time.time()

    # Add request ID to request state
    request.state.request_id = request_id

    # Get trace context
    span = trace.get_current_span()
    trace_id = None
    span_id = None

    if span and span.get_span_context().is_valid:
        trace_id = format(span.get_span_context().trace_id, "032x")
        span_id = format(span.get_span_context().span_id, "016x")

    # Process request
    response = await call_next(request)

    # Add request ID header
    response.headers["X-Request-Id"] = request_id

    # Calculate duration
    duration_ms = (time.time() - start_time) * 1000

    # Get client info
    client_ip = get_client_ip(request)
    user_agent = request.headers.get("user-agent", "unknown")

    # Log request
    log_extra = {
        "request_id": request_id,
        "client_ip": client_ip,
        "method": request.method,
        "path": request.url.path,
        "status_code": response.status_code,
        "duration_ms": round(duration_ms, 2),
        "user_agent": user_agent,
    }

    if trace_id:
        log_extra["trace_id"] = trace_id
        log_extra["span_id"] = span_id

    app_logger.info(f"{request.method} {request.url.path} - {response.status_code}", extra=log_extra)

    # Record metrics
    http_requests_total.labels(
        method=request.method,
        path=request.url.path,
        status=response.status_code
```

```
    ).inc()

    http_request_duration_seconds.labels(
        method=request.method,
        path=request.url.path
    ).observe(duration_ms / 1000.0)

    return response

# ============================================================================
# Routes
# ============================================================================

@app.get("/", response_class=HTMLResponse)
async def login_page():
    """Render the login page"""
    return render_login_page()

@app.post("/login", response_class=HTMLResponse)
async def login(
    request: Request,
    username: str = Form(...),
    password: str = Form(...),
    note: str = Form(default="")
):
    """Handle login submission with simulated DB and timeout"""
    client_ip = get_client_ip(request)
    user_agent = request.headers.get("user-agent", "unknown")
    request_id = request.state.request_id

    # Get trace context
    span = trace.get_current_span()
    trace_id = None
    if span and span.get_span_context().is_valid:
        trace_id = format(span.get_span_context().trace_id, "032x")

    # Calculate payload size
    payload_size = len(username.encode("utf-8")) + len(password.encode("utf-8")) + len(note.encode("utf-8"))

    # Create auth span
    with tracer.start_as_current_span("auth.verify_credentials") as auth_span:
        auth_span.set_attribute("username", username)
        auth_span.set_attribute("payload.size_bytes", payload_size)

        try:
            # Wrap DB query with timeout
            is_valid = await asyncio.wait_for(
                simulate_db_query(username, password, note),
                timeout=LOGIN_TIMEOUT_SECONDS
            )

            if is_valid:
                # Success
```

```
        auth_span.set_status(Status(StatusCode.OK))
        login_attempts_total.labels(result="success").inc()

        # Calculate actual sleep time for logging (same formula as simulate_db_query)
        total_chars = len(note)
        sleep_ms = min(
            DB_BASE_MS + (total_chars * DB_PER_CHAR_MS),
            DB_MAX_MS
        )

        app_logger.info("Login successful", extra={
            "event": "auth.login",
            "request_id": request_id,
            "trace_id": trace_id,
            "username": username,
            "result": "success",
            "payload_size_bytes": payload_size,
            "db_sleep_ms": round(sleep_ms, 2),
            "client_ip": client_ip,
            "user_agent": user_agent,
            "http_referer": request.headers.get("referer", "direct"),
            "note_length": len(note),
            "password_length": len(password),
            "username_length": len(username),
        })

        # Create session cookie
        session_cookie = create_session_cookie(username)
        response = RedirectResponse(url="/protected", status_code=status.HTTP_303_SEE_OTHER)
        response.set_cookie(
            key="session",
            value=session_cookie,
            httponly=True,
            secure=COOKIE_SECURE,
            samesite="lax",
            max_age=3600
        )
        return response
    else:
        # Invalid credentials
        auth_span.set_status(Status(StatusCode.ERROR, "Invalid credentials"))
        login_attempts_total.labels(result="fail").inc()

        app_logger.warning("Login failed - invalid credentials", extra={
            "event": "auth.login",
            "request_id": request_id,
            "trace_id": trace_id,
            "username": username,
            "result": "fail",
            "payload_size_bytes": payload_size,
            "client_ip": client_ip,
            "user_agent": user_agent,
            "http_referer": request.headers.get("referer", "direct"),
```

```
                "note_length": len(note),
                "password_length": len(password),
                "username_length": len(username),
                "user_exists": username in USERS,
            })

            return HTMLResponse(
                content=render_login_page(error="Invalid username or password"),
                status_code=status.HTTP_401_UNAUTHORIZED
            )

        except asyncio.TimeoutError:
            # Timeout
            auth_span.set_status(Status(StatusCode.ERROR, "Request timeout"))
            auth_span.add_event("timeout", {"timeout_seconds": LOGIN_TIMEOUT_SECONDS})
            login_attempts_total.labels(result="timeout").inc()

            app_logger.error("Login timeout", extra={
                "event": "auth.login",
                "request_id": request_id,
                "trace_id": trace_id,
                "username": username,
                "result": "timeout",

                "payload_size_bytes": payload_size,
                "timeout_seconds": LOGIN_TIMEOUT_SECONDS,
                "client_ip": client_ip,
                "user_agent": user_agent,
                "http_referer": request.headers.get("referer", "direct"),
                "note_length": len(note),
                "password_length": len(password),
                "username_length": len(username),
            })

            return HTMLResponse(
                content=render_login_page(
                    error=f"Request timed out after {LOGIN_TIMEOUT_SECONDS} seconds. Try with less data."
                ),
                status_code=status.HTTP_504_GATEWAY_TIMEOUT
            )

@app.get("/protected", response_class=HTMLResponse)
async def protected_page(request: Request):
    """Protected page that requires authentication"""
    session_cookie = request.cookies.get("session")
    username = verify_session_cookie(session_cookie)

    if not username:
        # Unauthorized
        authz_denied_total.inc()

        client_ip = get_client_ip(request)
        user_agent = request.headers.get("user-agent", "unknown")
```

```
            app_logger.warning("Authorization denied - invalid session", extra={
                "event": "auth.denied",
                "request_id": request.state.request_id,
                "path": "/protected",
                "client_ip": client_ip,
                "user_agent": user_agent,
                "http_referer": request.headers.get("referer", "direct"),
                "has_session_cookie": "session" in request.cookies,
            })

            return RedirectResponse(url="/", status_code=status.HTTP_303_SEE_OTHER)

        return render_protected_page(username)

    @app.post("/logout")
    async def logout():
        """Clear session and redirect to login"""
        response = RedirectResponse(url="/", status_code=status.HTTP_303_SEE_OTHER)
        response.delete_cookie(key="session")
        return response

    @app.get("/metrics")
    async def metrics():
        """Prometheus metrics endpoint"""
        return Response(content=generate_latest(), media_type=CONTENT_TYPE_LATEST)

    @app.get("/health")
    async def health():
        """Health check endpoint"""
        return {"status": "healthy", "service": SERVICE_NAME}

    if __name__ == "__main__":
        import uvicorn
        uvicorn.run(
            "app:app",
            host="0.0.0.0",
            port=8000,
            log_config=None,  # Disable uvicorn's logging, we handle it
        )
```