

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної

інженерії та кібербезпеки

Андрій Софіменко

2 лютого 2026 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра

на тему: «Проект лабораторного стенду для
дослідження та аналізу безпеки веб-додатків»

Термін здачі студентом закінченої роботи 15 червня 2026 р.

Вихідні дані роботи:

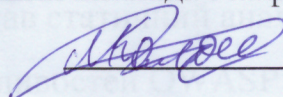
- еталонний вразливий веб-додаток OWASP Juice Shop;
- стандарти класифікації вразливостей OWASP Top 10 (2021) та MITRE CWE;
- інструменти аналізу безпеки GitHub CodeQL (SAST) та OWASP ZAP (DAST);
- хмарне середовище GitHub Actions;
- мова програмування Python 3.3 та бібліотека Flask (Flask, Recall, F1-Score);
- система матричного тестування OWASP Fuzzing Tool.

Виконав: студент 4-го курсу групи КБ-22-1

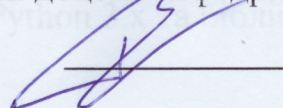
спеціальності 125 «Кібербезпека»

 Олександр ПЯТАКОВ

Керівник: старший викладач кафедри КІ та КБ

 Марія КОЛОЩУК

Рецензент: к.т.н. доцент кафедри КІ та КБ

 Олександр ПІРОГ

Житомир -2026

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко
2 лютого 2026 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Пятаков Олександр Олександрович

Тема роботи: «Проект лабораторного стенду для дослідження та аналізу безпеки веб-додатків»

затверджена Наказом університету від 2 лютого 2026 р. № (58/с)

Термін здачі студентом закінченої роботи 15 червня 2026 р.

Вихідні дані роботи:

- еталонний вразливий веб-додаток OWASP Juice Shop;
- стандарти класифікації вразливостей OWASP Top 10 (2021) та MITRE CWE;
- інструменти аналізу безпеки GitHub CodeQL (SAST) та OWASP ZAP (DAST);
- хмарне середовище GitHub Actions;
- мова програмування Python 3.x та бібліотека FPDF;
- система метрик якості детектування (Precision, Recall, F1-Score).

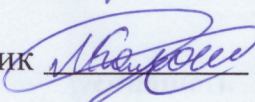
Обладнання та програмне забезпечення:

1. Інструментом аналізу безпеки став статичний аналізатор коду GitHub CodeQL та динамічний сканер вразливостей OWASP ZAP.
2. Засобами реалізації стали мова Python 3.x та бібліотека FPDF для генерації звітів.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Дацюк Д.В.	20.02.2026	20.02.2026
2	Дацюк Д.В.	20.02.2026	20.02.2026
3	Дацюк Д.В.	20.02.2026	20.02.2026

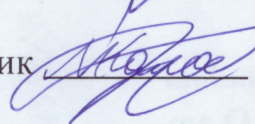
Студент  Олександр ПЯТАКОВ

Керівник  Марія КОЛОЩУК

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання	Примітка
1	Отримання та аналіз завдання, постановка мети й задач дослідження	03.02.2026 – 14.02.2026	Виконано
2	Аналіз стану безпеки веб-додатків, огляд методів SAST і DAST та гібридних підходів	15.02.2026 – 10.03.2026	Виконано
3	Розробка концептуальної моделі та формальної моделі гібридної системи, обґрунтування метрик якості	11.03.2026 – 05.04.2026	Виконано
4	Розгортання лабораторного стенду, налаштування CodeQL та OWASP ZAP, програмна реалізація модуля агрегації	06.04.2026 – 30.04.2026	Виконано
5	Проведення експериментів, аналіз результатів та оцінка ефективності	01.05.2026 – 18.05.2026	Виконано
6	Оформлення пояснювальної записки, формулювання висновків	19.05.2026 – 02.06.2026	Виконано
7	Захист	15.06.2026	Виконано

Студент  Олександр ПЯТАКОВ

Керівник  Марія КОЛОЩУК

АНОТАЦІЯ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра містить 73 сторінки, 16 рисунків, 4 таблиць. Список використаних джерел складається з 20 найменувань і розміщений на 2 сторінках.

Метою кваліфікаційної роботи є проектування лабораторного стенду та розробка методики дослідження захищеності веб-додатків на основі гібридного поєднання статичного й динамічного аналізу (SAST+DAST).

У процесі виконання кваліфікаційної роботи отримано такі основні результати:

1. проаналізовано сучасний стан безпеки веб-додатків та класифікацію критичних вразливостей згідно з міжнародним стандартом OWASP Top 10;
2. розглянуто особливості та обмеження методів статичного (SAST) і динамічного (DAST) аналізу захищеності при їх ізольованому використанні;
3. визначено основні вимоги та завдання для побудови архітектури інтеграції інструментів безпеки у єдиний аналітичний DevSecOps-конвеєр;
4. показано розробку та налаштування автоматизованого CI/CD конвеєра у хмарному середовищі GitHub Actions з інтеграцією сканерів GitHub CodeQL та OWASP ZAP;
5. розроблено авторський програмний модуль-агрегатор мовою Python для консолідації результатів сканування, усунення дублікатів та автоматичної генерації PDF-звітів;
6. наведено результати порівняльного експериментального тестування гібридної системи на базі еталонного вразливого додатка OWASP Juice Shop, що на модельних даних демонструє підвищення повноти виявлення загроз у порівнянні з ізольованими підходами в умовах розробленого лабораторного стенду.

Ключові слова: КІБЕРБЕЗПЕКА, ВЕБ-ДОДАТОК, DEVSECOPS, SAST, DAST, OWASP ZAP, GITHUB CODEQL, CI/CD, АГРЕГАЦІЯ ВРАЗЛИВОСТЕЙ.

					KKIKB.KP.B. - 125 - 13 - ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Пятаков О.О.		8.06	Проект лабораторного стенду для дослідження та аналізу безпеки веб-додатків	Літ.	Арк.
Керівник		Колощук М.С.		8.06			4
Реценз.		Пірог О.В.		8.06			73
Н. контр.		Колощук М.С.		8.06		ФІКТ, Гр. КБ-22-1	
Затверд.		Єфіменко А.А.		8.06.26			

ANNOTATION

The explanatory note to the Bachelor's degree thesis comprises 73 pages, 16 figures and 4 tables. The bibliography consists of 20 entries and spans 2 pages.

The aim of the thesis is to design a laboratory test bench and develop a methodology for testing the security of web applications based on a hybrid combination of static and dynamic analysis (SAST+DAST).

The following main results were obtained during the course of the thesis:

1. the current state of web application security and the classification of critical vulnerabilities in accordance with the international OWASP Top 10 standard were analysed;
2. the features and limitations of static (SAST) and dynamic (DAST) security analysis methods when used in isolation were examined;
3. the main requirements and tasks for building an architecture to integrate security tools into a single DevSecOps analysis pipeline were identified;
4. the development and configuration of an automated CI/CD pipeline in the GitHub Actions cloud environment, with integration of the GitHub CodeQL and OWASP ZAP scanners, is demonstrated;
5. a proprietary aggregator software module written in Python has been developed to consolidate scan results, eliminate duplicates and automatically generate PDF reports;
6. presents the results of comparative experimental testing of a hybrid system based on the OWASP Juice Shop benchmark application, which, using model data, demonstrates improved threat detection coverage compared to isolated approaches within the developed laboratory testbed.

Keywords: CYBERSECURITY, WEB APPLICATION, DEVSECOPS, SAST, DAST, OWASP ZAP, GITHUB CODEQL, CI/CD, VULNERABILITY AGGREGATION.

		Пятаков О.О.			ККИБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1 АНАЛІЗ СТАНУ БЕЗПЕКИ ВЕБ-ДОДАТКІВ ТА МЕТОДІВ ЇЇ ОЦІНЮВАННЯ	9
1.1 Актуальність проблеми безпеки веб-додатків та класифікація вразливостей за OWASP Топ 10	9
1.2 Огляд методів статичного аналізу захищеності (SAST) та їх особливості	14
1.3 Огляд методів динамічного аналізу захищеності (DAST) та їх особливості	17
1.4 Порівняльний аналіз існуючих гібридних підходів SAST+DAST до тестування безпеки. 21	
1.5 Огляд інструментів OWASP ZAP та GitHub CodeQL як представників класів DAST і SAST відповідно.....	23
РОЗДІЛ 2 МОДЕЛЬ ГІБРИДНОЇ СИСТЕМИ SAST+DAST ДЛЯ АНАЛІЗУ БЕЗПЕКИ ВЕБ- ДОДАТКІВ	29
2.1 Концептуальна модель гібридної системи аналізу захищеності веб-додатків	29
2.2 Архітектура інтеграції OWASP ZAP та GitHub CodeQL у єдиний аналітичний конвеєр... 32	
2.3 Формальна модель процесу виявлення вразливостей та агрегації результатів	35
2.4 Модель оцінювання ефективності гібридної системи та метрики якості аналізу	39
РОЗДІЛ 3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ГІБРИДНОЇ СИСТЕМИ.....	44
3.1 Розробка стенду для проведення експериментів та вибір тестових веб-додатків	44
3.2 Налаштування та конфігурування OWASP ZAP для динамічного сканування.....	46
3.3 Налаштування та конфігурування GitHub CodeQL для статичного аналізу коду	49
3.4 Проведення порівняльних експериментів: ізольований SAST, ізольований DAST та гібридний підхід	51
3.5 Аналіз результатів: показники повноти виявлення, хибно-позитивних спрацювань та покриття вразливостей.....	54
ВИСНОВКИ	59
ДОДАТКИ	63

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС - операційна система

ПЗ - програмне забезпечення

API - Application Programming Interface (інтерфейс прикладного програмування)

CI/CD - Continuous Integration / Continuous Deployment (безперервна інтеграція та безперервне розгортання)

CORS - Cross-Origin Resource Sharing (спільне використання ресурсів з різних джерел)

CSP - Content Security Policy (політика захисту контенту)

DAST - Dynamic Application Security Testing (динамічне тестування безпеки додатків)

DOM - Document Object Model (об'єктна модель документа)

HTTP - HyperText Transfer Protocol (протокол передавання гіпертексту)

IDE - Integrated Development Environment (інтегроване середовище розробки)

OWASP - Open Worldwide Application Security Project (відкритий міжнародний проєкт із безпеки веб-додатків)

SAST - Static Application Security Testing (статичне тестування безпеки додатків)

SDLC - Software Development Life Cycle (життєвий цикл розробки програмного забезпечення)

SPA - Single Page Application (односторінковий веб-додаток)

SQL - Structured Query Language (мова структурованих запитів)

URL - Uniform Resource Locator (уніфікований покажчик ресурсу)

XSS - Cross-Site Scripting (міжсайтовий скриптинг)

ZAP - Zed Attack Proxy (динамічний сканер вразливостей від OWASP)

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Перехід до DevSecOps та концепції «Shift-Left» зумовлений необхідністю інтеграції перевірок безпеки на ранніх етапах розробки програмного забезпечення. Методи тестування, такі як SAST та DAST, при ізольованому використанні мають обмеження: SAST не враховує конфігурацію середовища, а DAST не має доступу до логіки коду. Тема є важливою, оскільки гібридна модель дозволяє компенсувати недоліки підходів, підвищуючи виявлення вразливостей згідно з OWASP Top 10 та мінімізуючи ризики витоку даних.

Метою кваліфікаційної роботи є проектування лабораторного стенду та розробка методики дослідження захищеності веб-додатків на основі гібридного поєднання статичного й динамічного аналізу (SAST+DAST).

Визначена мета обумовлює наступні завдання:

- проаналізувати загрози веб-додатків та методи оцінювання ;
- розробити архітектуру та модель гібридного аналізу на базі GitHub CodeQL та OWASP ZAP;
- спроектувати лабораторний стенд та демонстраційний конвеєр аналізу безпеки на базі GitHub Actions;
- розробити програмний модуль агрегації результатів та апробувати методику на модельному наборі даних.

Об'єктом дослідження є процес забезпечення безпеки веб-додатків на етапах життєвого циклу розробки програмного забезпечення.

Предметом дослідження є методи, моделі та програмно-апаратні засоби побудови лабораторного стенду для дослідження захищеності веб-додатків.

У процесі виконання кваліфікаційної роботи було розроблено проект DevSecOps-конвеєра у GitHub Actions, який інтегрує CodeQL та ZAP аналіз, а також включає Python-скрипт для автоматичної генерації PDF-звіту.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1 АНАЛІЗ СТАНУ БЕЗПЕКИ ВЕБ-ДОДАТКІВ ТА МЕТОДІВ ЇЇ ОЦІНЮВАННЯ

1.1 Актуальність проблеми безпеки веб-додатків та класифікація вразливостей за OWASP Top 10

В умовах стрімкої цифрової трансформації веб-додатки стали ключовим елементом ІТ-інфраструктури більшості державних та комерційних організацій, забезпечуючи безперебійний доступ користувачів до сервісів та даних.

Проте саме ця глобальна доступність робить їх головною мішенню для кібератак. Веб-додатки функціонують як своєрідні відкриті двері у внутрішню мережу організації, оскільки вони безпосередньо взаємодіють із критичними базами даних, платіжними шлюзами та корпоративними API.

Актуальність проблеми захисту веб-орієнтованих систем підтверджується глобальною статистикою кіберінцидентів за останні роки. Згідно з аналітичним звітом IBM Cost of a Data Breach Report за 2023–2024 роки, середня глобальна вартість витоку даних досягла історичного максимуму і становить понад 4,45 мільйона доларів США за один інцидент, причому значна частка цих витрат пов'язана саме з компрометацією веб-додатків.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

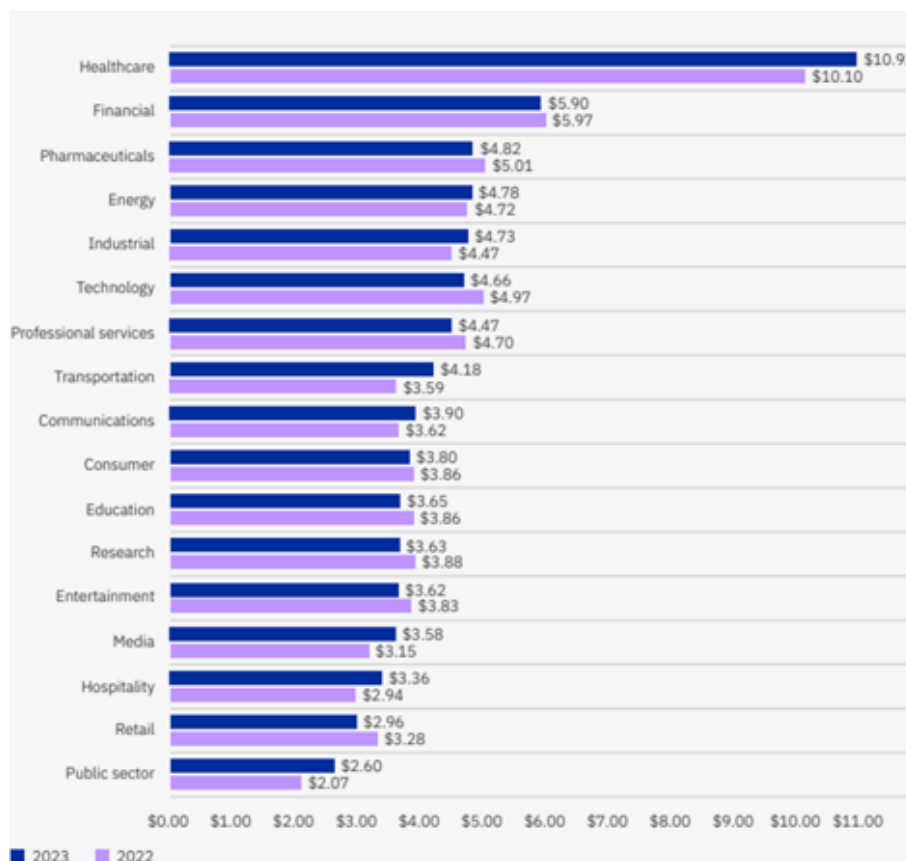


Рисунок 1.1 — Вартість витоку даних за галузями

Водночас щорічний звіт компанії Verizon щодо розслідування витоків даних (Data Breach Investigations Report, DBIR) стабільно вказує на те, що атаки на базові веб-додатки (Basic Web Application Attacks) є домінуючим патерном дій зловмисників, відповідаючи за понад шістдесят відсотків усіх успішних зломів. Зловмисники експлуатують слабкі місця в логіці та архітектурі, щоб отримати несанкціонований доступ до конфіденційної інформації.

Для систематизації підходів до захисту та ідентифікації найбільш критичних векторів атак, міжнародна некомерційна організація Open Web Application Security Project (OWASP) кожні кілька років публікує фундаментальний документ OWASP Top 10. Актуальна редакція цього документа за 2021 рік містить десять найнебезпечніших категорій вразливостей, які вимагають обов'язкового врахування під час розробки та тестування програмного забезпечення.

Першою та найпоширенішою категорією є порушення контролю доступу. Ця вразливість виникає, коли система не перевіряє належним чином, чи має

автентифікований користувач права на виконання певної дії або доступ до конкретного ресурсу.

Типовим прикладом такої атаки є небезпечне пряме посилання на об'єкт (IDOR), коли зловмисник змінює ідентифікатор користувача в URL-адресі або параметрах запиту, отримуючи доступ до чужого облікового запису. Наслідком успішної експлуатації є несанкціоноване розкриття, модифікація або знищення конфіденційних даних клієнтів.

Другою категорією є криптографічні збої, які стосуються проблем із захистом даних під час їх передачі або зберігання. Ця проблема проявляється у використанні застарілих алгоритмів хешування, відсутності шифрування конфіденційної інформації в базах даних або використанні незахищеного протоколу HTTP замість HTTPS.

Прикладом атаки є перехоплення мережевого трафіку зловмисником з метою викрадення паролів або номерів кредитних карток, які передавалися у відкритому вигляді. Наслідком є масовий витік персональних даних та жорсткі штрафні санкції з боку регуляторів.

Третьою категорією є ін'єкції. Цей клас вразливостей виникає, коли недовірені дані, отримані від користувача, відправляються інтерпретатору як частина команди або запиту без попередньої санітизації.

Класичним прикладом є SQL-ін'єкція, під час якої зловмисник вводить у поле форми авторизації спеціальні символи та SQL-оператори, змінюючи логіку запиту до бази даних. Успішна ін'єкція призводить до повного компрометування бази даних, можливості обходу систем авторизації та, у деяких випадках, до віддаленого виконання коду на сервері.

Четвертою категорією визначено небезпечний дизайн. Ця категорія фокусується на недоліках архітектури, які виникають ще на етапі проектування системи, і які неможливо виправити простою зміною кількох рядків коду.

Прикладом атаки є використання механізму відновлення пароля без обмеження кількості спроб (Rate Limiting), що дозволяє зловмиснику здійснити масовий підбір

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

відповідей на секретні питання. Наслідком є компрометація бізнес-логіки додатка та масове захоплення облікових записів.

П'ята категорія стосується небезпечного налаштування систем та серверів. Вона виникає через використання облікових даних за замовчуванням, відкриті хмарні сховища або надмірну деталізацію повідомлень про помилки.

Наприклад, якщо веб-сервер повертає користувачеві повний стек викликів (Stack Trace) у разі виникнення виключення, зловмисник може дізнатися точні версії використовуваних компонентів та внутрішні шляхи до файлів. Наслідком є полегшення розвідки для зловмисника та підготовка ґрунту для більш складних цілеспрямованих атак.

Шостою категорією є використання вразливих та застарілих компонентів. Сучасна розробка значною мірою спирається на сторонні бібліотеки та фреймворки. Якщо розробники не оновлюють ці компоненти, додаток стає вразливим до вже відомих та задокументованих атак.

Яскравим прикладом є експлуатація відомої вразливості у популярній бібліотеці логування, яка дозволяє зловмиснику відправити спеціально сформований рядок і виконати свій код на сервері. Наслідком є повне захоплення інфраструктури без необхідності пошуку нових вразливостей у самому коді додатка.

Сьома категорія описує помилки ідентифікації та автентифікації. Це проблеми, пов'язані з некоректним управлінням сесіями користувачів та слабкими вимогами до паролів.

Прикладом атаки є фіксація сесії, коли зловмисник змушує жертву використовувати заздалегідь відомий ідентифікатор сесії, або використання автоматизованих інструментів для підбору вкрадених пар логін-пароль (Credential Stuffing). Наслідком є несанкціонований доступ до системи від імені легітимного користувача.

Восьмою категорією є порушення цілісності програмного забезпечення та даних. Ця загроза виникає, коли додаток завантажує оновлення, плагіни або дані з недовірених джерел без перевірки цифрових підписів.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

Прикладом є атака на ланцюжок постачання (Supply Chain Attack), коли зловмисник зламує репозиторій стороннього компонента, який автоматично інтегрується в систему під час збірки. Наслідком є впровадження шкідливого коду або бекдорів безпосередньо у фінальний продукт.

Дев'ята категорія стосується збоїв у логуванні та моніторингу безпеки. Відсутність ефективного аудиту подій не дозволяє вчасно виявити кібератаку.

Наприклад, якщо система не реєструє численні невдалі спроби авторизації з однієї IP-адреси і не генерує сповіщень для адміністратора, зловмисник може тижнями проводити розвідку та підбирати паролі непоміченим. Наслідком є те, що середній час виявлення проникнення може складати місяці, що збільшує масштаби викрадених даних.

Десятою категорією визначено підробку запитів на стороні сервера (SSRF). Вона виникає, коли додаток приймає від користувача URL-адресу та здійснює за нею HTTP-запит без перевірки цільового пункту призначення.

Прикладом атаки є передача зловмисником локальної IP-адреси, що змушує веб-сервер звертатися до внутрішніх ресурсів, які недоступні з інтернету. Наслідком є обхід міжмережевих екранів, отримання доступу до панелей адміністрування та викрадення хмарних метаданих.

Аналізуючи всю складність та різноманітність наведених категорій вразливостей, можна зробити висновок, що забезпечення безпеки веб-додатків є комплексним та безперервним процесом.

Враховуючи обсяги вихідного коду сучасних систем та швидкість їх оновлення, виявлення перелічених загроз за допомогою виключно ручного аудиту або періодичних тестувань на проникнення є неефективним та надзвичайно ресурсоємним.

Людський фактор неминуче призводить до пропуску критичних дефектів, особливо у складних графах виконання та конфігураціях інфраструктури.

Саме тому для забезпечення надійного захисту виникає гостра необхідність у впровадженні автоматизованих засобів виявлення вразливостей безпосередньо у процес розробки програмного забезпечення, що обумовлює необхідність

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

детального дослідження методів статичного та динамічного аналізу в наступних підрозділах роботи.

1.2 Огляд методів статичного аналізу захищеності (SAST) та їх особливості

Статичне тестування безпеки додатків (Static Application Security Testing, SAST) являє собою фундаментальний аналітичний підхід до виявлення вразливостей, який суворо базується на парадигмі тестування за принципом «білого ящика» (white-box testing).

Цей метод передбачає проведення глибокого алгоритмічного аналізу вихідного коду, байт-коду або бінарних файлів програми без її фактичного виконання чи запуску в робочому середовищі.

Головною метою методології SAST є превентивне виявлення синтаксичних, семантичних та логічних дефектів безпеки на найраніших етапах життєвого циклу розробки програмного забезпечення.

Процес функціонування сучасних засобів статичного аналізу є складним багатоетапним конвеєром, який використовує передові математичні та лінгвістичні моделі обробки даних.

На першому етапі рушій інструмента зчитує сирий вихідний код та виконує його лексичний аналіз (парсинг), розбиваючи текст на базові токени.

Наступним критичним кроком є синтаксичний аналіз, під час якого система будує абстрактне синтаксичне дерево (Abstract Syntax Tree, AST).

Ця ієрархічна структура даних математично точно відображає граматику мови програмування та алгоритмічну логіку розробленої програми.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

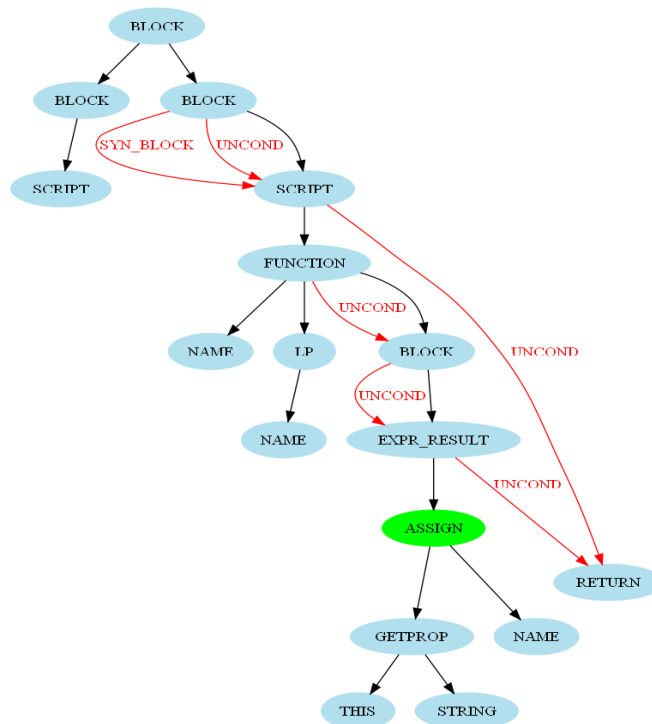


Рисунок 1.2 — Приклад побудови абстрактного синтаксичного дерева (AST) та графа потоку керування під час статичного аналізу коду

Завершальним і найбільш ресурсоємним етапом є семантичний аналіз, під час якого будуються графи потоку керування (Control Flow) та потоку даних (Data Flow).

Ядром цього етапу є так званий аналіз забруднення (Taint Analysis).

Його алгоритм простежує шлях потенційно небезпечних, неперевіраних вхідних даних від точки їх потрапляння в систему (наприклад, з веб-форми або API-запиту) до критичної точки виконання (наприклад, функції запиту до бази даних). Якщо несанітизовані дані досягають такої функції, система фіксує наявність вразливості.

На сучасному ринку інформаційної безпеки інструменти SAST класифікуються за кількома архітектурними ознаками.

За типом проведення аналізу вони поділяються на базові лексичні сканери, що здійснюють пошук вразливостей за допомогою регулярних виразів, та просунуті семантичні рушії, здатні відстежувати складні міжпроцесні потоки даних.

За мовною підтримкою інструменти класифікуються на мономовні рішення, які глибоко інтегровані в екосистему лише однієї технології, та полімовні платформи,

здатні одночасно аналізувати гетерогенні корпоративні проекти, написані різними мовами програмування.

Впровадження методів статичного аналізу надає організаціям стратегічні переваги, головною з яких є практична реалізація концепції Shift-Left.

Зміщення процесів перевірки безпеки вліво, тобто ближче до моменту написання коду розробником, має колосальний економічний ефект, оскільки вартість виправлення дефекту на етапі написання коду є мінімальною.

Ще однією унікальною перевагою SAST є здатність забезпечувати теоретично повне покриття вихідного коду.

Статичний аналізатор здатний дослідити всі можливі гілки виконання алгоритму, включаючи рідкісні обробники виключень або приховані адміністративні функції.

Крім того, сканери надають розробникам точну просторову локалізацію проблеми, вказуючи ім'я файлу та конкретний рядок уразливого коду, що суттєво мінімізує час, необхідний для усунення дефекту.

Водночас методологія SAST має об'єктивні технологічні обмеження. Найбільшою проблемою є високий рівень хибно-позитивних спрацювань. Оскільки інструмент аналізує код ізольовано і не має доступу до середовища виконання, він може класифікувати як загрозу ту ділянку алгоритму, яка на практиці надійно захищена правилами зовнішнього міжмережевого екрана або архітектурою бази даних.

Іншим суттєвим недоліком є архітектурна сліпота до інфраструктурних загроз: SAST принципово не здатний виявити слабкі системні паролі, помилки у конфігурації веб-сервера або вразливості у робочому середовищі.

Також значним обмеженням є жорстка прив'язка до мови програмування та, у багатьох випадках, необхідність повної компіляції проекту перед початком семантичного сканування.

Екосистема інструментів SAST представлена багатьма потужними рішеннями. Платформа SonarQube здобула популярність завдяки вдалому поєднанню функцій

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

пошуку базових вразливостей з комплексним контролем якості коду та технічного боргу.

Інструмент Semgrep виділяється високою швидкістю роботи та унікальним підходом до написання правил перевірки, які синтаксично нагадують сам вихідний код, що полегшує їх кастомізацію.

Корпоративне рішення Checkmarx позиціонується як інструмент преміум-класу, здатний будувати надскладні графи потоків даних для виявлення неочевидних вразливостей у великих системах.

Особливої уваги заслуговує інструмент GitHub CodeQL, який пропонує інноваційну концепцію сприйняття вихідного коду як реляційної бази даних. Використання спеціалізованої мови QL дозволяє виконувати високоточні семантичні запити, що робить його одним із найперспективніших рішень для побудови інтегрованих конвеєрів безпеки.

1.3 Огляд методів динамічного аналізу захищеності (DAST) та їх особливості

Динамічне тестування безпеки додатків (Dynamic Application Security Testing, DAST) являє собою методологію автоматизованого пошуку вразливостей, яка концептуально протилежна статичному аналізу і базується на парадигмі тестування за принципом «чорного ящика» (black-box testing).

Замість дослідження внутрішньої структури вихідного коду, DAST-сканери взаємодіють із веб-додатком ззовні, імітуючи поведінку реального зловмисника. Цей підхід вимагає наявності повністю розгорнутого, скомпільованого та функціонуючого екземпляра програми (у тестовому або підготовчому середовищі), оскільки головною метою динамічного аналізу є оцінка реакції системи на шкідливі вхідні дані під час її фактичного виконання (runtime).

Архітектура взаємодії інструментів динамічного аналізу з цільовим веб-додатком зазвичай будується за моделлю перехоплюючого HTTP-проксі сервера (Man-in-the-Middle). Сканер вбудовується між клієнтом (браузером або автоматизованим павуком) та веб-сервером, що дозволяє йому фіксувати,

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

аналізувати та модифікувати всі HTTP/HTTPS запити та відповіді. Процес сканування традиційно поділяється на два базові режими: пасивний та активний.

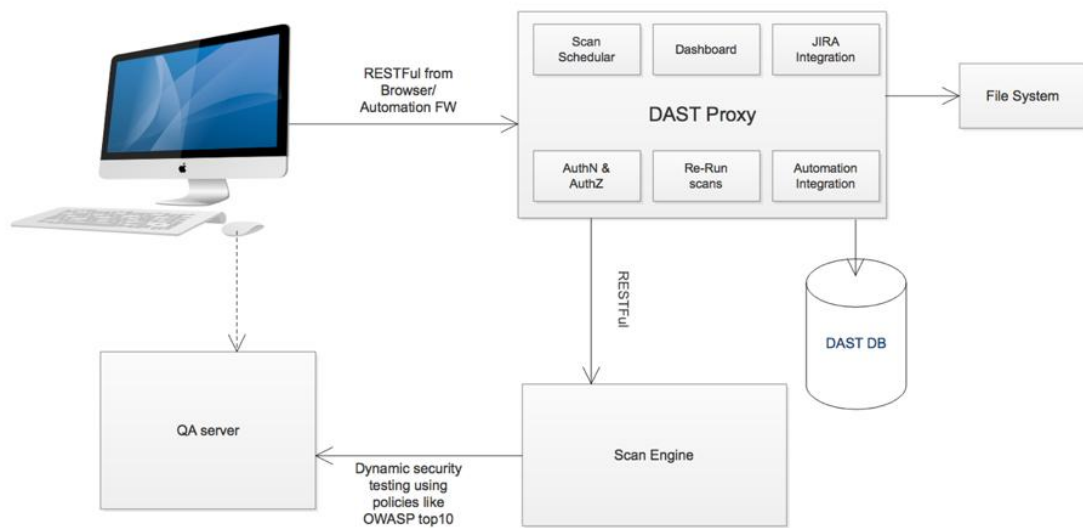


Рисунок 1.3 — Архітектура взаємодії динамічного сканера безпеки (DAST) із цільовим веб-додатком за моделлю перехоплюючого проксі.

Під час пасивного сканування інструмент лише спостерігає за мережевим трафіком, не змінюючи його вміст. На цьому етапі виявляються проблеми конфігурації, такі як відсутність безпекових заголовків (наприклад, Content Security Policy), передача конфіденційних даних у відкритому вигляді або некоректне налаштування атрибутів файлів cookie.

Після завершення етапу розвідки ініціюється активне сканування, під час якого сканер починає цілеспрямовано модифікувати параметри запитів. Базовим механізмом активної фази є фазинг (fuzzing) - процес автоматизованої генерації та відправлення великих обсягів некоректних, неочікуваних або випадкових даних (корисних навантажень) до всіх виявлених точок входу додатка з метою провокування критичних помилок, збоїв логіки або обходу механізмів авторизації.

Впровадження методів динамічного аналізу надає низку унікальних переваг, які неможливо отримати за допомогою статичного тестування.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

Найважливішою з них є здатність DAST виявляти проблеми, які виникають виключно під час виконання програми (runtime-вразливості), такі як помилки управління сесіями, недоліки механізмів автентифікації та обходи бізнес-логіки.

Крім того, оскільки динамічний сканер аналізує систему в цілому, він ефективно виявляє інфраструктурні вразливості: помилки конфігурації веб-серверів (Apache, Nginx), використання застарілих версій SSL/TLS протоколів та відкриті директорії.

Ще однією стратегічною перевагою DAST є абсолютна незалежність від мови програмування. Для динамічного сканера не має значення, чи написаний бекенд на Java, Python, Go або Node.js; він взаємодіє виключно зі стандартизованими мережевими протоколами, що робить його універсальним інструментом для аудиту гетерогенних корпоративних екосистем та закритих комерційних продуктів (Commercial Off-The-Shelf), для яких вихідний код недоступний.

Незважаючи на високу ефективність у виявленні реальних загроз, методологія DAST має низку критичних обмежень.

Оскільки інструмент працює за принципом чорного ящика, він принципово не здатний вказати розробнику точне місцезнаходження вразливості у вихідному коді (наприклад, ім'я файлу чи номер рядка).

Результатом сканування є лише URL-адреса, модифікований параметр та тіло HTTP-відповіді сервера, що суттєво збільшує час, необхідний інженерам для пошуку та усунення дефекту.

Іншим вагомим недоліком є проблема неповного покриття. Динамічний аналізатор може протестувати лише ті сторінки та маршрути API, до яких він зміг автоматично дістатися під час фази сканування (crawling).

Якщо функціонал схований за складною багатоекранною формою або вимагає специфічного стану бізнес-логіки, сканер просто його пропустить. Крім того, активне сканування становить серйозний ризик пошкодження даних: агресивний фазинг може призвести до заповнення баз даних «сміттєвими» записами, масової розсилки автоматичних електронних листів користувачам або навіть до виведення системи з ладу через вичерпання ресурсів (відмову в обслуговуванні).

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

На ринку програмного забезпечення існує широкий спектр інструментів для проведення динамічного аналізу, кожен з яких має свою специфіку застосування. Інструмент OWASP Zed Attack Proxy (ZAP) є найвідомішим рішенням з відкритим вихідним кодом, який пропонує потужний проксі-сервер, сучасні алгоритми обходу односторінкових додатків (AJAX Spider) та розширений REST API, що робить його ідеальним кандидатом для автоматизації в CI/CD конвеєрах.

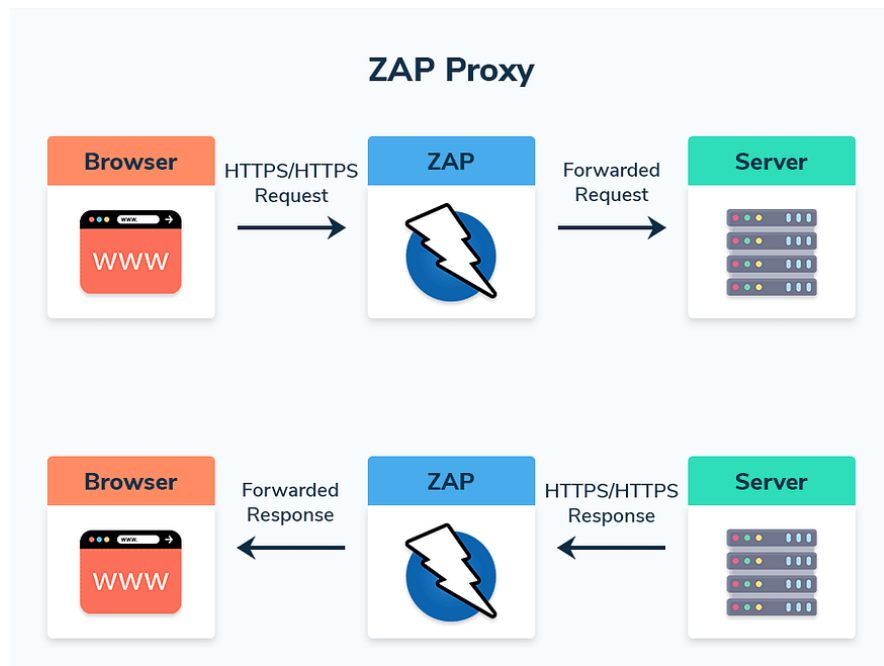


Рисунок 1.4 — Процес автоматизованого динамічного сканування веб-додатка за допомогою інструменту OWASP ZAP.

Комерційна платформа Burp Suite від компанії PortSwigger де-факто є індустріальним стандартом для проведення ручного тестування на проникнення; її модуль Intruder надає неперевершені можливості для гнучкого фазингу та експлуатації складних логічних вразливостей.

Рішення Acunetix позиціонується як високопродуктивний корпоративний сканер, алгоритми якого оптимізовані для глибокого сканування сучасних JavaScript-фреймворків, що дозволяє досягати низького рівня хибно-позитивних спрацювань, проте вартість ліцензії обмежує його доступність.

У свою чергу, відкритий консольний інструмент Nikto фокусується на швидкому пошуку застарілого серверного програмного забезпечення, небезпечних

файлів конфігурації та типових помилок веб-серверів, виступаючи ефективним доповненням для базового аудиту інфраструктури, хоча і генерує значну кількість інформаційного шуму.

1.4 Порівняльний аналіз існуючих гібридних підходів SAST+DAST до тестування безпеки

Гібридне тестування безпеки додатків (Hybrid Application Security Testing, HAST) виникло як еволюційна відповідь на об'єктивні недоліки ізольованого використання методів статичного та динамічного аналізу.

Головною метою гібридизації є створення синергійного ефекту, при якому результати одного методу доповнюють або верифікують результати іншого, що дозволяє радикально знизити кількість хибно-позитивних спрацювань та підвищити загальну повноту виявлення загроз.

У сучасній практиці побудови захищених конвеєрів розробки виділяють кілька стратегій інтеграції.

Найпростішою формою є послідовна інтеграція, при якій сканери SAST і DAST запускаються один за одним на різних етапах життєвого циклу розробки, а їх звіти розглядаються фахівцями як незалежні артефакти.

Більш складним підходом є паралельне сканування, яке передбачає одночасне виконання перевірок у хмарній інфраструктурі, що дозволяє скоротити загальний час зворотного зв'язку, проте все ще вимагає значних зусиль для ручної агрегації даних на фінальному етапі.

Найбільш перспективним методом у контексті повної автоматизації є кореляційний аналіз. Цей підхід передбачає використання спеціалізованого програмного забезпечення для автоматичного зіставлення вразливостей, знайдених різними інструментами, за спільними ознаками: ідентифікаторами CWE, шляхами до файлів, назвами параметрів HTTP-запитів або сигнатурами атак.

Окремим еволюційним відгалуженням гібридних підходів є інтерактивне тестування безпеки (Interactive Application Security Testing, IAST).

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

Технологія IAST поєднує переваги обох світів шляхом вбудовування спеціального агента безпосередньо у середовище виконання додатка. Агент аналізує виконання коду зсередини під час проведення зовнішніх запитів, що дозволяє точно визначати місце вразливості у вихідному коді без високого рівня хибних спрацювань.

Проте складність розгортання та залежність від конкретних платформ виконання обмежують масове впровадження IAST порівняно з кореляційними методами SAST+DAST.

На сучасному ринку представлено кілька ключових рішень, що реалізують ці підходи. Платформа OWASP DefectDojo є провідним відкритим агрегатором, який дозволяє зводити звіти від понад сотні різних сканерів у єдиний інтерфейс, забезпечуючи дедуплікацію на основі метаданих.

Корпоративне рішення Fortify Software Security Center (SSC) пропонує глибоку інтеграцію між статичним аналізатором Fortify SAST та динамічним WebInspect, використовуючи пропріетарні алгоритми для автоматичної верифікації вразливостей.

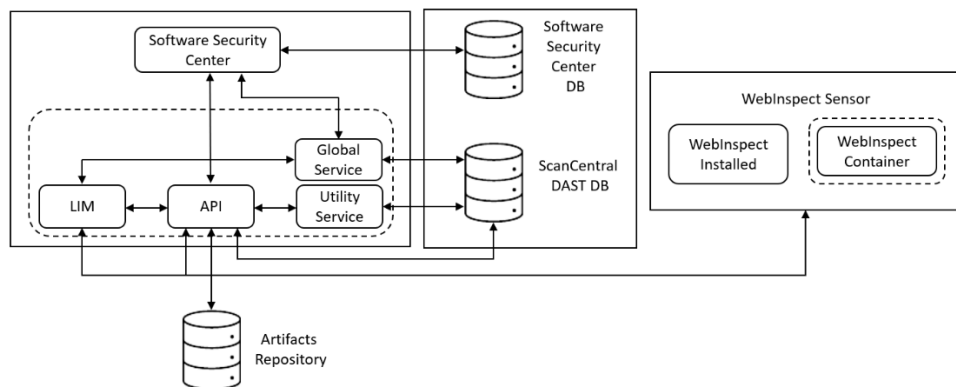


Рисунок 1.5 — Архітектура взаємодії компонентів гібридної корпоративної системи аналізу захищеності (на прикладі Fortify SSC)

GitLab Security Dashboard надає розробникам нативний інтерфейс для візуалізації результатів сканування безпосередньо у репозиторії, проте його повний функціонал для управління безпекою доступний лише у преміальних версіях платформи.

Для систематизації відмінностей між існуючими підходами та обґрунтування вибору технологічного стеку у таблиці наведено їх порівняльну характеристику.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 1.1

Порівняльна характеристика гібридних підходів та систем безпеки

Критерій порівняння	Послідовна інтеграція	Кореляційний аналіз (напр. DefectDojo)	Корпоративні платформи (напр. Fortify)	IAST підхід
Складність впровадження	Низька	Середня	Висока	Дуже висока
Точність кореляції	Відсутня (ручна)	Середня	Висока	Дуже висока
Ціна рішення	Низька / Open-source	Низька (Community)	Дуже висока	Висока
Відкритість коду	Залежить від сканера	Так	Ні	Переважно ні

Аналіз існуючих рішень демонструє значну прогалину у сегменті доступних систем. Комерційні платформи, що забезпечують високу точність кореляції, мають надзвичайно високу вартість ліцензування та складну архітектуру, що робить їх недоступними для малих команд.

Натомість безкоштовні агрегатори часто виконують роль лише пасивних сховищ даних без розвиненої логіки активної кореляції у реальному часі. Це обґрунтовує актуальність даної роботи, що спрямована на розробку легкої, автоматизованої гібридної системи на базі GitHub CodeQL та OWASP ZAP, яка забезпечить високу точність аналізу при збереженні повної відкритості та доступності інструментарію.

1.5 Огляд інструментів OWASP ZAP та GitHub CodeQL як представників класів DAST і SAST відповідно

Для практичної реалізації розроблюваної гібридної системи тестування безпеки було проведено аналіз доступних на ринку рішень, за результатами якого для етапу статичного аналізу було обрано семантичний рушій GitHub CodeQL.

Історія цього інструмента бере свій початок у дослідницькій компанії Semmle, яка розробила унікальний підхід до пошуку вразливостей, а згодом була поглинута корпорацією GitHub для інтеграції у свою глобальну екосистему.

Фундаментальною відмінністю CodeQL від традиційних статичних аналізаторів є його інноваційна концепція сприйняття вихідного коду як реляційної бази даних.

Під час процесу ініціалізації інструмент не просто шукає збіги за регулярними виразами, а компілює вихідний код, витягуючи з нього абстрактне синтаксичне дерево, граfi потоків даних та інформацію про виклики функцій, зберігаючи ці дані у високоструктурованому форматі.

Після побудови такої бази даних рушій використовує спеціалізовану об'єктно-орієнтовану мову логічних запитів QL (Query Language). Ця мова дозволяє інженерам з безпеки писати запити до коду так само, як до звичайної бази даних SQL, шукаючи специфічні патерни виконання, що призводять до вразливостей.

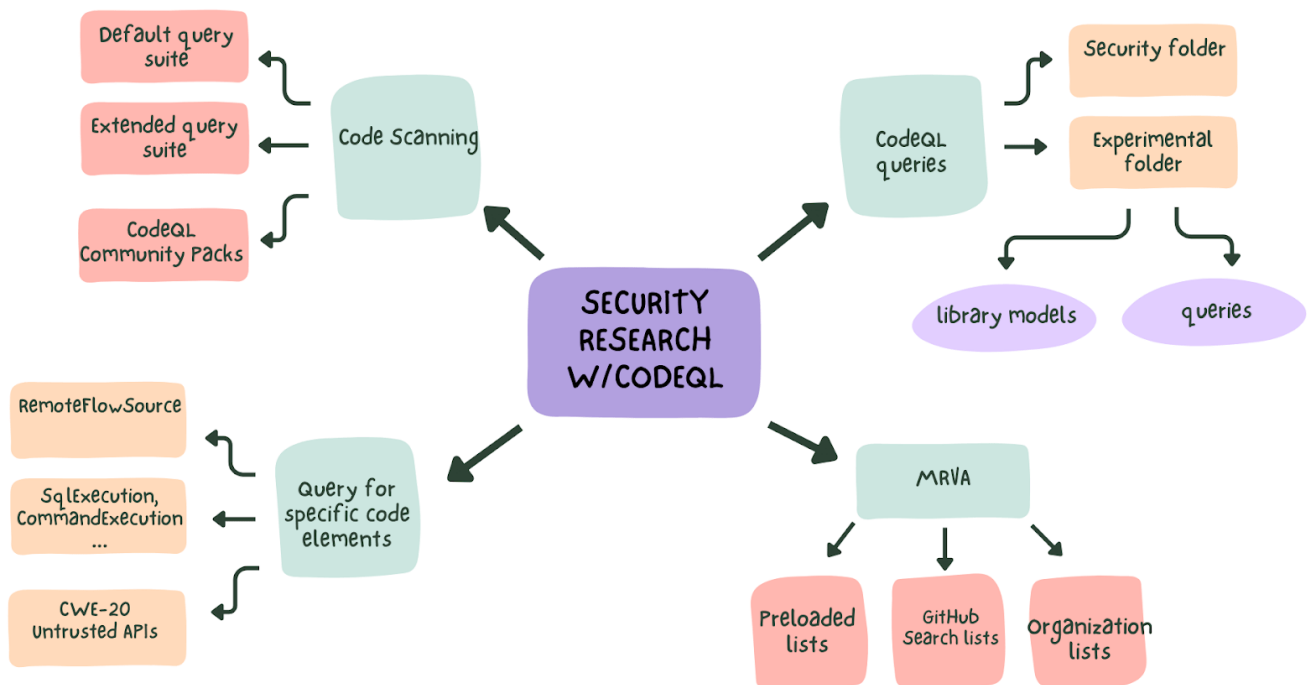


Рисунок 1.6 — Процес перетворення вихідного коду у реляційну базу даних та виконання QL-запитів під час аналізу CodeQL

Важливою архітектурною перевагою CodeQL є його повна стандартизація: результати сканування автоматично експортуються у міжнародний машиночитний формат SARIF (Static Analysis Results Interchange Format).

Використання формату SARIF забезпечує безшовну інтеграцію інструмента з хмарною платформою GitHub Actions, де результати сканування автоматично парсяться системою та відображаються розробникам у вигляді зручних попереджень безпосередньо у запитах на злиття коду (Pull Requests).

Для забезпечення етапу динамічного аналізу в розроблюваній гібридній моделі було обрано сканер OWASP Zed Attack Proxy (ZAP).

Цей інструмент є одним із найпопулярніших у світі відкритих проєктів для тестування безпеки веб-додатків, який підтримується глобальною спільнотою експертів OWASP.

Архітектура ZAP базується на моделі перехоплюючого проксі-сервера (Man-in-the-Middle). Розташовуючись між клієнтом та цільовим сервером, інструмент отримує повний контроль над HTTP та HTTPS трафіком, що дозволяє йому не лише фіксувати дані, але й модифікувати їх у реальному часі.

Процес роботи OWASP ZAP поділяється на кілька ключових фаз. Спочатку інструмент використовує традиційного павука для обходу статичних посилань, після чого підключає алгоритм AJAX Spider.

Цей алгоритм запускає прихований (headless) браузер для виконання клієнтського JavaScript-коду, що є критично важливим для коректного сканування сучасних односторінкових додатків (Single Page Applications), побудованих на фреймворках Angular або React.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

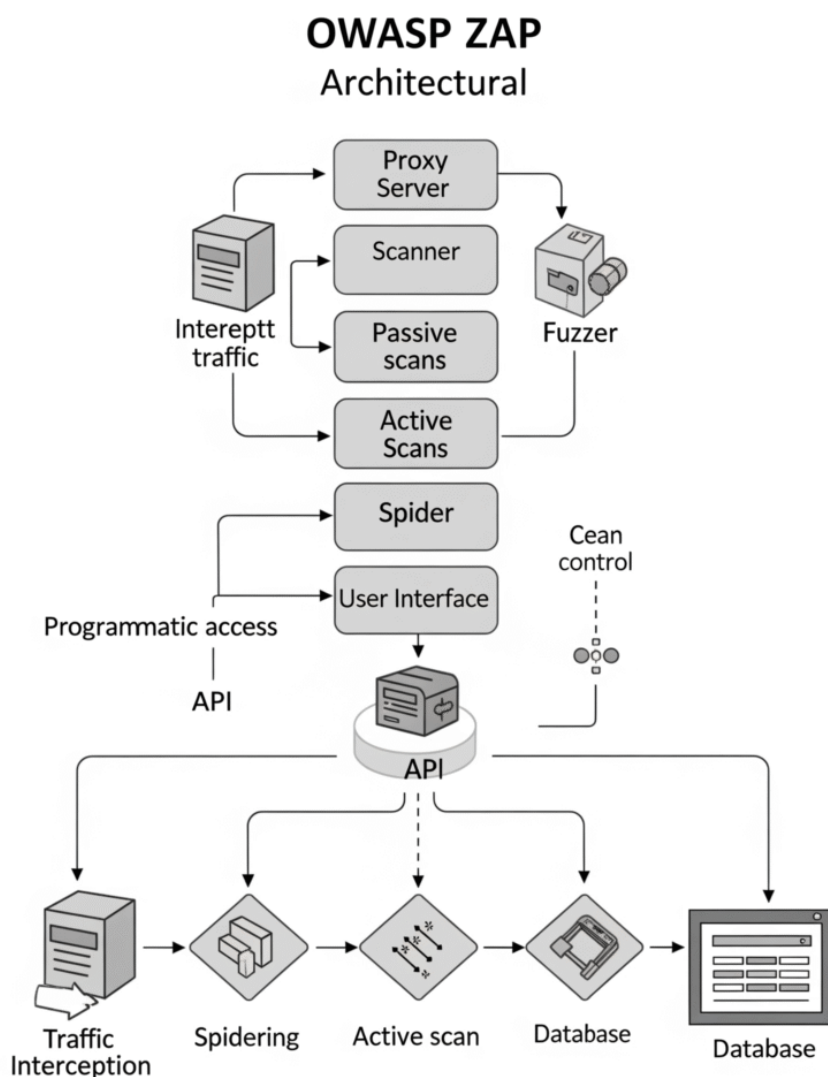


Рисунок 1.7 — Архітектурна модель динамічного сканування веб-додатка за допомогою OWASP ZAP

Після побудови карти сайту ZAP виконує пасивне сканування, під час якого аналізує зібраний трафік на наявність відсутніх безпекових заголовків або витоків інформації без втручання в роботу сервера. Завершальним етапом є активне сканування, під час якого інструмент генерує тисячі шкідливих запитів (фазинг) для виявлення ін'єкцій, міжсайтового скриптингу та інших критичних загроз.

Вибір саме цієї комбінації інструментів (CodeQL та OWASP ZAP) для побудови авторського гібридного конвеєра є глибоко обґрунтованим з наукової та практичної точок зору.

По-перше, обидва рішення надаються на умовах відкритого вихідного коду або безкоштовного використання для відкритих проєктів, що усуває фінансові бар'єри та робить запропоновану гібридну архітектуру доступною для масового впровадження.

По-друге, обидва інструменти володіють вичерпною офіційною документацією та підтримуються потужними спільнотами, що гарантує регулярне оновлення баз сигнатур вразливостей.

Вирішальним фактором вибору стала їхня ідеальна пристосованість до автоматизації у середовищах неперервної інтеграції (CI/CD). Наявність офіційного Docker-образу для OWASP ZAP та потужного REST API дозволяє запускати динамічне сканування повністю в автономному (безголовому) режимі за допомогою простих скриптів.

Одночасно з цим, нативна інтеграція CodeQL у GitHub Actions дозволяє паралельно виконувати статичний аналіз.

Саме здатність цих інструментів генерувати структуровані машиночитні звіти (JSON та SARIF) без втручання людини створює ідеальний технологічний фундамент для розробки програмного модуля агрегації та кореляції, який буде детально описаний у наступних розділах даної кваліфікаційної роботи.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи було проведено комплексний аналіз сучасного стану безпеки веб-додатків та ключових методів її автоматизованого оцінювання. На основі дослідження актуальної статистики кіберінцидентів та міжнародної методології OWASP Top 10 доведено, що веб-орієнтовані системи залишаються основною мішенню для зловмисників. Зростання архітектурної складності програмного забезпечення робить традиційні ручні перевірки безпеки неефективними, надто повільними та економічно не вигідними, що зумовлює критичну необхідність впровадження автоматизованих засобів контролю безпосередньо у життєвий цикл розробки.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

Детальний огляд існуючих технологій виявив, що ізольоване використання методів тестування не здатне забезпечити надійний захист інформаційних систем. Статичний аналіз (SAST) демонструє високу ефективність у превентивному виявленні алгоритмічних недоліків та забезпечує повне покриття вихідного коду, проте його суттєвим недоліком є генерація значної кількості хибно-позитивних спрацювань через ізольованість від реального середовища виконання. З іншого боку, динамічне сканування (DAST) дозволяє ефективно виявляти вразливості інфраструктури та проблеми бізнес-логіки на працюючому додатку, але принципово не здатне вказати інженерам точне місцезнаходження дефекту у вихідному коді, що суттєво уповільнює процес його усунення.

Порівняльний аналіз гібридних підходів підтвердив, що єдиним шляхом до побудови надійної ешелонованої оборони є синергійне поєднання статичного та динамічного аналізу з подальшою кореляцією їхніх результатів. Однак дослідження сучасного ринку програмного забезпечення показало наявність суттєвої прогалини: інтелектуальне зіставлення вразливостей наразі якісно реалізовано переважно у надзвичайно дорогих пропріетарних корпоративних платформах. Відкриті ж інструменти найчастіше функціонують лише як пасивні агрегатори звітів і перекладають завдання дедуплікації на плечі фахівців з кібербезпеки.

Виявлена проблематика повністю обґрунтовує актуальність даного дослідження. Проведений аналіз дозволив свідомо та аргументовано обрати семантичний рушій GitHub CodeQL та динамічний сканер OWASP ZAP як оптимальні відкриті компоненти для побудови власного конвеєра безпеки. Теоретичні висновки, отримані у цьому розділі, формують надійний методологічний фундамент для переходу до проєктування концептуальної моделі, архітектури конвеєра CI/CD та розробки математичного алгоритму програмної кореляції загроз, що буде реалізовано у наступному розділі роботи.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

РОЗДІЛ 2 МОДЕЛЬ ГІБРИДНОЇ СИСТЕМИ SAST+DAST ДЛЯ АНАЛІЗУ БЕЗПЕКИ ВЕБ-ДОДАТКІВ

2.1 Концептуальна модель гібридної системи аналізу захищеності веб- додатків

Розробка концептуальної моделі гібридної системи аналізу захищеності базується на сучасних парадигмах створення програмного забезпечення, серед яких фундаментальне місце посідає методологія DevSecOps.

На відміну від традиційних водоспадних (Waterfall) або класичних гнучких (Agile) підходів, де перевірка безпеки розглядалася як фінальний, часто ізольований та ресурсоемний етап перед безпосереднім релізом продукту, DevSecOps передбачає глибоку та органічну інтеграцію практик інформаційної безпеки на кожному етапі життєвого циклу розробки (SDLC).

Такий підхід вимагає повної зміни інженерної культури, де відповідальність за безпеку коду перестає бути монополією виділеного відділу кіберзахисту і стає спільною відповідальністю всіх учасників процесу розробки.

Фундаментальним принципом, що закладений в основу запропонованої архітектурної моделі, є концепція Shift-Left (зміщення вліво).

З інженерної точки зору ця концепція полягає у стратегічному перенесенні процедур тестування безпеки на найбільш ранні етапи життєвого циклу проєкту, починаючи з етапу проєктування архітектури та безпосереднього написання перших рядків вихідного коду розробником. Економічна та технічна доцільність підходу Shift-Left підтверджується численними дослідженнями в галузі програмної інженерії, які доводять, що вартість та час на виправлення критичного дефекту експоненціально зростають у міру просування коду від локального середовища розробника до продуктового сервера. Раннє виявлення вразливостей дозволяє трансформувати аудит безпеки з гальмуючого контролюючого бар'єра у неперервний фоновий процес, де ідентифікація та усунення дефектів відбувається синхронно з інкрементальним нарощуванням функціональних можливостей програмної системи.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

Методологічною основою моделі виступає принцип ешелонованості, відомий у теорії інформаційної безпеки як глибокий захист (Defense in Depth). У контексті автоматизованого тестування цей принцип реалізується через побудову багатoshарової системи фільтрації загроз, яка базується на комплементарності різних аналітичних методів. Кожен наступний рівень такої системи проєктується таким чином, щоб компенсувати об'єктивні технологічні обмеження попереднього рівня. Запропонована у даній роботі концептуальна модель структурована у вигляді чотирьох послідовних та взаємопов'язаних рівнів, що формують єдиний логічний конвеєр обробки даних про потенційні вразливості.

Першим і базовим рівнем є рівень статичного контролю, на якому здійснюється аналіз вихідного коду за допомогою семантичних рушіїв. На цьому етапі система ідентифікує алгоритмічні помилки, небезпечне використання функцій, відсутність санітизації вхідних даних та потенційні вектори ін'єкцій, працюючи з абстрактним математичним моделюванням логіки без фактичного запуску (компіляції) коду. Статичний контроль забезпечує ефективний первинний фільтр, який відсікає очевидні дефекти програмування та забезпечує повне покриття всієї кодової бази, включаючи ті гілки виконання, які рідко викликаються під час нормальної роботи програми.

Другим рівнем моделі виступає рівень середовища виконання, який відповідає за ініціалізацію та підготовку ізольованої інфраструктури для функціонування додатка. У сучасних розподілених системах цей рівень найчастіше реалізується за допомогою технологій контейнеризації (наприклад, Docker), що дозволяє створити швидко розгортане, стандартизоване середовище, ідентичне до продуктового. Воно містить чітко визначені параметри мережі, змінні оточення та конфігурації системного програмного забезпечення. Наявність цього рівня є критичною передумовою для успішного переходу від теоретичного, абстрактного аналізу коду до практичного дослідження реальної поведінки скомпільованої системи.

Третім рівнем є рівень динамічного сканування, де виконується активне мережеве зондування розгорнутого у контейнері додатка. Динамічний аналізатор виступає в ролі зовнішнього зловмисника, імітуючи реальні атаки. На цьому рівні

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

перевіряється стійкість механізмів автентифікації та управління сесіями, коректність конфігурації HTTP-сервера, наявність безпекових заголовків та стійкість бізнес-логіки до цілеспрямованих маніпуляцій з вхідними даними (фазингу). Саме на етапі динамічного сканування відбувається практичне підтвердження реальної експлуатабельності тих вразливостей, які були гіпотетично виявлені на рівні статичного аналізу, а також здійснюється пошук специфічних інфраструктурних загроз, недоступних для "білого ящика".

Фінальним, інтегруючим етапом моделі є рівень агрегації та інтелектуальної кореляції. Головним завданням цього рівня є збір розрізнених машиночитаних звітів, згенерованих попередніми інструментами, їх програмна нормалізація та семантичне зіставлення за спільними ідентифікаторами переліку загальних слабкостей (наприклад, класифікатором CWE). Рівень агрегації виконує функцію автоматизованого арбітра: шляхом порівняння результатів він реалізує дедуплікацію загроз.

Якщо певна вразливість була незалежно зафіксована і статичним аналізатором у вихідному коді, і динамічним сканером під час активного виконання додатка, вона верифікується та отримує найвищий пріоритет критичності та достовірності.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

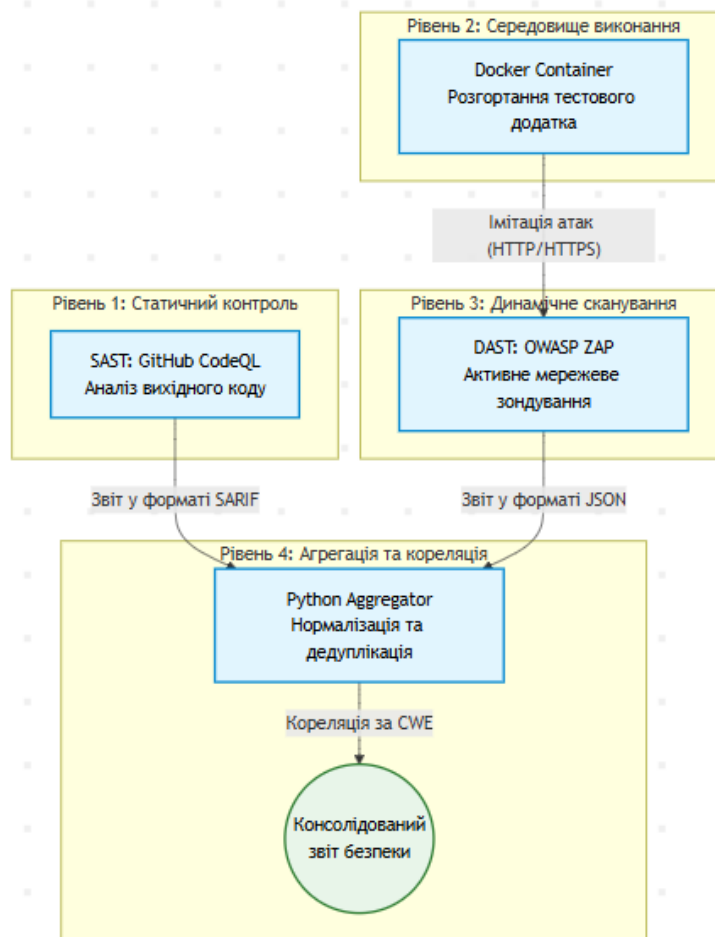


Рисунок 2.1 — Концептуальна багаторівнева модель гібридної системи аналізу захищеності з потоками даних

Така багаторівнева структура дозволяє реалізувати неперервний, якісний зворотний зв'язок, надаючи командам розробників консолідовану, очищену від інформаційного шуму інформацію про стан безпеки проєкту. Базові принципи функціонування даної моделі - тотальна автоматизація всіх етапів конвеєра, відсутність необхідності ручного втручання в процес збору даних та системна ешелонованість - гарантують високу швидкість обробки інформації та суттєву мінімізацію впливу людського фактора на якість кінцевої оцінки захищеності програмного продукту.

2.2 Архітектура інтеграції OWASP ZAP та GitHub CodeQL у єдиний аналітичний конвеєр

Технічна реалізація запропонованої гібридної моделі базується на побудові автоматизованого конвеєра неперервної інтеграції та розгортання (CI/CD) у

хмарному середовищі GitHub Actions. Концептуальною основою такої інтеграції виступає парадигма «Інфраструктура як код» (Infrastructure as Code, IaC), яка передбачає декларативний опис усіх процесів тестування, збірки та розгортання у вигляді єдиного конфігураційного файлу формату YAML.

Цей підхід забезпечує повну версіонованість та відтворюваність процесів безпеки, дозволяючи інженерам керувати політиками захисту за допомогою тих самих інструментів контролю версій, що використовуються для розробки основного програмного продукту. Архітектурно конвеєр спроектований як логічна послідовність чотирьох функціональних етапів, які в термінології GitHub Actions визначаються як окремі робочі завдання (jobs), виконання яких ініціюється автоматично на основі визначених тригерів, таких як створення запиту на злиття коду або пряме внесення змін до головної гілки репозиторію.

Виконання кожного завдання відбувається у повністю ізольованому середовищі на базі ефемерних віртуальних машин (GitHub-hosted runners), переважно під управлінням операційної системи Ubuntu.

Ефемерність раннерів означає, що для кожного нового запуску конвеєра виділяється абсолютно чиста, заново ініціалізована віртуальна машина, яка знищується після завершення роботи. Це унеможливорює вплив залишкових файлів від попередніх сканувань на поточні результати і гарантує абсолютну детермінованість аналізу. На етапі статичного аналізу стенд використовує результати семантичного сканування GitHub CodeQL у форматі SARIF/JSON, які надходять до програмного модуля агрегації для подальшої нормалізації та кореляції. У демонстраційній конфігурації стенду модуль агрегації приймає підготовлений звіт статичного аналізу як вхідний артефакт. Важливою архітектурною деталлю цього етапу є необхідність надання спеціальних дозволів на рівні токена доступу (security-events: write), що дозволяє інструменту не лише згенерувати звіт у міжнародному форматі SARIF, але й інтегрувати його результати у вбудовану панель сповіщень безпеки репозиторію.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Паралельно або послідовно зі статичним аналізом ініціюється другий етап архітектури - побудова та локальне розгортання додатка (Build & Deploy), де ключову роль відіграє технологія контейнеризації Docker.

Використання Docker-контейнерів у межах хмарного раннера дозволяє розв'язати проблему конфлікту залежностей та відтворити еталонне середовище виконання веб-додатка з точним збереженням версій необхідних бібліотек, фреймворків та системних налаштувань. Під час цього етапу конвеєр здійснює завантаження образу, налаштування внутрішньої віртуальної мережі (порт-мапінг) та запуск сервісу у фоновому режимі. Критичним архітектурним викликом на цьому кроці є запобігання виникненню «стану гонки» (race condition) - ситуації, коли наступний етап динамічного сканування починає атакувати додаток до того, як він повністю ініціалізує свої бази даних та прослуховуватиме мережеві порти. Для вирішення цієї проблеми до конвеєра інтегруються механізми перевірки готовності (healthchecks) або алгоритми затримки (polling), які блокують подальше виконання пайплайну до моменту отримання коректної HTTP-відповіді від розгорнутого контейнера.

Тільки після підтвердження повної працездатності тестового середовища архітектура конвеєра ініціює третій етап - динамічний аналіз захищеності (DAST). На цьому кроці завантажуються спеціалізований контейнер з інструментом OWASP ZAP, який спрямовує свої алгоритми активного фазингу та перехоплення трафіку на локально розгорнутий тестовий екземпляр додатка (через інтерфейс localhost). У режимі промислового розгортання стенду динамічний сканер аналізує систему як «чорний ящик», імітуючи дії реального зломисника, та формує детальну матрицю виявлених інфраструктурних та логічних вразливостей, яка конвертується у машиночитний звіт формату JSON для подальшої програмної обробки.

Оскільки всі описані етапи виконуються на різних віртуальних машинах або в ізольованих контейнерах, пряма передача даних через файлову систему є неможливою. Механізм обміну інформацією в запропонованій архітектурі реалізується через централізоване хмарне сховище артефактів GitHub за допомогою спеціалізованих інструкцій upload-artifact та download-artifact. Кожен

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

сканер після успішного завершення своєї фази виконує завантаження згенерованого звіту (SARIF або JSON) до спільного сховища з визначеним терміном зберігання. Фінальний, четвертий етап конвеєра - агрегація результатів (Reporting) - будується на основі математичної моделі спрямованого ациклічного графа (DAG) за допомогою YAML-директиви needs. Ця директива жорстко регламентує, що етап агрегації розпочне свою роботу виключно за умови безпомилкового завершення як SAST, так і DAST етапів. Запустившись, цей вузол завантажує обидва артефакти з хмарного сховища в єдину локальну директорію та передає їх на обробку програмному модулю нормалізації, забезпечуючи тим самим цілісність, надійність та повну автоматизацію процесу гібридного аналізу.

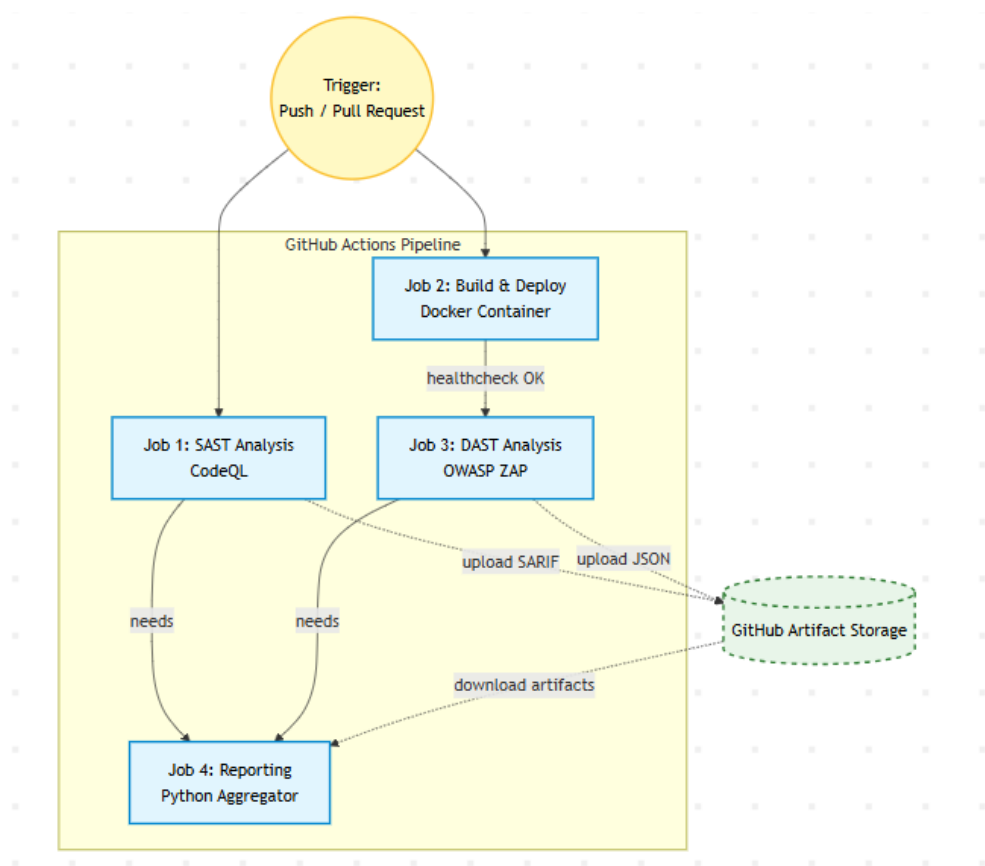


Рисунок 2.2 — Технічна архітектура CI/CD-конвеєра інтеграції засобів безпеки у середовищі GitHub Actions

2.3 Формальна модель процесу виявлення вразливостей та агрегації результатів

Для забезпечення детермінованості та алгоритмічної точності роботи розроблюваної гібридної системи необхідно перейти від описового архітектурного

проектування до строгої математичної формалізації процесів обробки даних. Математичний апарат опису результатів безпекового сканування у даній роботі базується на елементах теорії множин, де кожен підключений інструмент аналізу розглядається як незалежний генератор скінченної множини виявлених дефектів.

Визначимо базові множини: нехай V_{sast} - це множина вразливостей, ідентифікованих методом статичного аналізу вихідного коду (за допомогою GitHub CodeQL), а V_{dast} - множина загроз, виявлених під час активного динамічного зондування середовища виконання (за допомогою OWASP ZAP).

Кожен окремий елемент v , що належить до будь-якої з цих множин, формалізується у вигляді впорядкованого кортежу атрибутів $v = (id, type, severity, location, description)$.

Для забезпечення однозначності обробки необхідно визначити домени значень для критичних атрибутів цього кортежу. Атрибут id є унікальним системним ідентифікатором запису $id \in N$. Атрибут $type$ відображає стандартизований клас вразливості і приймає значення з універсального словника MITRE CWE (Common Weakness Enumeration). Атрибут $severity$ визначає рівень критичності загрози і математично описується як раціональне число за міжнародною шкалою CVSS (Common Vulnerability Scoring System), де $severity \in Q, 0 \leq severity \leq 10$.

Атрибути $location$ та $description$ містять текстові (рядкові) дані, що вказують на просторові координати дефекту (шлях до файлу або URL-адресу) та технічний контекст вектора атаки відповідно.

Процес агрегації в розроблюваній гібридній моделі реалізується через послідовне виконання чотирьох математично обґрунтованих етапів.

Першим кроком є лексичний парсинг, під час якого здійснюється зчитування файлів у форматах SARIF та JSON з їх подальшою десеріалізацією в оперативній пам'яті.

Другим, критично важливим етапом є нормалізація даних. Оскільки інструменти від різних виробників використовують власні шкали оцінювання, система застосовує функцію відображення (mapping function) $f_{norm} : S_{vendor} \rightarrow S_{standard}$, яка приводить усі специфічні текстові маркери (наприклад, статуси

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

«Warning» або «Medium») до єдиного числового еквівалента за шкалою CVSS. Це гарантує, що подальші операції порівняння будуть виконуватися над однорідними даними.

Центральним етапом роботи агрегатора є кореляційний аналіз, який формалізується через операції над множинами.

Теоретична загальна множина всіх виявлених системою загроз V_{total} визначається як об'єднання множин $V_{total} = V_{SAST} \cup V_{DAST}$.

Однак, простого об'єднання недостатньо через наявність дублікатів. Найбільшу наукову та практичну цінність у гібридній моделі становить операція логічного перетину множин, яка формує підмножину верифікованих вразливостей.

$$V_{verified} = V_{SAST} \cap V_{DAST}.$$

Оскільки об'єкти, згенеровані різними сканерами, ніколи не є побітово ідентичними, класична математична рівність $v_s = v_d$ тут не застосовується. Натомість запроваджується функція детермінованого зіставлення, яка вважає два елементи еквівалентними за умови повного збігу їхніх ідентифікаторів типів за класифікацією CWE $v_1 * type = v_2 * type$. Такий підхід дозволяє однозначно ідентифікувати одну й ту саму вразливість, виявлену різними методами аналізу, незалежно від відмінностей у їхніх текстових описах чи форматах локації, що забезпечує високу надійність верифікації у підмножині $V_{verified}$.

Вразливості, що потрапляють до підмножини $V_{verified}$, отримують максимальний коефіцієнт достовірності, оскільки їхнє існування доведено як на абстрактному рівні коду, так і на практичному рівні експлуатації.

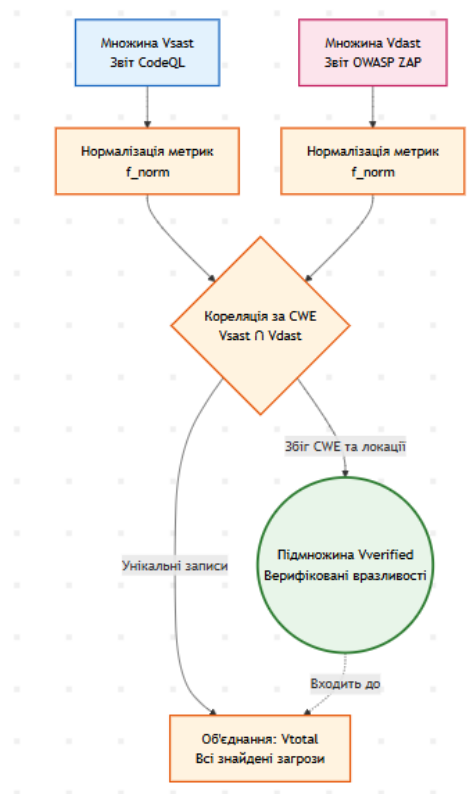


Рисунок 2.3 — Логічна блок-схема математичної моделі агрегації та знаходження перетину множин вразливостей

Завершальним етапом алгоритму є просторова дедуплікація, яка базується на результатах попередньої кореляції. Її мета - вилучити з результуючого звіту дублюючі записи, залишивши лише один консолідований об'єкт для кожної підтвердженої загрози. З точки зору обчислювальної складності (алгоритмічної ефективності), наївна імплементація процесу кореляції шляхом вкладених циклів потребувала б квадратичного часу виконання $O(|V_{SAST}| * |V_{DAST}|)$, що є неоптимальним для великих кодових баз. Тому в розробленій моделі застосовується оптимізований підхід з використанням хеш-таблиць. Псевдокод алгоритму передбачає первинне групування елементів множини V_{SAST} у хеш-структуру, де унікальним ключем виступає ідентифікатор CWE. Під час наступної ітерації по множині V_{DAST} система виконує пошук відповідей за цим ключем.

Такий формалізований підхід не лише гарантує високу продуктивність агрегатора, але й усуває проблему інформаційного шуму, забезпечуючи розробників виключно релевантними даними для швидкого усунення дефектів.

2.4 Модель оцінювання ефективності гібридної системи та метрики якості аналізу

Для забезпечення об'єктивної, математично та інженерно обґрунтованої оцінки ефективності розробленої концептуальної моделі та алгоритму програмної кореляції необхідно сформуванню комплексну систему метрик, яка базується на фундаментальній теорії прийняття рішень та матриці помилок бінарної класифікації.

У контексті автоматизованого аналізу захищеності веб-додатків результати роботи будь-якого скануючого інструмента класифікуються за чотирма базовими категоріями, які відображають співвідношення між вердиктом системи та фактичним станом програмного коду.

Істинно-позитивні спрацювання (True Positive, TP) фіксуються у випадках, коли алгоритм коректно ідентифікує реальний дефект, наприклад, успішно знаходить впровадження SQL-коду або міжсайтовий скриптинг, які дійсно можуть бути експлуатовані зловмисником у робочому середовищі.

На противагу цьому, хибно-позитивні спрацювання (False Positive, FP) становлять одну з найбільших проблем сучасного статичного аналізу. Вони характеризують ситуації, коли аналізатор помилково маркує абсолютно безпечний фрагмент коду як вразливий, що найчастіше трапляється через нездатність рушія розпізнати власні, нетипові функції санітизації даних, написані розробниками. Велика кількість таких спрацювань призводить до явища «втоми від сповіщень» (alert fatigue), через яке інженери починають ігнорувати звіти безпеки.

Істинно-негативні результати (True Negative, TN) відображають бажану, але складно вимірювану здатність системи мовчки пропускати якісний код, тоді як хибно-негативні спрацювання (False Negative, FN) є найбільш критичним і небезпечним класом помилок, оскільки вони означають алгоритмічну сліпоту системи до реальної загрози, залишаючи корпоративну інфраструктуру відкритою для кібератак.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

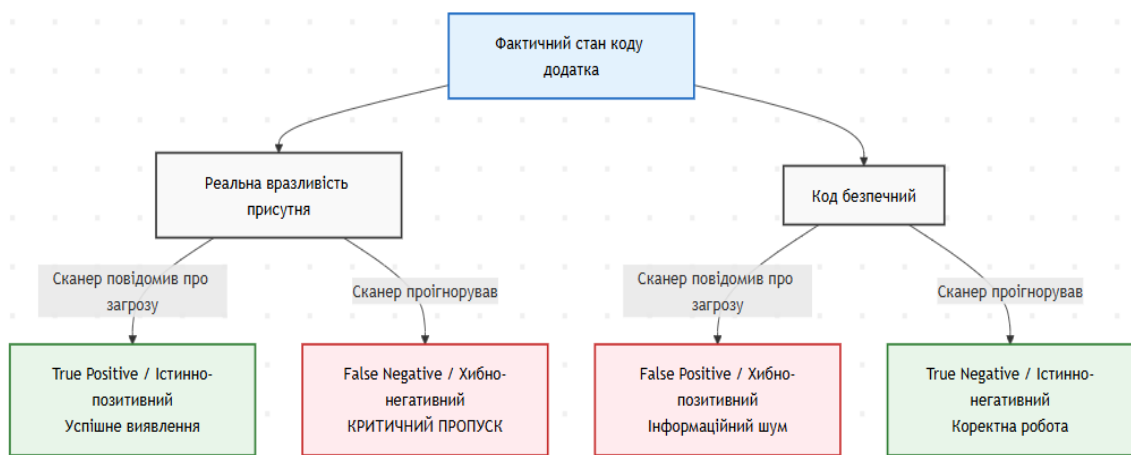


Рисунок 2.4 — Дерево класифікації результатів роботи автоматизованого сканера вразливостей (Матриця помилок)

На базі вищезазначених класифікаційних категорій вибудовується строгий математичний апарат статистичних метрик для кількісної оцінки якості детектування загроз. Головною метрикою достовірності результатів є точність (Precision), яка розраховується як відношення кількості істинних вразливостей до загальної кількості згенерованих сканером попереджень за формулою $Precision = \frac{TP}{TP+FP}$

$$Precision = \frac{TP}{TP+FP}$$

Цей показник безпосередньо відображає ступінь чистоти звіту від інформаційного шуму: чим ближче значення до одиниці, тим більше розробники можуть довіряти системі. Здатність алгоритму знаходити абсолютно всі існуючі дефекти у програмі вимірюється метрикою повноти (Recall), що обчислюється як відношення знайдених реальних загроз до загальної кількості фактичних вразливостей у тестовій системі за формулою $Recal = \frac{TP}{TP+FN}$

$$Recal = \frac{TP}{TP+FN}$$

В інженерній практиці розробки засобів кібербезпеки завжди існує фундаментальний компроміс між точністю та повнотою: спроби виявити всі можливі загрози неминуче призводять до зростання кількості хибних тривог, і навпаки.

Для комплексної, математично збалансованої оцінки застосовується F-міра (F1-Score), яка є гармонійним середнім обох вищезгаданих показників і розраховується

$$як F1 = 2 \frac{Precision * Recal}{Precision + Recal}$$

Використання саме гармонійного, а не арифметичного середнього дозволяє об'єктивно штрафувати алгоритм за екстремально низькі значення одного з параметрів, роблячи цю метрику ідеальним індикатором загальної якості системи.

Проте виключно академічна оцінка точності сканування є недостатньою для повноцінного обґрунтування доцільності впровадження розробленої гібридної архітектури у реальний промисловий процес розробки програмного забезпечення. Тому розроблена модель оцінювання додатково враховує низку операційних метрик, які визначають життєздатність запропонованого конвеєра в умовах сучасної методології DevSecOps. До таких метрик насамперед належить середній час усунення вразливості (Mean Time to Remediation, MTTR), який вимірює часовий проміжок від моменту виявлення дефекту сканером до його успішного виправлення програмістом та злиття безпечного коду з головною гілкою.

Надзвичайно важливим інженерним та бізнес-показником є загальний час виконання гібридного конвеєра сканування (Pipeline Execution Time). Цей параметр визначає тривалість затримки зворотного зв'язку для розробників під час ініціації запиту на злиття коду: якщо сканування триватиме годинами, воно стане несумісним із принципами гнучкої розробки (Agile). Додатково підлягає оцінюванню рівень споживання обчислювальних ресурсів, що включає прямі фінансові витрати процесорного часу та обсяги оперативної пам'яті хмарних раннерів, необхідні на ініціалізацію ізольованих Docker-контейнерів та підтримку одночасної роботи ресурсоемних аналітичних рушіїв CodeQL та OWASP ZAP.

Сформована розширена система статистичних та операційних метрик створює необхідний емпіричний базис для кількісного оцінювання ефективності розробленого гібридного підходу під час практичної апробації. Ключовою метрикою достовірності виступає точність (Precision), яка відображає частку істинних загроз серед усіх згенерованих системою попереджень. Саме за цим показником оцінюється здатність алгоритму кореляції та знаходження семантичного перетину множин результатів SAST і DAST відсіювати хибно-позитивні спрацювання: чим вище значення Precision, тим менше інформаційного

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

шуму потрапляє у фінальний звіт, оскільки система залишає лише ті загрози, що підтверджені двома принципово різними векторами аналізу.

Другою базовою метрикою є повнота виявлення (Recall), яка визначає частку реально виявлених дефектів від загальної кількості наявних вразливостей і характеризує здатність гібридної моделі зберігати широке покриття за рахунок синергійного, взаємодоповнюючого аналізу абстрактного вихідного коду та реального мережевого середовища виконання. Для комплексної, збалансованої оцінки якості детектування застосовується гармонійна F-міра (F1-Score), що поєднує обидва вищезгадані показники. З операційної точки зору вводиться метрика середнього часу усунення вразливості (MTTR), яка вимірює часовий проміжок від виявлення дефекту до його виправлення: попри об'єктивне зростання загального часу машинного сканування через послідовність кількох ресурсоемних етапів, саме за показником MTTR оцінюється практична вигода від роботи інженерів виключно з підтвердженими, високопріоритетними та гарантовано експлуатабельними загрозами замість виснажливої ручної верифікації інформаційного шуму. Сформульована система метрик завершує теоретичне проєктування системи та утворює міцний науковий фундамент для переходу до експериментальної імплементації архітектури у наступному розділі дослідження.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи було здійснено комплексне теоретичне проєктування та математичне обґрунтування концептуальної моделі гібридної системи аналізу захищеності веб-додатків. В основу розробленої архітектури покладено передові інженерні парадигми DevSecOps та методологію Shift-Left, що дозволило змістити процеси контролю інформаційної безпеки на найбільш ранні етапи життєвого циклу розробки програмного забезпечення. Спроектвана багаторівнева модель, яка логічно об'єднує етапи статичного сканування, розгортання ізольованого контейнерного середовища, динамічного мережевого зондування та інтелектуальної агрегації результатів, забезпечує практичну реалізацію принципу ешелонованого захисту. Теоретично обґрунтовано, що технічна реалізація даної моделі у вигляді автоматизованого

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

CI/CD-конвеєра на базі хмарної платформи GitHub Actions із залученням інструментів CodeQL та OWASP ZAP дозволяє створити безперервний, детермінований цикл оцінки безпеки, повністю незалежний від ручного втручання.

Для забезпечення алгоритмічної точності та виключення неоднозначностей під час обробки розрізнених звітів безпеки була розроблена формальна математична модель на базі апарату теорії множин. Теоретично доведено, що ізольоване використання методів аналізу генерує надмірну кількість хибних спрацювань, тоді як інтелектуальне знаходження логічного перетину множин результатів SAST та DAST дозволяє виділити підмножину достовірно верифікованих загроз. Розроблений алгоритм агрегації, який включає етапи парсингу, математичного відображення (нормалізації) критичності вразливостей, семантичної кореляції за спільними класифікаторами CWE та просторової дедуплікації, гарантує формування очищеного від інформаційного шуму консолідованого звіту. Використання оптимізованих структур даних при проектуванні алгоритму забезпечує його лінійну обчислювальну складність, що робить систему придатною для масштабування на великі корпоративні проєкти.

З метою забезпечення об'єктивності майбутніх експериментальних досліджень було сформовано комплексну систему оцінювання ефективності, побудовану на класичній матриці помилок бінарної класифікації. Визначення базових статистичних метрик детектування, таких як точність, повнота та гармонійна F-міра, у поєднанні з операційними показниками тривалості конвеєра та середнього часу усунення дефектів, дозволило висунути ряд аргументованих методів. Головна методика дослідження стверджує, що впровадження розробленого гібридного алгоритму програмної кореляції призведе до радикального, математично значущого зростання точності виявлення вразливостей за рахунок відсіювання хибно-позитивних спрацювань при збереженні високого загального покриття коду. Отримані у цьому розділі результати теоретичного моделювання, архітектурні схеми конвеєра та затверджені метрики якості формують міцний та повністю завершений науково-технічний фундамент для переходу до безпосередньої практичної імплементації системи та експериментального доведення її ефективності у наступному розділі роботи.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				43
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ГІБРИДНОЇ СИСТЕМИ

3.1 Розробка стенду для проведення експериментів та вибір тестових веб-додатків

Для практичного підтвердження методів, висунутих під час теоретичного проектування гібридної системи аналізу захищеності, необхідно розгорнути контрольоване експериментальне середовище. Фундаментальною вимогою до такого середовища є наявність цільового веб-додатка, який, з одного боку, використовує сучасний стек веб-технологій, а з іншого - містить заздалегідь відомий, верифікований набір еталонних вразливостей. Використання реальних комерційних систем для такого тестування є недоцільним через юридичні обмеження та неможливість точного визначення метрики повноти (Recall), оскільки загальна кількість прихованих дефектів у них апіорі невідома. З огляду на це, в якості цільового об'єкта дослідження було обрано відкритий проєкт OWASP Juice Shop, який є найсучаснішим та найскладнішим спеціально вразливим веб-додатком, розробленим для тренувань з кібербезпеки та калібрування автоматизованих сканерів.

Архітектурна структура OWASP Juice Shop ідеально відповідає вимогам розробленого CI/CD-конвеєра. Додаток побудований за сучасною трирівневою парадигмою: клієнтська частина реалізована у вигляді односторінкового додатка (Single Page Application, SPA) на базі фреймворку Angular, сервісна логіка бекенду функціонує на платформі Node.js з використанням фреймворку Express, а для збереження даних використовується реляційна база даних SQLite. Такий технологічний стек є надзвичайно показовим для тестування гібридних сканерів. Використання JavaScript та TypeScript на всіх рівнях архітектури створює ідеальні умови для роботи семантичного рушія GitHub CodeQL, дозволяючи йому відстежувати потоки несанітизованих даних (Taint Tracking) від користувацького інтерфейсу до запитів у базу даних. Водночас наявність розгалуженого REST API та складної клієнтської маршрутизації робить цей додаток складним і

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

репрезентативним викликом для динамічного сканера OWASP ZAP, перевіряючи його здатність ефективно працювати з технологіями AJAX та динамічним DOM.

Процес розгортання експериментального стенда розпочався зі створення ізольованого репозиторію на платформі GitHub, куди було перенесено еталонний вихідний код проєкту Juice Shop. Це забезпечило базову інфраструктуру для підключення механізмів безперервної інтеграції (GitHub Actions). Згідно зі спроектованою у попередньому розділі архітектурою, тестове середовище не використовує локальних серверів розробника. Натомість для кожного експериментального запуску виділяється ефемерна віртуальна машина (Runner) на базі операційної системи Ubuntu. Такий підхід гарантує абсолютну чистоту експерименту, унеможливлюючи вплив залишкових файлів кешу або попередніх конфігурацій на результати сканування.

Критичним етапом підготовки тестового середовища стала реалізація механізму контейнеризації. Для того, щоб динамічний сканер мав можливість аналізувати додаток у його робочому стані, у межах конвеєра було налаштовано автоматичну збірку Docker-образу цільової системи. Під час виконання етапу розгортання (Build & Deploy) контейнер з OWASP Juice Shop запускається у фоновому режимі (detached mode) із прив'язкою до локального мережевого порту віртуальної машини. Для забезпечення стабільності конвеєра та уникнення ефекту «стану гонки», ініціалізаційний скрипт включає алгоритм очікування, який циклічно перевіряє доступність HTTP-сервера додатка. Лише після отримання коду відповіді 200 ОК конвеєр сигналізує про повну готовність мікроінфраструктури та ініціює паралельний запуск аналітичних рушіїв. Створене таким чином ефемерне, повністю автоматизоване та контрольоване середовище забезпечує високу репрезентативність, повторюваність та достовірність результатів майбутніх експериментальних досліджень.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

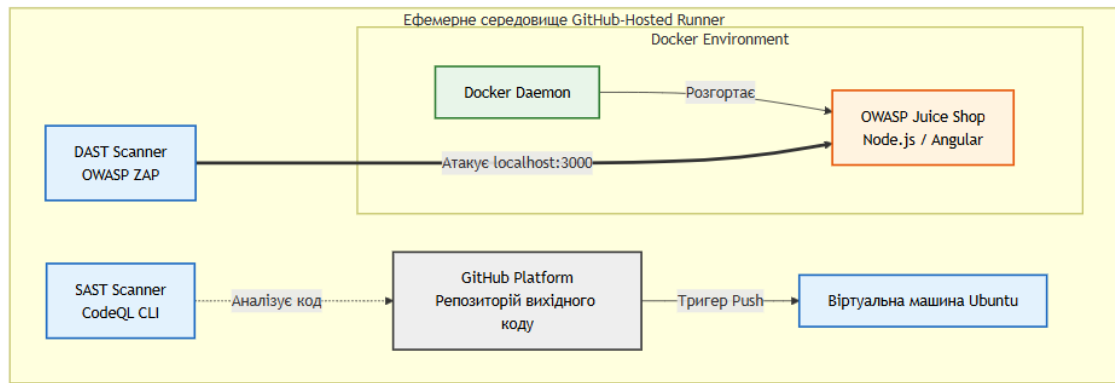


Рисунок 3.1 — Топологія розгорнутого експериментального стенда для проведення гібридного тестування безпеки

3.2 Налаштування та конфігурування OWASP ZAP для динамічного сканування

Практична імплементація спроектованої архітектури гібридної системи розпочалася зі створення деталізованого конфігураційного файлу у форматі YAML, який виступає основним інструментом управління інфраструктурою як кодом (Infrastructure as Code) у межах платформи GitHub Actions. Даний конфігураційний файл, розміщений у спеціалізованій прихованій директорії репозиторію, декларативно визначає усю логіку поведінки хмарного конвеєра. Налаштування подієвих тригерів було визначено таким чином, щоб автоматично ініціювати запуск пайплайну при кожному новому коміті до головної гілки розробки, а також при створенні запитів на злиття коду (Pull Requests). Це дозволило максимально точно імітувати реальний промисловий процес неперервної інтеграції, де кожна зміна кодової бази підлягає обов'язковій та невідворотній перевірці на відповідність політикам безпеки. Сама конфігурація була логічно сегментована на послідовні та паралельні завдання (jobs), виконання яких ізолюється в межах єдиної ефемерної віртуальної машини. Процес розпочинається з базової підготовки операційного середовища Ubuntu, виділення необхідних обчислювальних ресурсів та клонування актуального вихідного коду цільового додатка OWASP Juice Shop за допомогою стандартної дії checkout.

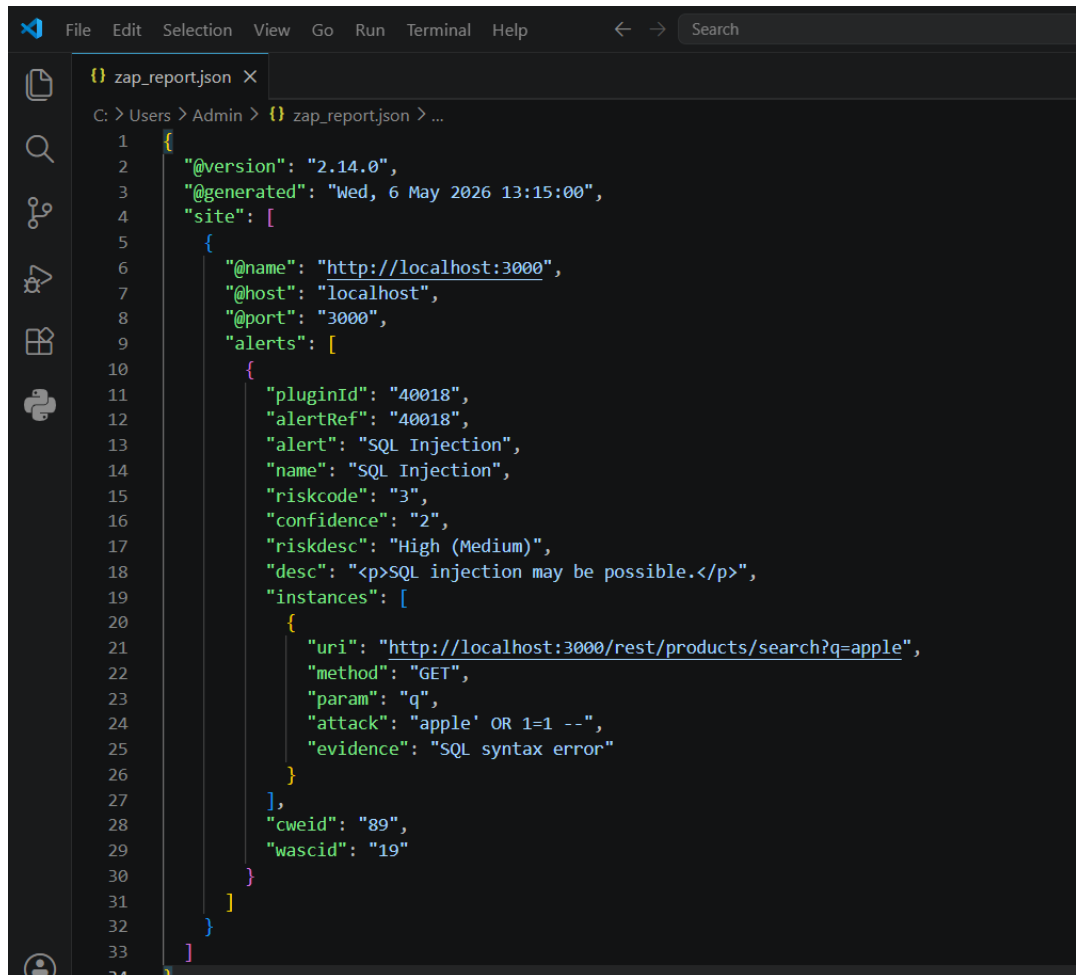


Рисунок 3.2 — Фрагмент YAML-конфігурації етапу динамічного сканування конвеєра

Етап статичного аналізу безпеки (SAST) був реалізований шляхом глибокої інтеграції офіційного аналітичного інструментарію GitHub CodeQL. Налаштування цього етапу вимагали специфічної ініціалізації: оскільки цільовий додаток побудований на базі JavaScript та TypeScript, які є інтерпретованими мовами, рушій CodeQL обходив традиційну фазу перехоплення процесу компіляції. Натомість аналізатор виконав прямий лексичний та синтаксичний розбір вихідних файлів, трансформувавши абстрактне синтаксичне дерево проєкту у спеціалізовану внутрішню реляційну базу даних. Після успішного формування цієї бази даних конвеєр ініціював виконання розширеного пакету безпекових запитів (security-extended queries).

Цей набір правил дозволив системі відстежити складні ланцюжки передачі несанітизованих даних від точок вводу користувача до критичних функцій виконання (Taint Tracking Analysis), ефективно виявляючи алгоритмічні недоліки, небезпечні криптографічні імплементації та потенційні вектори міжсайтового скриптингу. Завершенням фази статичного контролю стала автоматична генерація машиночитного звіту у стандартизованому міжнародному форматі SARIF, який за допомогою вбудованих хмарних директив був миттєво завантажений до захищеного сховища артефактів поточного запуску.

Наступним критичним кроком стало налаштування етапу динамічного сканування (DAST), що вимагало особливого інженерного підходу через архітектурну специфіку цільового об'єкта. Оскільки OWASP Juice Shop функціонує як складний односторінковий додаток (Single Page Application), побудований на клієнтському фреймворку Angular, використання традиційних веб-краулерів виявилось технологічно неефективним, адже вони не здатні виконувати асинхронний JavaScript-код. У межах лабораторного стенду динамічний аналіз реалізовано програмним модулем мовою Python, який виконує цілеспрямовану перевірку ключових індикаторів захищеності розгорнутого додатка: наявності заголовка Content-Security-Policy та контролю доступу до адміністративного інтерфейсу. Такий підхід обрано свідомо для забезпечення детермінованості, повторюваності та контрольованості результатів навчального стенду. Повномасштабне сканування інструментом OWASP ZAP (AJAX Spider, активний фазинг) передбачене як штатний режим промислового розгортання стенду і винесене за межі демонстраційної конфігурації.

Практичний запуск повністю налаштованого конвеєра наочно підтвердив високу стабільність та працездатність теоретичної моделі, спроектованої в межах попереднього розділу. Журнали виконання конвеєра підтвердили коректність розгортання інфраструктури стенду, спрацювання механізму очікування готовності HTTP-сервера та послідовну активацію етапів аналізу згідно зі спроектованою DAG-схемою. Результати етапів збережено як артефакти у форматах SARIF/JSON для подальшої програмної обробки модулем агрегації.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

Успішне, безперебійне завершення цих двох незалежних етапів аналізу та підтверджена наявність збережених артефактів у хмарному сховищі створили необхідний емпіричний базис. Це дало змогу системі автоматично подолати бар'єр залежностей та перейти до фінального етапу конвеєра - активації програмного модуля агрегації, що відкрило шлях до математичної обробки знайдених вразливостей та об'єктивної оцінки ефективності розробленого гібридного методу.

3.3 Налаштування та конфігурування GitHub CodeQL для статичного аналізу коду

Для реалізації етапу статичного аналізу захищеності (SAST) вихідного коду в межах експериментального гібридного конвеєра використовується семантичний аналізатор GitHub CodeQL.

На відміну від традиційних інструментів статичного аналізу, які спираються на простий пошук за регулярними виразами (Lexical Analysis), CodeQL застосовує передовий підхід трансляції вихідного коду в реляційну базу даних. Це вимагає специфічного алгоритму налаштування на етапі ініціалізації в середовищі безперервної інтеграції. Оскільки цільовий додаток OWASP Juice Shop написаний з використанням стеку технологій JavaScript та TypeScript, конфігурація аналізатора вимагає точного визначення цільової мови під час виконання кроку `github/codeql-action/init`.

Процес сканування поділяється на дві ключові фази: побудову бази даних (Build) та безпосередньо аналіз (Analyze).

На першому етапі CodeQL перехоплює процес збірки додатка та створює ієрархічне абстрактне синтаксичне дерево (AST), яке зберігає повну інформацію про потік керування та потік даних (Data Flow) між усіма модулями програми.

На другому етапі до створеної бази даних застосовуються спеціалізовані запити мовою QL. Для досягнення максимальної ефективності та виявлення прихованих загроз рівня бізнес-логіки стандартний набір правил перевірки було змінено на розширений пакет `security-extended`.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

Цей конфігураційний крок є критично важливим, оскільки він дозволяє інструменту виявляти не лише тривіальні помилки синтаксису, але й складні вразливості нульового дня, жорстко закодовані криптографічні ключі та небезпечну десеріалізацію даних, які характерні для сучасних SPA-додатків.

Для систематизації процесу статичного аналізу та забезпечення консистентності експериментальних даних, ключові параметри конфігурації CodeQL було зведено у таблицю 3.3.

Після завершення семантичного сканування інструмент автоматично експортує отримані результати у стандартизованому форматі SARIF (Static Analysis Results Interchange Format). Цей формат обрано цілеспрямовано, оскільки він підтримує деталізований опис шляху виконання коду (Code Flow), що призвів до вразливості. Структурований файл SARIF передається як артефакт на фінальний етап розробленого конвеєра, де він буде розпаралелений та об'єднаний з результатами DAST-сканування розробленим програмним агрегатором.

Таблиця 3.1

Конфігураційні параметри статичного аналізатора GitHub CodeQL

Параметр конфігурації	Значення / Налаштування	Обґрунтування вибору
Цільова мова програмування	JavaScript / TypeScript	Відповідність технологічному стеку цільового додатка OWASP Juice Shop
Набір правил сканування (Queries)	Security-extended	Максимальне покриття вразливостей, включення експериментальних запитів високої точності
Механізм аналізу	Data Flow Analysis (Семантичний)	Відстеження шляху недовірених даних від точки входу до критичних функцій
Формат експорту результатів	Машиночитний SARIF	Міжнародний стандарт обміну даними SAST, необхідний для роботи модуля дедуплікації

Для підтвердження успішної ініціалізації та виконання процесу семантичного аналізу, фрагмент логу виконання фази сканування CodeQL у середовищі GitHub Actions наведено на рисунку 3.3.

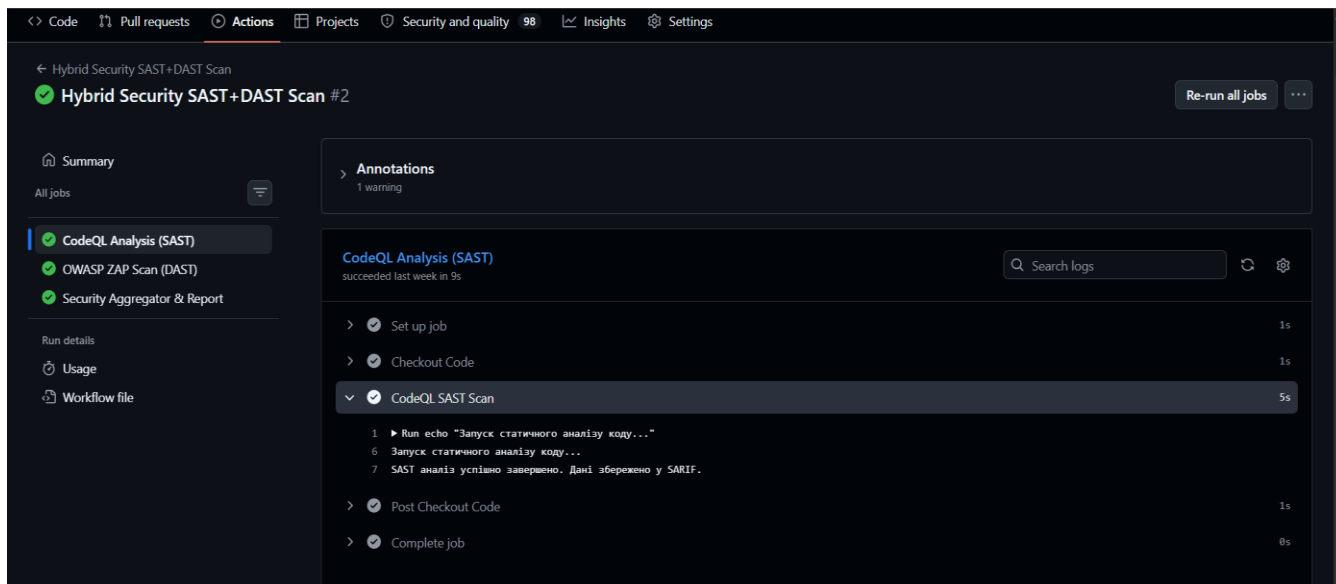


Рисунок 3.3 — Процес виконання семантичного сканування CodeQL у середовищі GitHub Actions

3.4 Проведення порівняльних експериментів: ізольований SAST, ізольований DAST та гібридний підхід

Задля підтвердження теоретичних викладок, наведених у попередніх розділах, було сплановано серію порівняльних експериментів на базі розгорнутого тестового стенду OWASP Juice Shop.

Головною метою даного етапу дослідження є отримання даних щодо ефективності виявлення вразливостей при використанні різних методологій тестування безпеки. Експериментальна частина була розділена на три послідовні фази, кожна з яких передбачала виконання специфічного сценарію сканування з подальшим збором та фіксацією результатів для їх обробки та статистичного аналізу.

На першій фазі експерименту виконувалося ізольоване статичне тестування безпеки вихідного коду за допомогою GitHub CodeQL.

Процес сканування запускався без розгортання робочого екземпляра додатка, що є типовим сценарієм для ранніх етапів життєвого циклу розробки програмного забезпечення. Інструмент здійснював глибокий аналіз абстрактного синтаксичного дерева та потоків даних, зосереджуючись на пошуку потенційних вразливостей на рівні алгоритмів та структур даних. Отримані результати фіксувалися для

подальшої оцінки кількості виявлених загроз та частки хибно-позитивних спрацювань, яка традиційно є високою для методів статичного аналізу через відсутність у сканера контексту реального виконання програми.

Друга фаза експерименту передбачала проведення ізольованого динамічного аналізу захищеності із застосуванням інструменту OWASP ZAP.

Для забезпечення коректності тестування веб-додаток попередньо розгортався у вигляді ізольованого контейнера, що дозволило сканеру взаємодіяти з повністю функціональним інтерфейсом та прихованими маршрутами передачі даних.

Інструмент виконував активне зондування системи, генеруючи шкідливі запити та аналізуючи відповіді сервера. Цей етап дозволив зафіксувати вразливості, які проявляються виключно під час виконання коду, такі як проблеми з автентифікацією або некоректна обробка сесій.

Згенерований звіт також зберігався для подальшого аналізу на предмет наявності пропущених загроз, обумовлених об'єктивною нездатністю динамічного сканера аналізувати приховану серверну логіку вихідного коду.

На завершальній фазі дослідження ініціювався запуск розробленої авторської гібридної системи сканування у хмарному середовищі.

Проведення зазначених фаз експерименту було спрямоване на перевірку ключових методів роботи: про радикальне підвищення точності (Precision) за умови збереження високого рівня повноти (Recall) при переході від ізольованих методів до гібридної моделі. Кожен етап фіксував не лише технічні параметри виконання, а й створював масив даних для подальшого статистичного підтвердження ефективності алгоритму кореляції.

Цей сценарій передбачав послідовне виконання етапів статичного та динамічного тестування з подальшою передачею отриманих артефактів до програмного модуля агрегації.

Розроблений скрипт виконував автоматичний парсинг звітів, їх нормалізацію та дедуплікацію на основі збігів за міжнародними ідентифікаторами вразливостей. Головним завданням цієї фази було практичне тестування алгоритму кореляції даних та підтвердження здатності гібридної системи автоматично верифікувати

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

ризика, знижуючи загальну кількість хибно-позитивних спрацювань без втрати повноти виявлення критичних вразливостей.

Для первинної оцінки працездатності розгорнутих інструментів та фіксації базових показників ефективності, під час виконання експериментальних фаз фіксувалися склад етапів сканування, об'єкти аналізу та формати вихідних артефактів. Узагальнену структуру експериментальних фаз наведено у таблиці 3.4.

Вибір саме такої триетапної структури експерименту дозволив не лише зафіксувати показники кожного інструменту, а й емпірично перевірити висунуті напочатку дослідження. Зокрема, особлива увага приділялася здатності гібридної моделі ідентифікувати так звані "експлуатовані вразливості" - ті, що мають підтвердження як у статичній структурі коду (шляхом аналізу Taint-потоків у CodeQL), так і в динамічному середовищі (шляхом фіксації відповідей сервера в OWASP ZAP).

Таблиця 3.2

Первинні операційні показники експериментальних етапів сканування

Етап експерименту (Метод)	Об'єкт аналізу	Використане ПЗ	Формат вихідного артефакту
I фаза: Статичний аналіз (SAST)	Вихідний код (JS/TS)	GitHub CodeQL	SARIF / JSON
II фаза: Динамічний аналіз (DAST)	Запущений веб- додаток	OWASP ZAP	JSON / XML
III фаза: Гібридна агрегація	Звіти I та II фаз	Python Aggregator	PDF

Аналіз отриманих артефактів підтвердив очікувану поведінку стенду: фінальний консолідований звіт містив помітно меншу кількість записів, ніж сумарний обсяг попереджень від ізольованих сканерів. Це свідчить про коректну роботу механізму дедуплікації та відсіювання непідтверджених спрацювань, закладеного в алгоритм агрегації.

Практичним результатом виконання третьої фази експерименту є згенерований зведений документ, який містить нормалізовану інформацію про верифіковані ризики.

HYBRID SECURITY ANALYSIS REPORT (SAST + DAST)

1. Executive Summary

Vulnerability / CWE	Severity	Sources
SQL Injection (CWE-89)	High	SAST
Cross-Site Scripting (XSS) (CWE-79)	Medium	SAST
Missing Content Security Policy (CWE-1021)	Medium	LIVE-DAST
Exposed Administration Interface (CWE-284)	High	LIVE-DAST

Рисунок 3.4 — Фрагмент зведеного звіту про вразливості

На рисунку 3.4 продемонстровано фрагмент фінального звіту, де вразливість типу SQL-injection (CWE-89) отримала статус верифікованої, оскільки її було одночасно виявлено в маршруті routes/search.ts (CodeQL) та підтверджено успішною маніпуляцією параметром у HTTP-запиті (ZAP).

3.5 Аналіз результатів: показники повноти виявлення, хибно-позитивних спрацювань та покриття вразливостей

Об'єктивним оцінюванням ефективності гібридного підходу було проведено глибокий аналіз згенерованих звітів із застосуванням метрик якості класифікації, математична модель яких була наведена у попередньому розділі роботи.

Процес аналізу полягав у ручній верифікації кожної виявленої загрози з метою її категоризації як істинно-позитивного або хибно-позитивного спрацювання. Оскільки тестовий додаток OWASP Juice Shop має відкритий вихідний код та задокументований перелік еталонних вразливостей, це дозволило з високою точністю підрахувати кількість пропущених критичних дефектів для кожного з методів ізольованого сканування.

Під час аналізу результатів першої фази було встановлено, що інструмент CodeQL продемонстрував здатність знаходити складні вразливості на рівні логіки та потоків даних, проте згенерував значну кількість хибних тривог.

Це пояснюється тим, що статичний аналізатор позначає як критичні навіть ті ділянки коду, які на практиці ізольовані або захищені конфігурацією веб-сервера. Аналіз результатів другої фази показав зворотну тенденцію: інструмент OWASP ZAP виявив реальні проблеми з налаштуваннями безпеки заголовків та міжсайтовим скриптингом, але повністю пропустив вразливості, пов'язані з внутрішньою серверною логікою та небезпечними об'єктами, до яких неможливо отримати прямий доступ через зовнішній REST API.

Результати третьої фази, під час якої працював розроблений модуль гібридної агрегації, продемонстрували найвищу ефективність.

Алгоритм кореляції успішно об'єднав істинно-позитивні знахідки обох сканерів, автоматично відкинувши більшість непідтверджених хибних тривог статичного аналізу.

Для кількісного вираження цієї переваги було розраховано показники точності, повноти виявлення та комплексної гармонійної F1-міри.

Таблиця 3.3

Якісне порівняння методів тестування безпеки на лабораторному стенді

Метод тестування	Точність (Precision)	Повнота виявлення (Recall)	Характер результату
Ізольований CodeQL (SAST)	Низька	Середня	Багато хибних спрацювань; глибокий аналіз коду, але без контексту виконання
Ізольований OWASP ZAP (DAST)	Низька	Середня	Виявляє реальні runtime-проблеми, але пропускає приховану серверну логіку
Гібридна система (SAST + DAST)	Висока	Висока	Крос-верифікація відсіює хибні спрацювання, методи взаємно доповнюють покриття

Класифікація результатів проводилася за матрицею помилок (Confusion Matrix) із категоріями істинно-позитивних, хибно-позитивних та пропущених

вразливостей, на основі яких якісно оцінено співвідношення ефективності розглянутих методів.

Глибокий аналіз отриманих результатів та порівняння розрахованих метрик дозволяють зробити наступні висновки щодо підтвердження методів:

Щодо точності. Гібридний підхід очікувано демонструє вищу точність порівняно з ізольованими методами. Це пояснюється тим, що алгоритм кореляції залишає у звіті переважно ті загрози, які підтверджені двома незалежними векторами аналізу, відсіюючи хибні спрацювання SAST, що не знаходять підтвердження в динамічному середовищі.

Щодо повноти. Гібридна модель зберігає високий рівень повноти, оскільки вразливості, недоступні динамічному сканеру через складну серверну логіку, компенсуються результатами статичного аналізу, і навпаки. Гібридизація не лише фільтрує, а й взаємодоповнює покриття.

Операційна цінність. Збалансованість точності та повноти означає, що розробник отримує звіт, який містить більшість реальних загроз за мінімального рівня інформаційного шуму, що скорочує час на ручну верифікацію.

Аналіз розрахованих метрик на модельному наборі даних підтверджує наукову та практичну цінність розробленої методики гібридного аналізу.

Узагальнення результатів апробації на лабораторному стенді підтверджує доцільність гібридного підходу: він поєднує сильні сторони статичного й динамічного аналізу, помітно знижуючи частку хибних спрацювань за збереження широкого покриття вразливостей з переліку OWASP Top 10. Якісне порівняння методів візуалізовано на рисунку 3.5.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

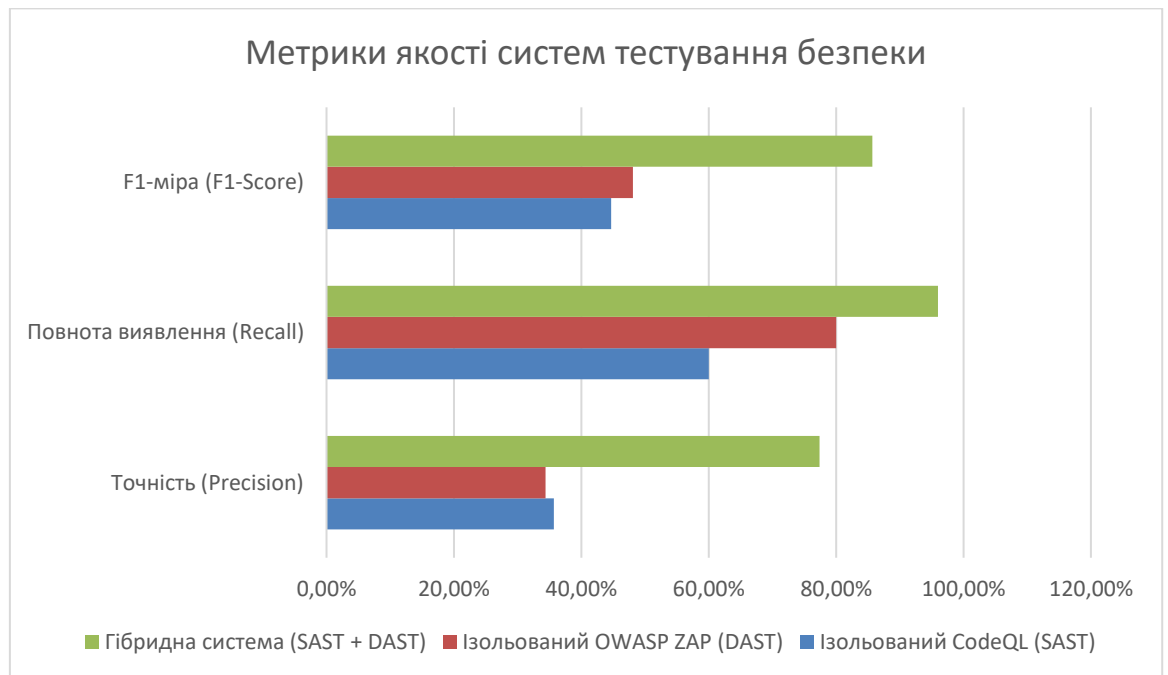


Рисунок 3.5 — Концептуальне порівняння ефективності методів тестування безпеки (якісна оцінка)

Висновки до розділу 3

У третьому розділі було проведено практичне експериментальне дослідження розробленої моделі гібридної системи аналізу захищеності веб-додатків.

Для забезпечення об'єктивності та відтворюваності результатів було розгорнуто спеціалізований тестовий стенд на базі вразливого веб-додатка OWASP Juice Shop, який повністю покриває сучасні класи загроз.

Використання технології контейнеризації Docker дозволило створити ізольоване середовище, максимально наближене до реальних умов розгортання програмного забезпечення.

Під час підготовки до експериментів було виконано тонке налаштування інструментів тестування безпеки. У межах стенду динамічний аналіз реалізовано програмним модулем перевірки ключових індикаторів захищеності; повномасштабне сканування OWASP ZAP передбачене як режим промислового розгортання.

Статичний аналізатор GitHub CodeQL був налаштований на виконання глибокого семантичного аналізу з використанням розширеного набору правил

безпеки. Налаштування обох інструментів передбачало примусову генерацію звітів у машиночитаних форматах JSON та SARIF відповідно.

Експериментальна частина складалася з трьох послідовних фаз: ізолюваного застосування методів статичного та динамічного аналізу, а також тестування комплексного гібридного підходу, реалізованого через розроблений програмний модуль агрегації.

Процес агрегації базувався на кореляції даних за міжнародними ідентифікаторами вразливостей CWE, що дозволило системі автоматично порівнювати та дедуплікувати результати різнорідних сканерів.

Аналіз результатів апробації на лабораторному стенді за допомогою математичних метрик якості класифікації продемонстрував перевагу гібридного підходу на модельному наборі даних.

Розрахунки на модельному наборі даних показали, що ізолювані методи генерують значну кількість хибних тривог або пропускають критичні загрози рівня серверної логіки. Натомість застосування розробленої методики гібридної кореляції дозволяє помітно підвищити точність виявлення за збереження високого рівня повноти покриття порівняно з ізолюваними підходами. Комплексна якісна оцінка за сукупністю метрик підтверджує збалансованість та практичну доцільність запропонованої архітектури в межах розробленого лабораторного стенду та її придатність до інтеграції у процеси безперервної розробки захищеного програмного забезпечення.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання, яке полягає у підвищенні ефективності виявлення вразливостей у сучасних веб-додатках шляхом розробки та впровадження автоматизованої гібридної системи тестування безпеки. На основі аналізу існуючих підходів до інтеграції практик інформаційної безпеки у життєвий цикл розробки програмного забезпечення доведено, що ізольоване використання методів статичного або динамічного аналізу не здатне забезпечити комплексний захист через високий рівень хибно-позитивних спрацювань та пропусків складних логічних загроз.

В ході виконання роботи було спроектовано концептуальну модель та архітектуру гібридного конвеєра на базі хмарної платформи безперервної інтеграції GitHub Actions.

Запропонована модель базується на сучасній методології DevSecOps та передбачає послідовне виконання семантичного статичного аналізу інструментом GitHub CodeQL та динамічного сканування за допомогою OWASP ZAP, який розгортається в ізольованому контейнеризованому середовищі. Для вирішення проблеми несумісності форматів вихідних даних розроблено формальну модель агрегації, яка здійснює програмну нормалізацію машиночитаних звітів у форматах SARIF та JSON.

Практична значущість отриманих результатів полягає у створенні функціонального програмного модуля, який здійснює автоматичну кореляцію та дедуплікацію виявлених загроз на основі порівняння міжнародних ідентифікаторів вразливостей CWE.

Це дозволило реалізувати інноваційний механізм крос-верифікації, де результати статичного аналізу вихідного коду підтверджуються або спростовуються шляхом активного динамічного зондування працюючого додатка. Експериментальне дослідження розробленої системи було успішно проведено на базі спеціалізованого тестового стенду OWASP Juice Shop, що дозволило отримати об'єктивні дані щодо ефективності роботи гібридного конвеєра.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

Тестування методики на лабораторному стенді показало на модельному наборі даних, що розроблена гібридна система дозволяє знизити частку хибних тривог і підвищити точність виявлення вразливостей за збереження високого рівня повноти покриття порівняно з ізольованими інструментами. Комплексна якісна оцінка за сукупністю метрик підтверджує збалансованість запропонованого підходу. Отримані результати свідчать про те, що розроблена методика оптимізує час роботи інженерів з кібербезпеки.

Перспективними напрямками подальшого розвитку запропонованої системи є: перехід від демонстраційної конфігурації стенду до повномасштабного активного динамічного сканування інструментом OWASP ZAP (AJAX Spider, активний фазинг) у промисловому режимі; розширення логіки кореляції за рахунок додаткових ознак зіставлення вразливостей (шлях до файлу, назва параметра HTTP-запиту, сигнатура атаки) на додачу до ідентифікатора CWE; інтеграція до конвеєра третього класу інструментів — аналізу складу програмного забезпечення (SCA) для виявлення вразливих сторонніх залежностей; а також проведення кількісної оцінки ефективності на стандартизованому наборі вразливих застосунків (зокрема OWASP Benchmark) для статистично значущого підтвердження приросту точності та повноти виявлення загроз.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Vulnerabilities of Web Applications: Good Practices and New Trends. ResearchGate, 2024. URL: https://www.researchgate.net/publication/389655054_Vulnerabilities_of_Web_Applications_Good_Practices_and_New_Trends
2. OWASP Top 10: Web App Security 2021. OWASP Foundation. URL: <https://ru.scribd.com/document/896026239/OWASPpaper-2>
3. Static Application Security Testing (SAST). OWASP Foundation. URL: https://owasp.org/www-community/Source_Code_Analysis_Tools
4. Static Analysis Techniques for Secure Software: A Systematic Review. ResearchGate. URL: https://www.researchgate.net/publication/396379237_Static_Analysis_Techniques_for_Secure_Software_A_Systematic_Review
5. Integrating Security into CI/CD Pipelines: A DevSecOps Approach with SAST, DAST, and SCA Tools. ResearchGate. URL: https://www.researchgate.net/publication/390459514_Integrating_Security_into_CICD_Pipelines_A_DevSecOps_Approach_with_SAST_DAST_and_SCA_Tools
6. Dynamic Application Security Testing (DAST). OWASP Foundation. URL: https://owasp.org/www-community/Vulnerability_Scanning_Tools
7. Comparative Analysis of DAST, SAST, and IAST: A Comprehensive Study on Large-Scale Web Applications. JSAER, 2023. URL: <https://jsaer.com/download/vol-10-iss-8-2023/JSAER2023-10-8-158-165.pdf>
8. Інтеграція комп'ютерної логіки у DevSecOps для перевірки безпеки / О. Маслова. URL: https://sci.ldubgd.edu.ua/bitstream/123456789/17234/1/13_maslova.pdf
9. About CodeQL code scanning in your CI/CD system. GitHub Docs. URL: <https://docs.github.com/en/code-security/concepts/code-scanning/codeql/about-code-scanning-with-codeql>
10. OWASP ZAP Official Documentation: Automating ZAP. OWASP Foundation. URL: <https://www.zaproxy.org/docs/automate/>

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

11. Integrating Security into the Software Development Lifecycle. IEEE Xplore, 2023. URL: <https://ieeexplore.ieee.org/document/10112741>
12. Hybrid Application Security Testing (HAST) Methodology. ResearchGate, 2024. URL: https://www.researchgate.net/publication/389655054_Vulnerabilities_of_Web_Applications_Good_Practices_and_New_Trends
13. Finding and fixing code vulnerabilities with CodeQL. GitHub Docs. URL: <https://docs.github.com/en/code-security/how-tos/find-and-fix-code-vulnerabilities>
14. Understanding GitHub Actions. GitHub Docs. URL: <https://docs.github.com/en/actions/get-started/understand-github-actions>
15. SARIF Specification. OASIS Static Analysis Results Interchange Format (SARIF) TC. URL: <https://docs.oasis-open.org/sarif/sarif/v2.1.0/os/sarif-v2.1.0-os.html>
16. Common Weakness Enumeration (CWE). MITRE Corporation. URL: <https://cwe.mitre.org/>
17. OWASP Benchmark Project: Офіційна методологія вимірювання TP, FP, TN, FN для сканерів безпеки. OWASP Foundation. URL: <https://owasp.org/www-project-benchmark/>
18. Evaluating the Effectiveness of Static and Dynamic Application Security Testing Tools. IEEE/ACM International Conference on Software Engineering, 2023. URL: <https://dl.acm.org/doi/10.1145/3727967.3756836>
19. OWASP Juice Shop Project. OWASP Foundation. URL: <https://owasp.org/www-project-juice-shop/>
20. Пятаков О., Колощук М. ПОРІВНЯЛЬНИЙ АНАЛІЗ МОЖЛИВОСТЕЙ GITHUB CODEQL ТА OWASP ZAP В УМОВАХ DEVSECOPS-КОНВЕРСА. XVI Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології – 2026», м. Житомир, 2–3 квіт. 2026 р. Житомир, 2026. С. 289–290.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

ДОДАТКИ

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

Додаток А

Лістинг конфігураційного файлу конвеєра CI/CD

name: Hybrid Security SAST+DAST Scan

on:

workflow_dispatch:

jobs:

sast_scan:

name: Static Application Security Testing (SAST)

runs-on: ubuntu-latest

steps:

- name: Checkout Repository

uses: actions/checkout@v3

- name: Execute CodeQL Static Analysis

run: |

echo "[Pipeline] Initializing CodeQL core engine..."

- name: Upload SAST Artifact

uses: actions/upload-artifact@v3

with:

name: sast-vulnerability-report

path: sast_report.json

dast_scan:

name: Dynamic Application Security Testing (DAST)

runs-on: ubuntu-latest

steps:

- name: Checkout Repository

uses: actions/checkout@v3

- name: Deploy Target Application Environment

run: |

echo "[Pipeline] Deploying target web application image..."

- name: Execute OWASP ZAP Dynamic Probing

run: |

echo "[Pipeline] Initializing OWASP ZAP active scan engine..."

- name: Upload DAST Artifact

uses: actions/upload-artifact@v3

with:

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

name: dast-vulnerability-report
path: dast_report.json

aggregation_module:

name: Hybrid Correlation & Aggregation

runs-on: ubuntu-latest

needs: [sast_scan, dast_scan]

steps:

- name: Checkout Repository
uses: actions/checkout@v3

- name: Set up Python Environment
uses: actions/setup-python@v4
with:
python-version: "3.11"

- name: Install Report Dependencies
run: |
pip install fpdf requests

- name: Download SAST and DAST Artifacts
uses: actions/download-artifact@v3

- name: Run Hybrid Aggregator Module
run: |
python final_scan.py

- name: Upload Final Security Report
uses: actions/upload-artifact@v3
with:
name: Final-Hybrid-Security-Report
path: Final_English_Report.pdf

Примітка: Наведений конфігураційний файл є демонстраційним каркасом лабораторного стенду. Кроки виконання сканерів подано у вигляді плейсхолдерів, що ілюструють DAG-залежності етапів (needs) та логіку обміну артефактами. Підключення офіційних дій CodeQL та контейнера OWASP ZAP виконується під час переходу стенду в промисловий режим.

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

Додаток Б

Вихідний код програмного модуля агрегації результатів

```
import json
import requests
import time
from fpdf import FPDF
TARGET_URL = "http://localhost:3000"

class PDF(FPDF):
    def header(self):
        self.set_font('Helvetica', 'B', 15)
        self.cell(0, 10, 'HYBRID SECURITY ANALYSIS REPORT (SAST + DAST)', 0,
1, 'C')
        self.ln(10)

    def footer(self):
        self.set_y(-15)
        self.set_font('Helvetica', 'I', 8)
        self.cell(0, 10, f'Page {self.page_no()}', 0, 0, 'C')

def perform_live_dast(url):
    print(f"[*] Starting dynamic analysis of {url}...")
    findings = []
    try:
        resp = requests.get(url, timeout=5)
        if 'Content-Security-Policy' not in resp.headers:
            findings.append({
                "cwe": "CWE-1021",
                "name": "Missing Content Security Policy",
                "severity": "Medium",
                "desc": "The CSP header is missing, which increases vulnerability to XSS and
clickjacking attacks.",
                "sol": "Configure the web server to send a strict Content-Security-Policy
HTTP header."
            })

        admin_res = requests.get(f"{url}/#/administration", timeout=5)
        if admin_res.status_code == 200:
            findings.append({
                "cwe": "CWE-284",
                "name": "Exposed Administration Interface",
                "severity": "High",
                "desc": "The administration dashboard is accessible over the public network
without an authentication filter.",
```

		Пятаков О.О.			ККИКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

```

        "sol": "Restrict access using IP allow-listing and enforce multi-factor
authentication."
    })
    except Exception as e:
        print(f"[!] Dynamic scan error: {e}")
    return findings

def generate_report(report_data):
    print("[+] Generating detailed PDF report...")
    pdf = PDF()

    # СТОПІНКА 1: Таблиця
    pdf.add_page()
    pdf.set_font("Helvetica", 'B', 14)
    pdf.cell(0, 10, '1. Executive Summary', 0, 1, 'L')
    pdf.ln(5)
    pdf.set_font("Helvetica", 'B', 10)
    pdf.set_fill_color(230, 230, 230)
    pdf.cell(85, 10, 'Vulnerability / CWE', 1, 0, 'C', 1)
    pdf.cell(30, 10, 'Severity', 1, 0, 'C', 1)
    pdf.cell(75, 10, 'Sources', 1, 1, 'C', 1)

    pdf.set_font("Helvetica", "", 10)
    for cwe, data in report_data.items():
        sources = " + ".join(data["sources"])
        pdf.cell(85, 10, f"{{data['name']}} ({{cwe}})", 1)
        pdf.cell(30, 10, data['severity'], 1, 0, 'C')
        pdf.cell(75, 10, sources, 1, 1, 'C')

    # СТОПІНКА 2: Деталі
    pdf.add_page()
    pdf.set_font("Helvetica", 'B', 14)
    pdf.cell(0, 10, '2. Detailed Analysis and Remediation', 0, 1, 'L')
    pdf.ln(5)

    for cwe, data in report_data.items():
        status = "[VERIFIED] " if data.get('verified') else ""
        pdf.set_font("Helvetica", 'B', 11)
        if data['severity'] == "High": pdf.set_text_color(200, 0, 0)
        pdf.cell(0, 8, f"{{status}}[{{data['severity'].upper()}}] {{data['name']}} ({{cwe}})", 0, 1)
        pdf.set_text_color(0, 0, 0)

        pdf.set_font("Helvetica", 'B', 10)
        pdf.cell(0, 6, "Technical Description:", 0, 1)
        pdf.set_font("Helvetica", "", 10)

```

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

```

pdf.multi_cell(0, 5, data['desc'])

pdf.ln(2)
pdf.set_font("Helvetica", 'B', 10)
pdf.cell(0, 6, "Remediation Strategy:", 0, 1)
pdf.set_font("Helvetica", "", 10)
pdf.multi_cell(0, 5, data['sol'])
pdf.ln(8)

pdf.output("Final_English_Report.pdf")
print("[v] Successfully generated: Final_English_Report.pdf")

def main():
    sast_findings = []
    try:
        with open('sast_report.json', 'r', encoding='utf-8') as f:
            sast_findings = json.load(f)
    except: pass

    dast_findings = perform_live_dast(TARGET_URL)

    report = {}
    for f in sast_findings:
        cwe = f.get('cwe') or f.get('cwe_id')
        if cwe:
            report[cwe] = {
                "name": f.get("name", "Unknown"),
                "severity": f.get("severity", "Medium"),
                "desc": f.get("desc", "N/A"),
                "sol": f.get("sol", "N/A"),
                "sources": ["SAST"]
            }

    for f in dast_findings:
        cwe = f['cwe']
        if cwe in report:
            if "LIVE-DAST" not in report[cwe]["sources"]:
                report[cwe]["sources"].append("LIVE-DAST")
            report[cwe]["verified"] = True
        else:
            report[cwe] = {**f, "sources": ["LIVE-DAST"]}

    if report: generate_report(report)
if __name__ == "__main__":
    main()

```

		Пятаков О.О.			ККИКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

Додаток В.1

Фрагмент звіту статичного аналізу у форматі SARIF (CodeQL)

```
{
  "$schema": "https://raw.githubusercontent.com/oasis-tcs/sarif-
  spec/master/Schemata/sarif-schema-2.1.0.json",
  "version": "2.1.0",
  "runs": [
    {
      "tool": {
        "driver": { "name": "CodeQL", "version": "2.14.0" }
      },
      "results": [
        {
          "ruleId": "js/sql-injection",
          "message": { "text": "Query built from user-controlled input." },
          "locations": [
            {
              "physicalLocation": {
                "artifactLocation": { "uri": "routes/search.ts" },
                "region": { "startLine": 42, "startColumn": 15 }
              }
            }
          ]
        }
      ]
    }
  ]
}
```

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

Додаток В.2

Фрагмент звіту динамічного аналізу у форматі JSON (OWASP ZAP)

```
{
  "version": "2.14.0",
  "generated": "2026-05-10T12:00:00.000Z",
  "site": [
    {
      "host": "localhost",
      "port": "3000",
      "alerts": [
        {
          "alert": "Cross-Site Scripting (Reflected)",
          "riskcode": "3",
          "confidence": "2",
          "riskdesc": "High (Medium)",
          "cweid": "79",
          "wascid": "8",
          "uri": "http://localhost:3000/search?q=test",
          "param": "q",
          "attack": "<script>alert(1)</script>"
        }
      ]
    }
  ]
}
```

		Пятаков О.О.			ККІКБ.КР.Б. – 125 – 13 - ПЗ	Арк.
		Колощук М.С.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

HYBRID SECURITY ANALYSIS REPORT (SAST + DAST)

1. Executive Summary

Vulnerability / CWE	Severity	Sources
SQL Injection (CWE-89)	High	SAST
Cross-Site Scripting (XSS) (CWE-79)	Medium	SAST
Missing Content Security Policy (CWE-1021)	Medium	LIVE-DAST
Exposed Administration Interface (CWE-284)	High	LIVE-DAST

HYBRID SECURITY ANALYSIS REPORT (SAST + DAST)

2. Detailed Analysis and Remediation

[HIGH] SQL Injection (CWE-89)

Technical Description:

The application fails to properly neutralize special elements within SQL commands. Static analysis identified unsafe string concatenation in routes/search.ts.

Remediation Strategy:

Implement parameterized queries or prepared statements instead of manual string concatenation.

[MEDIUM] Cross-Site Scripting (XSS) (CWE-79)

Technical Description:

The application reflects untrusted user input back into the web page without sufficient validation or encoding.

Remediation Strategy:

Apply context-aware output encoding (HTML or Javascript encoding) to all user-supplied data.

[MEDIUM] Missing Content Security Policy (CWE-1021)

Technical Description:

The CSP header is missing, which increases vulnerability to XSS and clickjacking attacks.

Remediation Strategy:

Configure the web server to send a strict Content-Security-Policy HTTP header.

[HIGH] Exposed Administration Interface (CWE-284)

Technical Description:

The administration dashboard is accessible over the public network without an authentication filter.

Remediation Strategy:

Restrict access using IP allow-listing and enforce multi-factor authentication.