ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА СИСТЕМ ТА ТЕХНОЛОГІЙ КІБЕРБЕЗПЕКИ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Технологія статичного аналізу коду для виявлення вразливостей у вебзастосунках на базі рішення Snyk Code»

зі спеціальності *125 Кібербезпека та захист інформації*
(код, найменування спеціальності)

освітньо-професійної програми *Інформаційна та кібернетична безпека*
(назва програми)

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

_____ Данило КРИВЕЦЬ
(підпис)

Виконав: здобувач вищої освіти групи БСДМ-61
КРИВЕЦЬ Данило
(прізвище, ім'я)

Керівник д.ф. доцент, МАРЧЕНКО Віталій
(науковий ступінь, вчене звання, прізвище, ім'я)

Рецензент _____
(науковий ступінь, вчене звання, прізвище, ім'я)

Київ 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

AST – Application Security Testing

CI/CD – Continuous Integration / Continuous Deployment

CWE – Common Weakness Enumeration

DAST – Dynamic Application Security Testing

DevSecOps – Development, Security, and Operations

IDE – Integrated Development Environment

OWASP – Open Web Application Security Project

SAST – Static Application Security Testing

SDLC – Software Development Life Cycle

SSA – Semantic Static Analysis

SCA – Software Composition Analysis

SSRF – Server-Side Request Forgery

SQLi – SQL Injection

XSS – Cross-Site Scripting

ВСТУП

Актуальність дослідження. Швидкий розвиток веб-технологій та широке впровадження вебзастосунків як основних компонентів сучасних інформаційних систем значно підвищили важливість безпеки додатків. Сьогодні вебзастосунки обробляють великі обсяги конфіденційних даних і підтримують критично важливі бізнес-процеси, при цьому розробляються та впроваджуються високими темпами з використанням гнучких методів та методів безперервної доставки. За таких умов традиційні підходи до безпеки програмного забезпечення, які зосереджуються переважно на тестуванні на пізніх етапах розробки або після впровадження, вже не є достатніми.

Оскільки життєвий цикл розробки програмного забезпечення стає коротшим і складнішим, вразливості безпеки, що виникають на рівні вихідного коду, часто залишаються невиявленими до моменту їх використання. Як наслідок, організації все частіше переходять до проактивних моделей безпеки, що наголошують на ранньому виявленні та запобіганні вразливостей під час процесу розробки. Технологія статичного тестування безпеки додатків (SAST) дозволяє виявляти слабкі місця в системі безпеки безпосередньо в вихідному коді без запуску додатка. Такий підхід дає змогу інтегрувати засоби контролю безпеки на ранніх етапах життєвого циклу розробки програмного забезпечення та підтримує принципи безпечного проектування (secure-by-design) і DevSecOps.

Однак ефективність SAST залежить від можливостей обраного рішення, його точності та здатності безперешкодно інтегруватися в робочі процеси розробки.

Snyk Code – це сучасне рішення для статичного аналізу коду, яке застосовує методи семантичного аналізу для виявлення вразливостей у вебзастосунках і надає розробникам контекстні рекомендації щодо усунення недоліків.

Вищезазначені міркування визначають актуальність цієї кваліфікаційної роботи, яка зосереджена на вивченні технології статичного аналізу коду для виявлення вразливостей у вебзастосунках на основі рішення Snyk Code.

Об'єкт дослідження – безпека вебзастосунків під час життєвого циклу розробки програмного забезпечення.

Предмет дослідження – технологія статичного аналізу коду для виявлення вразливостей у вебзастосунках на основі рішення Snyk Code.

Мета роботи – аналіз технології статичного тестування безпеки вебзастосунків та розробка рекомендацій щодо використання Snyk Code для виявлення та запобігання вразливостей безпеки на ранніх етапах розробки програмного забезпечення.

Наукові завдання:

дослідити сутність проблеми забезпечення безпечної розробки вебзастосунків

проаналізувати підходи до забезпечення безпечної розробки вебзастосунків

проаналізувати існуючі рішення із забезпечення безпечної розробки вебзастосунків

проаналізувати методи та засоби забезпечення безпечної розробки вебзастосунків на базі SnykCode.

розкрити порядок реалізації технології забезпечення безпечної розробки вебзастосунків

Методи дослідження – опрацювання літератури за даною темою, аналіз експлуатаційної документації, міжнародних стандартів та їх порівняння.

Практичне значення одержаних результатів: запропоновано порядок застосування технології забезпечення безпечної розробки вебзастосунків, а також розроблено рекомендації фахівцям з кібербезпеки щодо її реалізації.

Результати кваліфікаційної роботи апробовані на Всеукраїнській науковій конференції «Актуальні проблеми кібербезпеки», яка відбулася 29 жовтня 2025 року в Державному університеті інформаційно-комунікаційних технологій, м. Київ.

1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ У ВЕБЗАСТОСУНКАХ

1.1 Оцінка ризиків, пов’язаних із використанням уразливого коду

1.1.1 Поняття вебзастосунку

Вебзастосунок - це програмне забезпечення, доступ до якого здійснюється та яке виконується через веб-браузер, використовуючи мережеве з'єднання, а не встановлюючись на локальному пристрої користувача.

Вебзастосунок є вигідною формою програмного забезпечення, оскільки використання браузерів дозволяє додатку бути сумісним з більшістю стандартних комп'ютерів та операційних систем [1]. Більше того, додаток не займає пам'ять на жорсткому диску комп'ютера і доступний майже з будь-якого комп'ютера або пристрою, який може використовувати людина.

Згідно звіту Facts and Factors [2], вебзастосунки, будучи легкими у розгортанні та широко використовуваними серед користувачів по всьому світу, очікують середньорічнього зростання ринку близько 34% у період з 2020 по 2026 рік (рисунок 1.1).

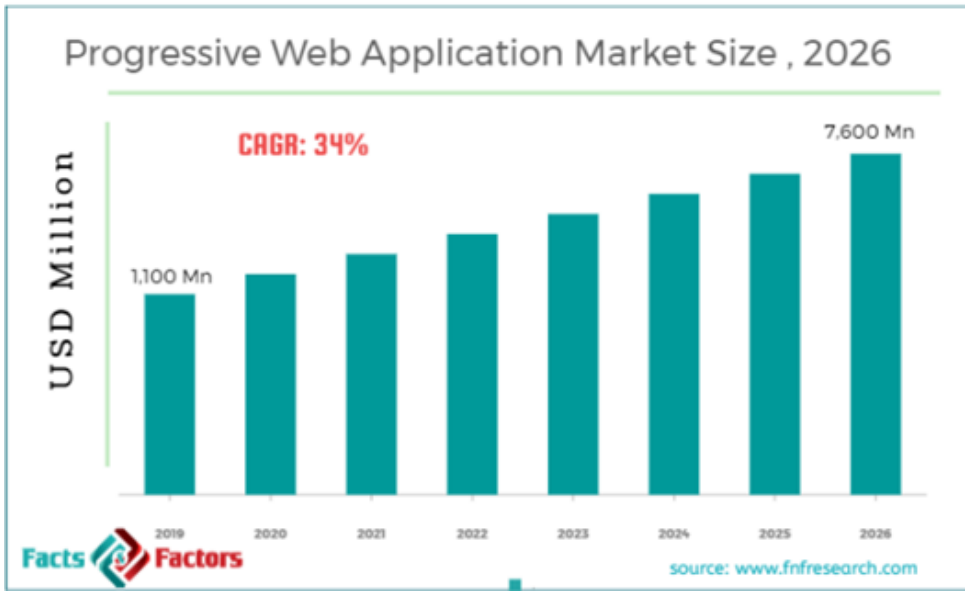

Рис. 1.1. Прогнозоване зростання вебзастосунків на IT ринку на 2026 рік[2]

Вебзастосунки побудовані за архітектурою «клієнт-сервер». Їхній код зазвичай поділяється на два основні компоненти: скрипти на стороні клієнта (виконуються в браузері) та скрипти на стороні сервера (виконуються на веб-сервері).

Наприклад, коли користувач відвідує вебзастосунок, браузер завантажує та виконує код на стороні клієнта (написаний на HTML/CSS/JavaScript) для відображення інтерфейсу та реагування на дії користувача. Коли користувач виконує дію, що вимагає даних (наприклад, надсилає форму або запитує додаткову інформацію), браузер надсилає запит на сервер. Потім серверний компонент обробляє запит (наприклад, запитує базу даних або виконує бізнес-логіку) і повертає результати браузеру [3].

Однією з ключових характеристик вебзастосунків є доступність. Оскільки вони працюють у браузерах, вони сумісні з різними операційними системами та пристроями, що мають підключення до Інтернету. Вони також не потребують ручної інсталяції на клієнті; оновлення та виправлення безпеки відбуваються на сервері. З цієї причини підприємства використовують вебзастосунки для безпечного та зручного надання послуг користувачам у будь-якому місці.

1.1.2 Типи вебзастосунків

Вебзастосунки бувають різних видів, що відображає відмінності в архітектурі та призначенні. Відповідна більшість вебзастосунків в інтернеті є статичними або динамічними вебзастосунками, також варто виділити і багато сторінковий та односторінковий варіант. Веб додатки також можуть бути розділені за їх функціоналом. Відповідно до сучасних політик та/або стандартів прийнято виділяти 11 основних типів:

1. Статичні вебзастосунки [4]

Це найпростіші типи вебзастосунків, створені за допомогою HTML і CSS, які підходять для створення портфоліо або цифрових брошур. Як випливає з назви, вміст змінюється тільки в разі ручного оновлення розробником.

Статичні вебзастосунки прості у створенні та розміщенні, оскільки не вимагають значної обробки на стороні сервера. Цей варіант є економічно вигідним

рішенням для приватних осіб або малих підприємств, яким потрібна проста присутність в Інтернеті. Однак їх простота також означає обмежену функціональність.

2. Динамічні вебзастосунки [4]

На відміну від цього, динамічні вебзастосунки є більш складними та інтерактивними. Вони використовують скрипти на стороні клієнта та сервера (такі як JavaScript, PHP, ASP або JSP) для генерації контенту в режимі реального часу. Ці категорії вебзастосунків підключені до бази даних, що дозволяє їм надавати персоналізовані можливості на основі взаємодій та уподобань користувачів.

Вони ідеально підходять для бізнесу, особливо якщо головними пріоритетами є залучення користувачів та різноманітність контенту. Через свою комплексність динамічні вебзастосунки складніші в розробці та підтримці. Вони вимагають більш надійного хостингового середовища та вищих витрат на веб-розробку.

3. Односторінкові додатки (SPA) [4]

SPA завантажують одну HTML-сторінку і динамічно оновлюють вміст у міру взаємодії користувачів з додатком. Ці категорії вебзастосунків ідеально підходять для платформ, де користувацький досвід і швидкість мають вирішальне значення, таких як:

- Соціальні мережі;
- Поштові клієнти;
- Хмарне програмне забезпечення.

Перевага цього типу вебзастосунків полягає в тому, що вони не перезавантажують всю сторінку при кожній дії користувача, що забезпечує більш плавний і швидкий користувацький досвід. Однак вони також мають свої недоліки, зокрема в оптимізації SEO та часі початкового завантаження, оскільки весь додаток повинен завантажуватися одночасно.

SPA створюються за допомогою фреймворків JavaScript, таких як Angular, React або Vue.js, які обробляють динамічне завантаження вмісту та елементів користувацького інтерфейсу.

4. Багатосторінкові вебзастосунки (MPA)

На відміну від SPA, MPA перезавантажують всю сторінку з сервера, коли користувач взаємодіє з додатком. Ці традиційні категорії вебзастосунків більше підходять для веб-сайтів з великим обсягом контенту та різноманітними функціональними можливостями, такими як:

- Сайти електронної комерції;
- Онлайн-каталоги;
- Освітні платформи.

MPA можуть обробляти складні структури та великі бази даних ефективніше, ніж SPA. Вони також краще оптимізовані для пошукових систем, оскільки кожна сторінка може індексуватися окремо. Однак MPA часто мають повільніші переходи між сторінками і можуть бути більш ресурсоємними, оскільки кожна нова сторінка потребує запиту до сервера і перезавантаження сторінки. Розробка MPA зазвичай передбачає більш масштабний процес бек-енду для управління декількома сторінками та їх взаємодією з сервером.

5. Прогресивні вебзастосунки (PWA)

PWA є гібридом звичайних веб-сторінок (або веб-сайтів) та мобільних додатків. Їх можна встановити на головний екран пристрою без завантаження з магазину додатків. Ключовою особливістю PWA є використання сервісних працівників або скриптів, що працюють у фоновому режимі, що забезпечує:

- Офлайн-функціональність;
- Push-повідомлення;
- Синхронізацію даних у фоновому режимі.

PWA також є адаптивними та можна на них посилатися, ділитися ними через URL-адресу. Вони забезпечують високий рівень продуктивності, залучаючи користувачів плавними анімаціями та без затримок при прокручуванні.

Таким чином, ці категорії вебзастосунків є надзвичайно ефективними, особливо для користувачів з обмеженим доступом до Інтернету. Розробники використовують для їх створення стандартні веб-технології, включаючи HTML, CSS та JavaScript.

6. Системи управління контентом (CMS)

CMS управляє створенням та модифікацією цифрового контенту, підтримуючи роботу декількох користувачів у спільному середовищі. Функції CMS можуть бути дуже різними, включаючи, але не обмежуючись:

- Веб-публікація;
- Управління форматами;
- Редагування історії;
- Контроль версій;
- Індексація;
- Пошук.

Вони підходять для блогів, електронної комерції та новинних веб-сайтів, де потрібно часто оновлювати контент без глибоких технічних знань. Популярними є такі CMS-платформи, як WordPress, Joomla та Drupal, які пропонують шаблони та плагіни для налаштування без необхідності писати код з нуля. Більше того, CMS мають зручний інтерфейс, що дозволяє легко оновлювати та керувати контентом.

7. Вебзастосунки для електронної комерції [4]

Ці типи вебзастосунків полегшують купівлю та продаж в Інтернеті. Це складні системи, що інтегрують різні функції, зокрема:

- Відображення або каталоги товарів;
- Пошук та фільтрування товарів;
- Кошики для покупок;
- Обробка платежів;
- Управління обліковими записами клієнтів та замовленнями;
- Інструменти обслуговування клієнтів.

Вебзастосунки для електронної комерції повинні забезпечувати безперебійну та зручну роботу, щоб стимулювати продажі та повторні покупки. Ці додатки повинні бути масштабованими, щоб обробляти різні рівні трафіку та обсяги продажів.

Крім того, безпека має першочергове значення для захисту конфіденційних даних клієнтів, включаючи платіжну інформацію. Такі платформи, як Shopify,

Magento та WooCommerce, є відомими прикладами, що пропонують настроюванні шаблони та різні плагіни для розширення функціональних можливостей. Вебзастосунки для електронної комерції революціонізували галузь роздрібної торгівлі, дозволивши підприємствам охопити ширшу аудиторію та працювати цілодобово.

8. Вебзастосунки на базі JavaScript

JavaScript - це універсальна мова програмування, яка використовується для створення динамічних та інтерактивних вебзастосунків. Згідно з доповіддю Statista під назвою «Найпопулярніші мови програмування серед розробників у всьому світі в 2024 році», опублікованою 6 лютого 2025 року Ліонелем Суджаєм Вейлшери, JavaScript був найпопулярнішою мовою програмування в 2023 році.

JavaScript забезпечує оновлення в режимі реального часу, плавні анімації та покращену взаємодію з користувачами, що робить його незамінним для розробки сучасних вебзастосунків.

JavaScript можна використовувати як на стороні клієнта (у браузері), так і на стороні сервера (за допомогою таких технологій, як Node.js), що робить його потужним інструментом для повноцінної розробки [4]. Вебзастосунки на базі JavaScript відомі своєю швидкістю та ефективністю; вони можуть оновлювати вміст без перезавантаження всієї сторінки. Ця функціональність робить їх ідеальними для додатків, які вимагають оновлення даних у реальному часі, таких як:

- Платформи соціальних медіа;
- Онлайн-ігри;
- Інструменти для спільної роботи.

9. RIA

RIA - це просунуті типи вебзастосунків, які забезпечують користувачам досвід, схожий на роботу з настільними додатками. Вони використовують клієнтські фреймворки для забезпечення інтерактивних функцій і більш багатого

користувацького інтерфейсу, наприклад:

- Adobe Flash;
- JavaFX;
- Microsoft Silverlight.

RIA працюють у веб-браузерах, але поводяться як настільні додатки, пропонуючи чуйний, привабливий користувацький досвід з кращою візуалізацією даних і можливостями взаємодії в режимі реального часу.

RIA можуть обробляти дані і виконувати завдання без постійного зв'язку з сервером, скорочуючи час завантаження і покращуючи продуктивність. Однак вони вимагають плагінів або спеціальних фреймворків, що може обмежувати доступність і сумісність на різних пристроях і в різних браузерах.

10. Вебзастосунки категорії портал

Вебзастосунки порталу є шлюзами до різної інформації, послуг та інших додатків. Підприємства часто використовують їх для внутрішніх цілей або для надання послуг клієнтам.

Ці типи вебзастосунків зазвичай вимагають автентифікації користувача, пропонуючи персоналізований контент та централізовану точку доступу до різних ресурсів. Вебзастосунки порталу призначені для агрегації контенту з різних джерел, забезпечуючи послідовний та інтегрований користувацький досвід. Вони можуть обробляти різні функції, включаючи:

- Пошукові системи;
- Системи електронної пошти;
- Форуми;
- Новини.

11. Анімовані вебзастосунки

Ці додатки орієнтовані на надання багатого візуального контенту та інтерактивних елементів за допомогою анімації. Вони особливо популярні в

сферах, що вимагають високого рівня залучення користувачів, таких як:

- Інтернет-реклама;
- Ігри;
- Освітні платформи.

Анімовані вебзастосунки створюються з використанням таких технологій, як CSS3, HTML5 та WebGL, що дозволяє розробникам створювати складні, але привабливі для користувачів інтерфейси. Вони забезпечують динамічний та візуально привабливий досвід, привертаючи увагу користувачів та покращуючи взаємодію.

1.2 Ідентифікація типових вразливостей у вебзастосунках згідно OWASP Top 10 (CWE)

Вебзастосунки зазвичай страждають від низки слабких місць у безпеці, виявлених такими фреймворками, як OWASP Top 10 та Common Weakness Enumeration (CWE). OWASP Top 10 (2021) виділяє такі критичні категорії, як ін'єкція, порушення контролю доступу та криптографічні збої, тоді як список CWE Top 25 акцентує увагу на поширених слабких місцях програмного забезпечення, таких як міжсайтовий скриптинг (CWE-79) та SQL-ін'єкція (CWE-89) [5]. Ці вразливості зазвичай виникають через неякісне кодування, відсутність перевірок або незахищені конфігурації.

Більшість пентестерів та розробників акцентують свою увагу на OWASP через її практичність та якісно описані причини виникнення вразливості та методи усунення. OWASP є некомерційною організацією з безпеки програмного забезпечення яка проводить дослідження аналіз ринку та сучасні вимоги вебзастосунків, та оновлює список найпопулярніших уразливостей вебзастосунків [6]. Останьою на сьогодні версією є версія 2021 року (рисунок 1.2).

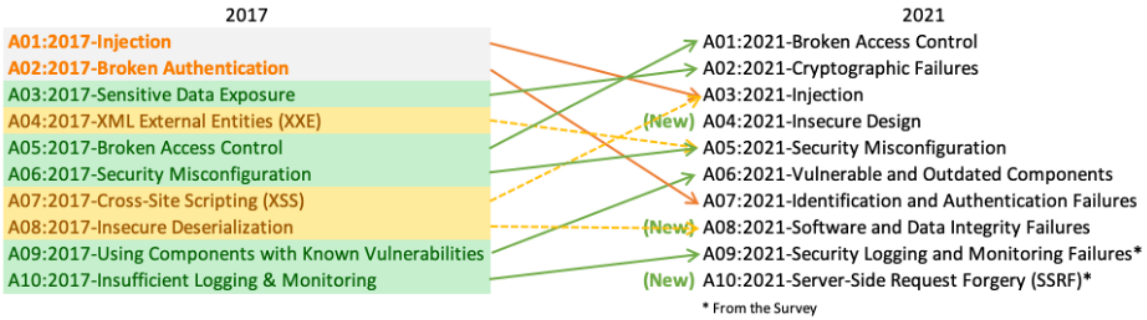

Рис. 1.2. Порівняння змін списків 2017 та 2021 року [6]

Далі ми розглянемо поточний перелік більш детально для аналізу та розуміння причини виявлення вразливостей в вебзастосунках.

Порушення контролю доступу (A01:2021)

Порушення контролю доступу відбувається, коли програма не може забезпечити виконання відповідних дій відповідними особами. Небезпечні або відсутні перевірки авторизації дозволяють користувачам виконувати дії, що виходять за межі їхніх повноважень. До поширених причин належать покладання на перевірки на стороні клієнта, ігнорування принципів «заборона за замовчуванням» та невдача у перевірці ролей користувачів для кожної функції.

Наприклад, якщо кінцеві точки URL або посилання на об'єкти не захищені, зловмисник може змінити параметр або примусово переглянути сторінку, щоб отримати несанкціонований доступ.

Криптографічні збої (A02:2021)

Криптографічні збої (раніше «розкриття конфіденційних даних») виникають, коли програми використовують слабке шифрування або взагалі не шифрують конфіденційні дані під час передачі або зберігання. Причини включають використання застарілих алгоритмів (наприклад, MD5, SHA-1), неналежне управління ключами або відсутність TLS у комунікаціях. Наприклад, якщо конфіденційні поля (паролі, PII, кредитні картки) зберігаються або передаються без належного шифрування або якщо використовуються власні/слабкі шифри, зловмисники можуть перехопити або розшифрувати дані.

Ін'єкція (A03:2021)

Помилки ін'єкції виникають, коли для побудови команд або запитів

використовуються ненадійні вхідні дані, внаслідок чого зловмисні вхідні дані змінюють передбачувану операцію. Поширеними формами є ін'єкція SQL, NoSQL, команд ОС, LDAP та скриптів. Вони виникають, якщо дані користувача не перевіряються або не очищаються і об'єднуються в інтерпретатори або запити до бази даних. Наприклад, побудова SQL-запиту шляхом об'єднання рядків з параметрами користувача (наприклад, «SELECT * FROM users WHERE id=" + userInput») дозволяє зловмиснику ін'єктувати синтаксис SQL (наприклад, “ OR '1”='1') [6].

Небезпечний дизайн (A04:2021)

Небезпечний дизайн це архітектурні або конструктивні недоліки, які створюють вразливі місця через те, що не були заплановані необхідні заходи безпеки. На відміну від помилок кодування, це недоліки в структурі програми або бізнес-логіці. Причини включають відсутність моделювання загроз, відсутність вимог безпеки або ігнорування крайніх випадків під час проєктування. Наприклад, якщо дизайн програми ніколи не включав перевірку вхідних даних через відсутність відповідної вимоги, жодна кількість безпечного кодування не зможе виправити цю прогалину.

Неправильна конфігурація безпеки (A05:2021)

Помилки в налаштуваннях безпеки трапляються, коли залишаються увімкненими стандартні або небезпечні налаштування, або відсутні необхідні виправлення. Сюди входять сервери без виправлень, увімкнені функції налагодження, стандартні облікові дані та неналежні дозволи. Наприклад, залишення зразків додатків на сервері, відкриті дозволи на зберігання в хмарі або докладні повідомлення про помилки (стек трасування) можуть розкрити внутрішні деталі. Сюди також потрапляють застарілі компоненти (хоча A06 охоплює їх окремо).

Вразливі та застарілі компоненти (A06:2021)

Використання бібліотек або фреймворків із відомими вразливостями є поширеним ризиком. Додаток наражається на ризик, якщо розробники не відстежують версії всіх компонентів (включно з вкладеними залежностями) або

якщо критичні виправлення затримуються. Наприклад, як показує історія, не виправлені веб-фреймворки або сервери (наприклад, Struts CVE) можуть дозволити віддалене виконання коду. Вразливості в коді сторонніх розробників стають вразливостями додатка.

Порушення ідентифікації та автентифікації (A07:2021)

Недоліки в аутентифікації та управлінні сеансами дозволяють зловмисникам отримати доступ до облікових даних користувачів або перехопити сеанси. Причини цього включають дозвіл на використання стандартних або слабких паролів, відсутність багатофакторної аутентифікації, неправильне скасування сеансів та незахищене зберігання паролів. Наприклад, якщо додаток приймає «admin/admin» або не обмежує кількість невдалих спроб входу, зловмисники можуть успішно застосувати метод credential stuffing або brute force. Зберігання паролів без хешування або повторне використання ідентифікаторів сеансів після виходу з системи також становлять ризик.

Помилки програмного забезпечення та цілісності даних (A08:2021)

Порушення цілісності відбуваються, коли код, бібліотеки або дані не перевіряються, що дозволяє підробляти або вводити ненадійний вміст. Причини включають сліпе завантаження коду або залежностей з ненадійних джерел, непідписані оновлення та незахищену десеріалізацію даних. Наприклад, якщо програма оновлюється шляхом завантаження коду без перевірки цифрового підпису, зловмисник може поширити шкідливе оновлення. Аналогічно, десеріалізація об'єктів, наданих користувачем, без перевірки може призвести до віддаленого виконання коду (ненадійні дані).

Несправності в журналюванні та моніторингу безпеки (A09:2021)

Недостатнє ведення журналів та моніторинг ускладнюють виявлення порушень. Якщо ключові події (невдалі входи, доступ до даних, помилки) не реєструються або не налаштовані сповіщення, атаки можуть залишитися непоміченими. Наприклад, відсутність журналів адміністративних дій або сповіщень про повторні невдалі спроби означає, що команда не має можливості провести розслідування або оперативно відреагувати. Крім того, ведення журналів

конфіденційних даних (паролів, токенів) саме по собі може бути вразливим місцем.

Підробка запитів на стороні сервера (SSRF) (A10:2021)

Уразливості SSRF виникають, коли програма отримує віддалений ресурс (наприклад, через URL-адресу) з використанням неперевірених даних, введених користувачем. Зловмисники використовують цю вразливість, надаючи шкідливу URL-адресу, яка змушує сервер надсилати запити від їхнього імені, можливо, до внутрішніх мереж. Наприклад, надсилання http://169.254.169.254/ в URL-адресу може розкрити секретні метадані хмари. SSRF є особливо небезпечним, оскільки обходить захист брандмауера, використовуючи контекст довіреного сервера.

Оскільки сучасні вебзастосунки все частіше пропонують користувачам зручні функції, використання URL-запитів стало поширеним явищем. Як наслідок, кількість випадків SSRF (Server-Side Request Forgery, підробка запитів на стороні сервера) зростає. Їхній вплив став ще більш значним із поширенням хмарних сервісів та збільшенням складності архітектури додатків.

Офіційний документ містить детальне пояснення кожної вразливості, включаючи критерії її класифікації, запобіжні заходи, приклади можливих сценаріїв атак та рекомендації щодо відповідних кроків, які слід вжити у разі виявлення такої вразливості. Тому експерти визнають OWASP Top 10 одним з найефективніших відправних пунктів для трансформації практик розробки програмного забезпечення з метою створення більш безпечного коду.

1.3 Оцінка ризиків, пов’язаних із використанням уразливого коду

Оцінка ризиків - це процес виявлення та аналізу загроз, які можуть виникнути через вразливості в програмному коді. У контексті цифрової трансформації вразливий код є однією з головних причин успішних атак. Неправильна перевірка вхідних даних, використання небезпечних функцій або недосконала реалізація криптографічних протоколів призводять до помилок, які дозволяють обійти аутентифікацію, виконати довільний код або отримати доступ до конфіденційних даних. Такі атаки призводять до масштабних витоків даних, критичних перебоїв у

роботі сервісів та шкоди репутації, тому регулярна оцінка ризиків вразливого коду повинна бути обов'язковою частиною життєвого циклу розробки програмного забезпечення. У цьому розділі буде проаналізовано кілька відомих вразливостей та наслідки їх використання, а також висвітлено типові методи зменшення ризиків.

1.3.1 Аналіз прикладів використання вразливого коду та впливу на компанії та індустрії

Follina (CVE 2022 30190) – виклик MSDT з документів Office

Уразливість Follina, виявлена навесні 2022 року, вплинула на механізм Microsoft Support Diagnostic Tool (MSDT). Використовуючи спеціально створений шаблон документа Word, зловмисники вставили об'єкт OLE, який посилався на віддалений HTML-файл; останній викликав протокол ms-msdt, який завантажував і виконував довільну команду PowerShell без попереднього запиту до macro [7].

Дослідники Fortinet повідомили про активне використання уразливості в цілеспрямованих атаках: кампанії китайської групи APT TA413 були спрямовані на тибетську діаспору, а один із зразків був надісланий із Саудівської Аравії і завантажив шпигунське програмне забезпечення Turian [7].

Вразливість мала оцінку CVSS 7,8; Microsoft спочатку запропонувала відключити протокол MSDT, а пізніше випустила патч. Аналітики Rapid7 відзначили, що несвоєчасне оновлення дозволило зловмисникам запускати програми та створювати нові облікові записи на комп'ютері [8].

ZeroLogon (CVE-2020-1472) – криптографічна помилка в протоколі Netlogon

У вересні 2020 року в протоколі Microsoft Netlogon Remote Protocol (MS NRPC) була виявлена уразливість ZeroLogon. Помилка полягала в тому, що для шифрування AES CFB8 використовувався вектор ініціалізації, заповнений нулями. Це дозволяло зловмисникам надсилати серію з 256 запитів з нульовими полями, що давало високу ймовірність аутентифікації та скидання пароля облікового запису контролера домену. Таким чином зловмисники могли отримати права адміністратора домену без будь-яких облікових даних [9].

Після використання вразливості зловмисники часто використовували інструменти Cobalt Strike для подальшого переміщення по мережі та розгортання

шкідливого програмного забезпечення, включаючи програми-вимагачі.

У статті GBHackers наводяться приклади груп Ryuk і RansomHub, які використовували ZeroLogon для швидкого розгортання атак — від отримання доступу до домену до повного шифрування мережі пройшло менше п'яти годин [9].

CISA випустила екстрену директиву про негайну установку патчів, а Microsoft почала поетапне впровадження захисту протоколу; проте багато організацій залишалися вразливими через застарілі системи та складність їх оновлення.

Log4Shell (CVE-2021-44228) – віддалене виконання коду через JNDI

Реєстрація помилок і подій є стандартною практикою, але помилки в сторонніх бібліотеках можуть мати катастрофічні наслідки. У грудні 2021 року в бібліотеці Apache Log4j2 була виявлена вразливість Log4Shell. Помилка в механізмі заміни рядків дозволила віддаленому зловмиснику передати спеціально створений рядок (наприклад, ${jndi:ldap://attacker.com/a}), який активував протокол JNDI та ініціював запит до зловмисного сервера.

В результаті програма завантажила та виконала довільний код з повними правами в системі. Вразливість отримала максимальний бал CVSS 10 і вплинула на численні продукти Java, включаючи Minecraft, VMware Horizon і хмарні сервіси [10].

Федеральні агентства та постачальники програмного забезпечення випустили термінові рекомендації щодо оновлення Log4j до версії 2.16.0, відключення функцій заміни та суворого фільтрування вхідних даних. Незважаючи на наявність патчів, Log4Shell залишалася однією з найпоширеніших вразливостей протягом 2023 року, а державні APT-групи та кіберзлочинці продовжували експлуатувати неналежним чином оновлені системи для розгортання ботнетів і підсистем майнінгу криптовалют [10].

ProxyLogon (CVE-2021-26855 та інші) – експлуатація Microsoft Exchange

Вразливості ProxyLogon, виявлені наприкінці лютого 2021 року, стали одними з найсерйозніших інцидентів року. За даними CISA, помилка підробки запитів сервера (CVE 2021 26855) дозволяла віддаленому зловмиснику надсилати

підроблені HTTP-запити до Microsoft Exchange, аутентифікуючись як сам сервер. У поєднанні з іншими помилками (CVE 2021 26857, CVE 2021 26858, CVE 2021 27065) зловмисник міг виконувати довільний код з привілеями SYSTEM і записувати файли в довільні місця.

Група HAFNIUM використовувала комбінацію цих вразливостей для масових хакерських атак: після отримання доступу вона розгортала веб-оболонки, виконувала дампи LSASS, експортувала поштові скриньки через PowerShell і завантажувала додаткові інструменти для подальших атак [11].

За кілька днів були скомпрометовані тисячі серверів по всьому світу, і багато організацій відклали оновлення, побоюючись порушення роботи своїх поштових систем. Після оприлюднення атак HAFNIUM компанія Microsoft випустила термінові оновлення, а CISA видала директиву, що зобов'язує застосовувати патчі та перевіряти веб-оболонки. Ці події продемонстрували, що недбалість у своєчасному встановленні оновлень може призвести до масштабного компрометації корпоративних поштових служб.

1.3.2 Методологія оцінки ризиків

Оцінка ризику пов'язана з трьома ключовими параметрами: ймовірністю експлуатації, потенційним впливом та наявністю засобів виправлення. Ймовірність залежить від популярності вразливого компонента, простоти створення експлойта та наявності готових інструментів.

Потенційний вплив визначається обсягом доступних привілеїв (виконання коду з правами SYSTEM, компрометація криптографічних ключів, відмова в обслуговуванні) та можливими наслідками для бізнесу (витік персональних даних, вимоги викупу, порушення договірних зобов'язань). Наявність засобів усунення відображає наявність патчів, складність їх встановлення та ризик простою системи.

Під час оцінки варто враховувати:

- Характер уразливості. До категорії критичних належать помилки, що дають можливість віддаленого виконання коду (RCE), обхід автентифікації або компрометацію ключів (Heartbleed, Log4Shell, ProxyLogon), оскільки вони дозволяють атакувальнику швидко заволодіти системою;

- Контекст використання. Слабке сегментування мережі та доступність вразливого сервісу із зовнішнього інтернету збільшують ризик. Наприклад, Equifax мала відкритий до мережі портал без належного захисту, що сприяло експлуатації Struts;
- Доступність експлойтів. Наявність готових PoC кодів і їхня інтеграція у популярні інструменти (Metasploit, Mimikatz) підвищують ймовірність масових атак. Для ZeroLogon експлойти були опубліковані за кілька днів після розкриття, що примусило CISA випускати екстрені директиви;
- Час реакції. Чим довше організація затримує встановлення патча, тим більша ймовірність, що уразливість буде використано. Багато серверів залишалися вразливими до Heartbleed та Log4Shell навіть через роки після виходу оновлень;
- Можливість моніторингу та виявлення. Витоки можуть тривалий час залишатися непоміченими, особливо якщо шифрування або відсутність журналів заважають виявленню. У випадку Equifax прострочений TLS сертифікат спричинив відключення мережевого моніторингу, що дозволило зловмисникам красти дані протягом кількох місяців.

Висновки до першого розділу

У першому розділі було розглянуто основні аспекти вебзастосунків та їх роль у сучасних інформаційних системах. Було встановлено, що вебзастосунки є критично важливим компонентом цифрової інфраструктури і широко використовуються для підтримки бізнес-процесів, державних послуг та орієнтованих на користувача платформ. Проведено аналіз типових вразливостей вебзастосунків на основі класифікацій OWASP Top 10 та CWE було виявлено найпоширеніші слабкі місця безпеки, включаючи недоліки ін'єкції, порушений контроль доступу, незахищені механізми автентифікації та неналежне оброблення вхідних даних користувачів.

2. АНАЛІЗ МЕТОДІВ ВИЯВЛЕННЯ ТА ЗАПОБІГАННЯ ВРАЗЛИВОСТЯМ У ВЕБЗАСТОСУНКАХ

2.1 Статичний та динамічний аналіз безпеки додатків

Оскільки безпека програмного забезпечення стає критично важливою, покладатися на один метод тестування недостатньо. Сучасні практики використовують як статичне тестування безпеки додатків (SAST), так і динамічне тестування безпеки додатків (DAST) для охоплення різних частин програмного забезпечення. SAST аналізує вихідний код або бінарні файли без виконання програми, тоді як DAST тестує працюючий додаток шляхом імітації атак.

Наприклад, CircleCI зазначає, що SAST «перевіряє код у стані спокою, щоб виявити недоліки безпеки перед розгортанням», тоді як DAST «імітує атаки на активні додатки», щоб виявити проблеми, які можна помітити лише під час виконання [12].

Експерти галузі визнають, що покладання на єдиний підхід до тестування є недостатнім для досягнення всебічної безпеки додатків. Checkmarx підкреслює, що «SAST і DAST відіграють унікальну роль у безпеці додатків», наголошуючи, що кожен метод спрямований на окремі аспекти життєвого циклу розробки та розгортання [13].

Аналогічно, GitLab підтримує цю точку зору, зазначаючи, що інтеграція обох типів тестування значно підвищує здатність організації виявляти та усувати вразливості до того, як ними можна скористатися. Статичне тестування безпеки додатків (SAST) зосереджується на аналізі вихідного коду або скомпільованих бінарних файлів з метою виявлення недоліків на ранніх етапах процесу розробки, що дозволяє розробникам усунути проблеми ще до запуску додатка. На відміну від цього, динамічне тестування безпеки додатків (DAST) працює з зовнішньої точки зору, оцінюючи запущені додатки в режимі реального часу з метою виявлення вразливостей, які можуть проявитися лише під час виконання.

Оскільки ці дві методології розглядають безпеку з принципово різних точок зору – SAST зсередини коду, а DAST з поведінки активної системи – їхнє комбіноване застосування забезпечує набагато більш комплексний захист. При спільному використанні вони заповнюють прогалину між превентивними та детективними заходами безпеки, створюючи багаторівневий підхід, який значно посилює загальний рівень безпеки організації.

2.1.1 Статичне тестування безпеки додатків (SAST)

Статичне тестування безпеки додатків (SAST) – це аналіз вихідного коду або двійкового коду методом «білого ящика». Воно перевіряє кодову базу програми без запуску додатка, зазвичай на ранній стадії процесу розробки. Інструменти SAST застосовують до коду зіставлення шаблонів і аналіз потоку даних, шукаючи небезпечні шаблони кодування та типові недоліки. Наприклад, CircleCI пояснює, що SAST «аналізує вихідний код програми для виявлення вразливостей безпеки», таких як SQL-ін'єкція та переповнення буфера [12].

Оскільки SAST працює на коді, а не на живій системі, його можна інтегрувати безпосередньо в IDE та CI/CD-пайплайни, щоб надавати розробникам миттєвий зворотній зв'язок щодо вразливостей під час написання коду. Важливо зауважити що статичне тестування також може використовуватись методом «сірого ящика» та «чорного ящика».

У цьому випадку інструменту статичного аналізу не дається доступ до самого коду однак він може працювати з його вхідними та вихідними даними. Однак з огляду на принцип його використання в сучасних індустріях метод «білого ящика» є більш ефективним для виявлення вразливостей.

Статичний аналіз є найбільш ефективним під час розробки та перевірки коду. Він виявляє помилки кодування до розгортання, що робить виправлення дешевшими та швидшими.

Дослідження показують, що виявлення недоліків на ранній стадії є набагато менш витратним: SAST виявляє вразливості безпеки на ранній стадії, коли код ще розробляється, і їх можна усунути, перш ніж вони поширяться на пізніші етапи. На практиці команди часто налаштовують SAST для сканування коду під час коміту

або запиту на витяг. Нижче наведено візуальний приклад виявлення вразливості типу SQL ін’єкції методом SAST (рисунок 2.1).

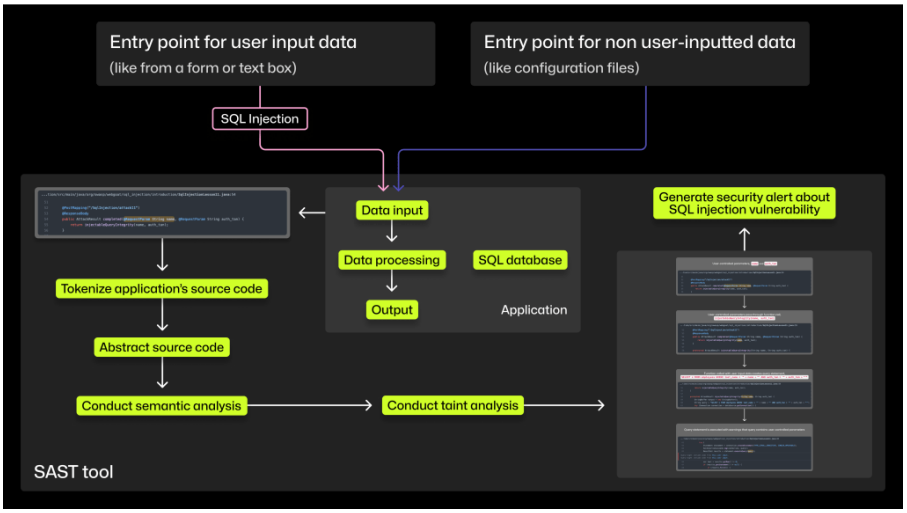

Рис. 2.1. Виявлення вразливості типу SQL ін’єкції методом SAST [14]

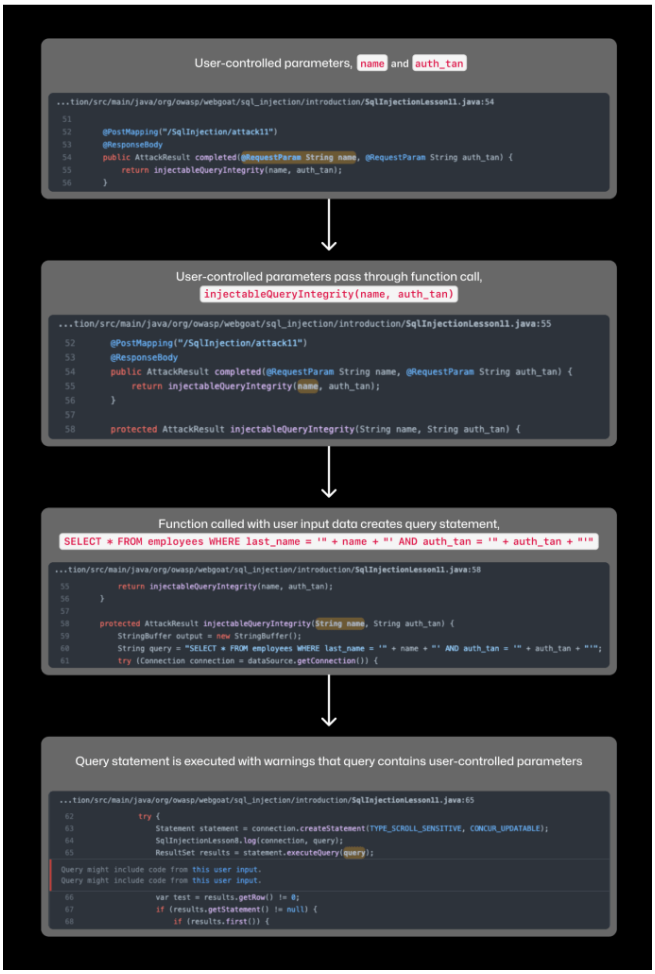

Рис. 2.2. Діаграма потоку даних користувацьких параметрів у вразливому коді (приклад SQL Injection)[14]

Вище наведено приклад того, як процес статичного тестування безпеки додатків відстежує потік даних (рисунок 2.2). Два твердження в останньому кроці, «Запит може містити код з цього введення користувача», є свідченням того, що аналіз потоку даних працює. Механізм статичного аналізу розпізнає, що надані користувачем дані, ім'я та auth_tan, безпосередньо вбудовані в SQL-запит:

SELECT * FROM employees WHERE last_name = '“ + name + “‘ AND auth_tan = '“ + auth_tan + “‘“.

Виконання цього запиту може спричинити появу попередження про безпеку, якщо введені користувачем дані не пройшли достатню санітарну обробку або взагалі не пройшли санітарну обробку. Це ілюструє основний принцип виявлення SAST – ідентифікацію потенційних точок введення шляхом відстеження потоку неперевірених даних, введених користувачем, у чутливі операції коду, такі як запити до бази даних.

2.1.2 Динамічне тестування безпеки додатків (DAST)

Динамічне тестування безпеки додатків (DAST) – це аналіз «чорного ящика», що виконується на працюючому додатку. Інструменти DAST сканують активне програмне забезпечення (наприклад, веб-сервер або API), імітуючи зовнішні атаки, без будь-якого знання вихідного коду.

Як пояснює CircleCI, DAST «імітує атаки на активні додатки, щоб знайти вразливості, які видно тільки під час виконання» [12]. DAST оцінює безпеку додатка під час його роботи, імітуючи погляд зловмисника ззовні. Оскільки DAST тестує розгорнутий додаток, для його роботи програмне забезпечення має перебувати в тестовому або проміжному середовищі (або навіть у робочому середовищі в сценаріях безперервного моніторингу).

DAST має можливість виявляти різні недоліки безпеки, такі як вразливості типу «відмова в обслуговуванні» (DoS) та незахищені конфігурації серверів. Імітуючи реальні сценарії атак, інструменти DAST оцінюють, як програма реагує на ці модельовані загрози. За допомогою такого підходу розробники можуть виявляти слабкі місця, які можуть бути пропущені статичними методами тестування, такими як SAST.

Зазвичай розробники виконують DAST на пізніх етапах життєвого циклу розробки програмного забезпечення, часто безпосередньо перед розгортанням. Тестуючи додатки в їхніх реальних середовищах, DAST надає чіткий огляд безпеки під час виконання та підтримує більш ефективні додатки, які можуть протистояти потенційним атакам.

Високорівнево процес виявлення вразливостей з допомогою DAST можна розділити на 4 етапи: сканування, моделювання атаки, виявлення вразливостей, звітність. Нижче наведено схему цих 4 етапів (рисунок 2.3).

Рис. 2.3. Діаграма 4 етапів DAST [15]

Етап сканування передбачає перевірку вебзастосунку з метою виявлення потенційних точок входу, таких як URL-адреси, форми та API. Цей процес відображає архітектуру додатку та виділяє можливі поверхні атаки, які можуть бути використані зловмисниками.

Далі йде етап моделювання атаки, під час якого система надсилає спеціально створені запити до виявлених точок входу. Ці симуляції призначені для виявлення

вразливостей, таких як міжсайтовий скриптинг (XSS) та міжсайтове підроблення запитів (CSRF), шляхом контрольованих спроб експлуатації.

На етапі виявлення вразливостей аналізуються відповіді додатка для визначення потенційних слабких місць. Цей аналіз перевіряє, чи додаток поводиться очікувано при впливі зловмисних вхідних даних. Наприклад, введення спеціально створених даних може виявити вразливість до SQL-ін'єкцій.

На етапі звітування генеруються аналітичні результати, що містять виявлені вразливості та рекомендації щодо їх усунення. Ці звіти надають розробникам практичні відомості для ефективного вирішення виявлених проблем безпеки. Сучасні реалізації DAST часто інтегрують передові технології, такі як штучний інтелект та обробка даних у реальному часі. Вони автоматично генерують адаптивні набори тестів, динамічно пристосовуються до поведінки додатка та використовують алгоритми машинного навчання для зменшення кількості помилкових спрацьовувань.

У таблиці нижче наведено основні відмінності між SAST і DAST.

Таблиця 2.1

Порівняння DAST та SAST методів

Критерій	SAST	DAST
Тип тестування	Тестування типу white-box. Перевіряє застосунок із середини, маючи доступ до вихідного коду.	Тестування типу black-box. Перевіряє застосунок зовні без доступу до вихідного коду.

Продовження таблиці 2.1

Критерій	SAST	DAST
Типи підтримуваного програмного забезпечення	Підтримує різні типи програмного забезпечення, включно з вебзастосунками, вебсервісами та «товстими» клієнтами.	Підтримує вебзастосунки та вебсервіси, але зазвичай не охоплює інші типи програмного забезпечення.
Етап у життєвому циклі розробки ПЗ (SDLC)	Виконується на ранніх етапах життєвого циклу розробки програмного забезпечення.	Проводиться на пізніших етапах SDLC – під час або після розгортання.
Виявлювані вразливості	Визначає помилки у вихідному коді, зокрема: SQL-ін’єкції, міжсайтові сценарії (XSS), переповнення буфера тощо.	Виявляє проблеми під час виконання, зокрема помилки конфігурації серверів, вразливості типу відмови в обслуговуванні (DoS) та інші дефекти на рівні застосунку.
Вартість усунення вразливостей	Менша, оскільки проблеми виявляються на ранніх етапах SDLC.	Вища, адже вразливості виявляються пізніше та часто потребують термінових виправлень для своєчасного розгортання.

Продовження таблиці 2.1

Критерій	SAST	DAST
Проблеми виконання та середовища	Не здатен виявляти вразливості, що залежать від середовища або з’являються лише під час виконання.	Може виявляти вразливості, які проявляються тільки під час виконання або зумовлені середовищем.
Глибина проти широти аналізу	Забезпечує глибоке розуміння вразливостей на рівні вихідного коду.	Надає широкий огляд зовнішніх ризиків безпеки застосунку.
Інтеграція з інструментами	Інтегрується з IDE та CI/CD-пайплайнами для автоматизованого статичного аналізу.	Інтегрується з CI/CD-пайплайнами для безперервного тестування під час виконання.
Хибнопозитивні та хибнонегативні результати	Має вищу ймовірність хибнопозитивних результатів через глибокий аналіз коду.	Має нижчу кількість хибнопозитивних результатів, але більший ризик хибнонегативних через обмежену видимість внутрішньої логіки.
Залежність від технологій	Залежить від мов програмування та фреймворків; потребує сумісності з інструментом.	Не залежить від фреймворків застосунку, оскільки взаємодіє із системою зовні.

2.2 Аналіз сучасних рішень для статичного аналізу коду (Semgrep, SonarQube, Brakeman, Snyk Code)

Сучасні інструменти статичного аналізу сканують вихідний код для виявлення вразливостей безпеки перед розгортанням. Вони аналізують код без виконання (тестування «білого ящика»), що робить їх придатними для раннього виявлення логічних недоліків, небезпечних шаблонів та помилок кодування. У веб-розробці зазвичай підтримуються такі поширені мови, як JavaScript, Java, Python, C#, PHP та Ruby. У цьому розділі розглядаються чотири типові інструменти SAST: Semgrep, SonarQube, Brakeman та Snyk Code. Кожен з них використовує різні підходи (правила на основі шаблонів, аналіз AST тощо) і призначений для різних випадків використання, але всі вони мають на меті допомогти розробникам виявляти та виправляти вразливості коду.

2.2.1 Semgrep

Semgrep (скорочення від semantic grep) – це швидкий і легкий механізм SAST, розроблений для простого написання правил та інтеграції в робочі процеси розробників. Правила пишуться у загальному форматі YAML, який працює в різних мовах, що дозволяє користувачам здійснювати семантичний пошук коду. Semgrep можна запускати локально, в IDE або в пайплайнах CI/CD для отримання зворотного зв'язку щодо безпеки коду в режимі реального часу.

Згідно з документацією, «Semgrep Code – це інструмент статичного тестування безпеки додатків (SAST), який виявляє вразливості безпеки у вашому власному коді» [16]. Коли Semgrep знаходить код, що відповідає шаблону правила (включаючи опціональні умови потоку даних), він повідомляє про знахідку, допомагаючи дотримуватися найкращих практик або виявляти помилки. Semgrep підтримує десятки мов (понад 35 у комерційній версії Semgrep Code), включаючи JavaScript, Python, Java, Go, C#, Ruby та інші. Наприклад, реєстр правил Semgrep містить перевірки на типові проблеми вебзастосунків (SQL-ін'єкції, XSS тощо) для цих мов.

Правила можна налаштовувати або писати з нуля, що робить Semgrep дуже

адаптивним. На практиці Semgrep Code (платна версія) додає міжфайловий аналіз для зменшення кількості помилкових спрацьовувань, тоді як відкрита версія Community Edition зосереджується на внутрішньофункціональних шаблонах (рисунок 2.4).

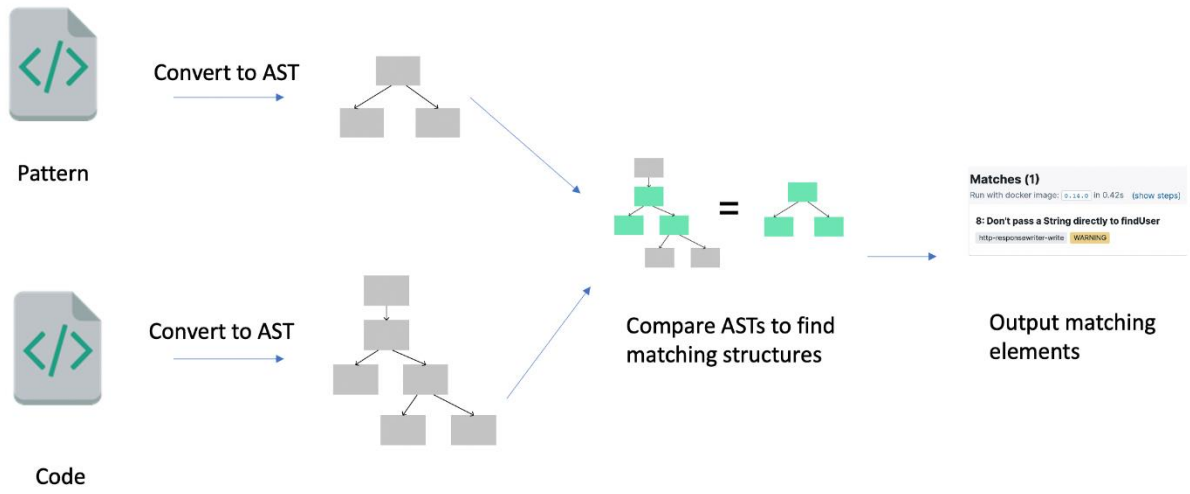

Рис. 2.4. Приклад діаграми співпадіння шаблонів Semgrep [16]

2.2.2 SonarQube

SonarQube – це стандартна в галузі платформа для автоматизованого перегляду коду. SonarScanner працює під час побудови CI, завантажує результати аналізу на сервер вендора SonarQube та застосовує до коду багатий набір правил і профілів якості. SonarQube підтримує понад 30 (що є типовою нормою для SAST додатків на ринку) мов програмування та пов'язані з ними фреймворки (Java, C#, C/C++, JavaScript/TypeScript, Python, Go, PHP, Ruby тощо).

Однак важливо зауважити, що він є обмеженим у доступних бібліотеках в цих мовах програмування, що відзначають розробники Python JS та інших мов з великою кількістю сторонніх бібліотек. Для кожного проєкту або запиту на витяг SonarQube обчислює метрики (дублювання, складність, покриття тестами) і позначає проблеми, включаючи помилки, підозрілі фрагменти коду та вразливості безпеки [17].

Згідно з офіційною документацією, SonarQube виконує автоматизовані перевірки якості та безпеки коду, надаючи практичну інформацію, яка допомагає розробникам створювати кращі та швидші продукти [17]. На практиці SonarQube широко використовується підприємствами. Community Edition (відкритий код LGPL) надає основні функції, а комерційні версії додають підтримку більшої кількості мов та правила безпеки (наприклад, перевірки OWASP Top 10) (рисунок 2.5).

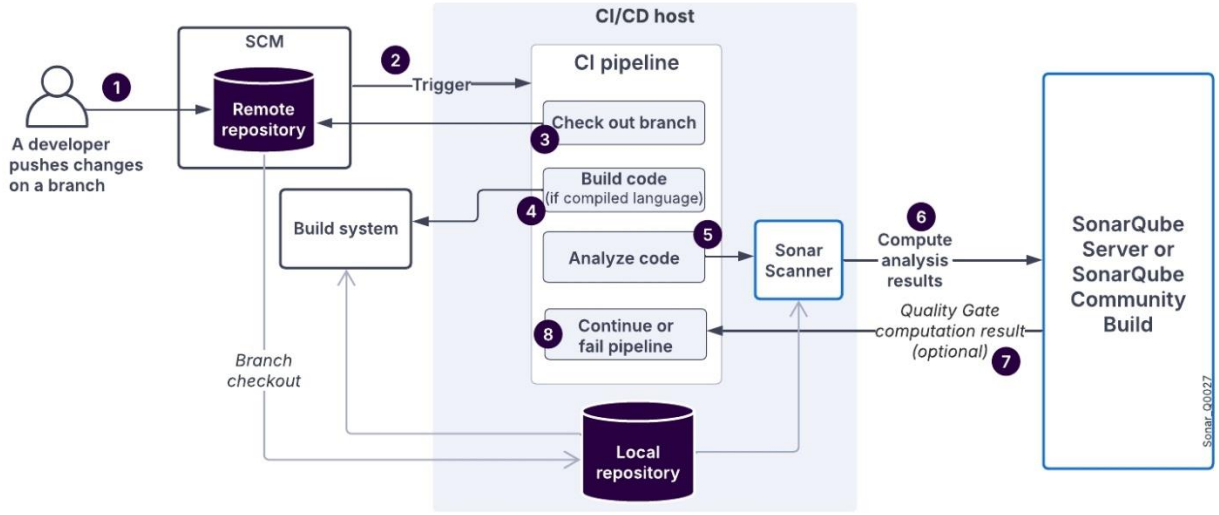

Рис. 2.5. Огляд серверного аналізу коду SonarQube [16]

2.2.3 Brakeman

Brakeman – це нішевий інструмент статичного аналізу, спеціально розроблений для додатків Ruby on Rails. На відміну від сканерів загального призначення, Brakeman розуміє конвенції та шаблони Rails, що дозволяє йому виявляти специфічні для Rails вразливості в вихідному коді на ранніх етапах розробки. Він є відкритим і безкоштовним для використання. Перевіряючи код Rails, Brakeman може виявити такі проблеми, як SQL-ін'єкція в запитах ActiveRecord, небезпечне масове присвоєння, небезпечне використання допоміжних засобів рендерингу та багато інших.

Brakeman навмисно простий у використанні – зазвичай він не вимагає жодних налаштувань і може бути запущений за допомогою простої команди (наприклад,

brakeman у каталозі проекту Rails)[17]. Він автоматично завантажує будь-які файли Ruby та конфігурації Rails, щоб перевірити наявність небезпечних шаблонів. Основні характеристики Brakeman:

- Brakeman адаптований до Rails, тому він розпізнає контролери, моделі, види та маршрутизацію Rails. Він може дотримуватися конвенцій Rails (шаблонів MVC) для пошуку джерел та приймачів вхідних даних користувача.
- Brakeman працює «з коробки» з розумними налаштуваннями за замовчуванням; для початку сканування додатка Rails не потрібні спеціальні налаштування
- Правила Brakeman виявляють багато поширених проблем вебзастосунків, таких як SQL-ін'єкція, міжсайтовий скриптинг (XSS), ін'єкція команд та десятки інших типів вразливостей.

2.2.4 Snyk Code

Snyk Code – це хмарне рішення SAST від Snyk, орієнтоване на розробників, інтегроване в платформу Snyk. Воно наголошує на швидкості, контекстуальній точності та автоматизованому виправленні.

На практиці Snyk Code безперервно сканує вихідний код (і навіть код, створений за допомогою генеративної штучної інтелекту) в IDE, pull-запитах і збірках, надаючи миттєвий зворотний зв'язок і пропозиції щодо виправлення безпосередньо в робочих процесах розробників. Така тісна інтеграція означає, що вразливості можна виявляти та виправляти під час кодування, а не після нього.

Snyk Code підтримує найпопулярніші мови програмування, що використовуються у веб-розробці (такі як JavaScript/TypeScript, Java, Python, Go, C#, PHP тощо). Snyk Code охоплює ~90% бібліотек AI/LLM (OpenAI, HuggingFace тощо), розглядаючи їх використання як джерела/приймачі даних [18].

В основі Snyk Code поєднує традиційний статичний аналіз із базою знань на основі машинного навчання: він змоделював понад 25 мільйонів випадків потоку даних з бібліотек з відкритим кодом для виявлення забруднених потоків. В

результаті рішення Snyk Code має менше хибно позитивних спрацювань в порівнянні з іншими рішеннями. Відповідно до офіційної документації можна виділити три основні характеристики рішення:

- Плагін IDE та перевірки pull-request відображають проблеми в режимі реального часу під час написання коду розробниками. Snyk рекламує «виправлення з точністю 80%» та миттєве сканування (не потрібно створювати збірку).
- Snyk Code охоплює всі основні мови та фреймворки, а також постійно додає нові. У документації зазначено широку сумісність з IDE та інструментами CI.
- Движок Snyk Code на основі штучного інтелекту надає практичні поради щодо виправлення та навіть автоматичні виправлення коду для багатьох виявлених проблем. Багата база знань допомагає позначати найризикованіші проблеми для особливої уваги.

Нижче наведено приклади спрацювань для Snyk Code та детальний опис вразливості (рисунок 2.6, 2.7).

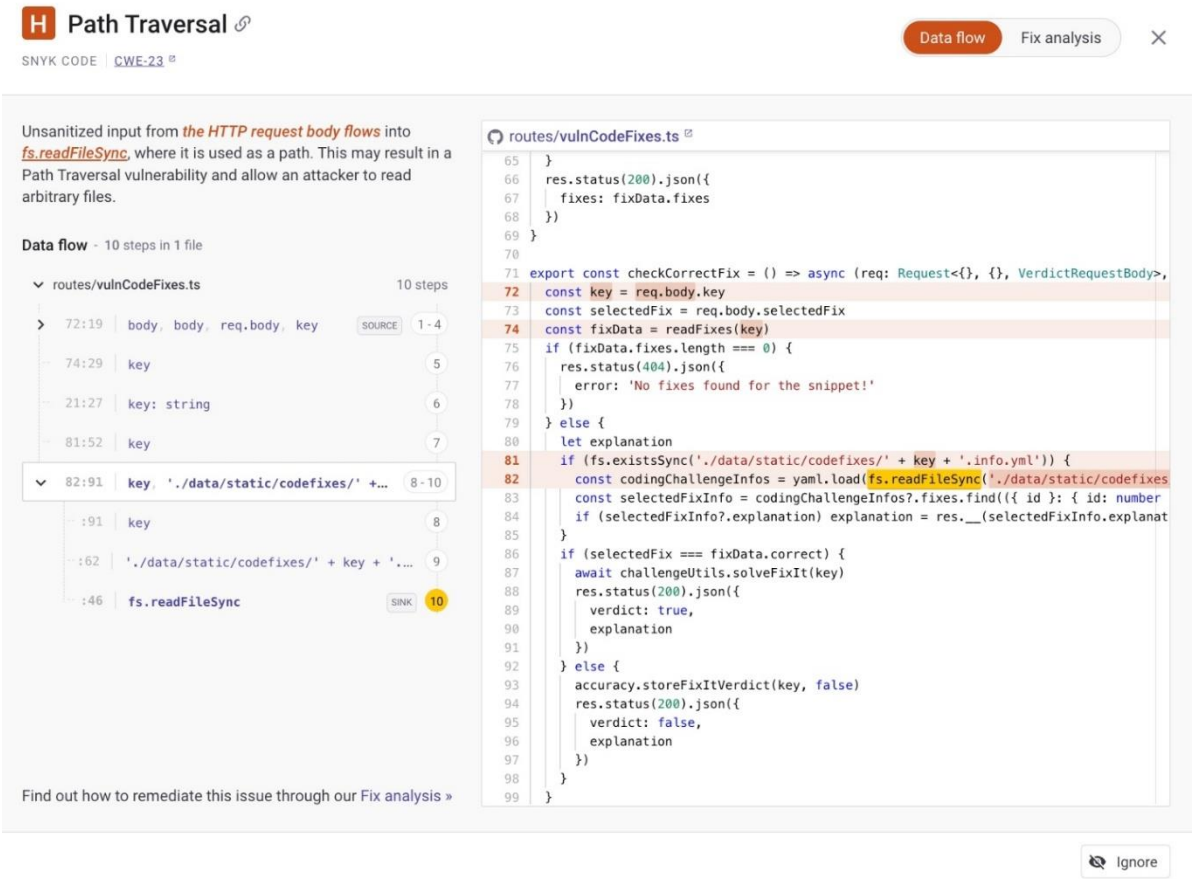

Рис. 2.6. Приклад роботи Snyk Code виявлення Path traversal [18]

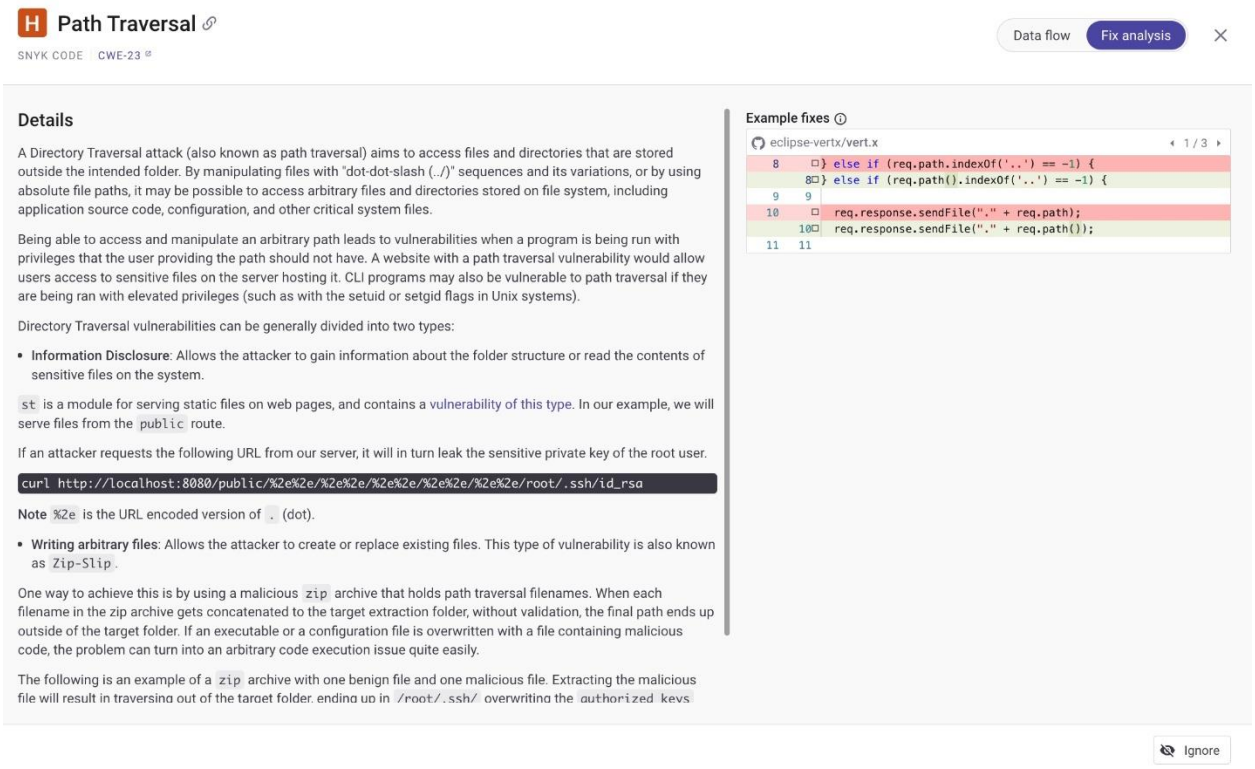

Рис. 2.7. Приклад роботи Snyk Code опис виявленої вразливості Path traversal [18]

У наступній таблиці наведено основні характеристики кожного інструменту.

Таблиця 2.2

Порівняння описаних рішень в розділі 2.2

	Semgrep	SonarQube	Brakeman	Snyk Code
Підтримувані мови програмування	Понад 30 мов (наприклад, Python, JS/TS, Java, Go, C#, Ruby, PHP тощо).	Понад 30 мов (Java, C#, C/C++, JS, TS, Python, Go, PHP, Ruby, Swift тощо).	Лише Ruby (додатки на Rails).	Основні мови (JavaScript/TypeScript, Java, Python, Go, C#, PHP тощо) (+ ~90% бібліотек для AI/LLM).

Продовження таблиці 2.2

	Semgrep	SonarQube	Brakeman	Snyk Code
Ліцензія / Редакція	Відкритий код (Community – LGPL); платні плани (Semgrep Code).	Community (відкритий код, LGPL) і комерційні (Enterprise) редакції.	Безкоштовний, відкритий код (MIT).	SaaS (комерційна), безкоштовна для малих команд.
Інтеграція / Використання	CLI/IDE/плагін; CI/CD (GitHub Actions тощо). Працює швидко в редакторі або під час pre-commit.	Локальний або хмарний сервер; інтеграція з інструментами збірки (Maven, Gradle, npm тощо) та системами контролю версій. Також плагін для IDE (SonarLint).	Працює як CLI-інструмент (команда brakeman) або етап у CI. Інтегрується в проєкти Rails.	Плагін для IDE (VS Code, IntelliJ), інтеграції з CI/CD (GitHub, GitLab, Azure Pipelines тощо), CLI.
Автоматичне виправлення	Ні (лише звіти)	Обмежено (лише пропозиції)	Ні	Так (вбудовано)

Продовження таблиці 2.2

	Semgrep	SonarQube	Brakeman	Snyk Code
Основні особливості	Швидке сканування на основі шаблонів; користувацькі правила; прозорий і детермінований аналіз.	Широкий аналіз якості та безпеки коду; тисячі правил; quality gates; метрики коду (покриття, дублювання); аналіз поширення даних (taint analysis).	Спеціалізований сканер для Rails; не потребує налаштування; виявляє вразливості Rails (SQLi, XSS тощо).	Орієнтований на розробників SAST; сканування в реальному часі; AI-рекомендації з виправлення; база знань для контексту; низький рівень хибних спрацьовувань.

2.3 Огляд архітектури рішення Snyk Code та його функціональних можливостей

Snyk Code - це інструмент для статичного тестування безпеки додатків (SAST), розроблений як рішення, орієнтоване на розробників. Він забезпечує сканування вихідного коду в режимі реального часу в IDE, репозиторіях контролю версій та пайплайнах CI/CD, надаючи практичну інформацію про вразливості на ранніх етапах процесу розробки. Основою Snyk Code є аналітичний движок на базі штучного інтелекту (спочатку похідний від DeepCode AI), який використовує передовий семантичний аналіз.

Цей гібридний движок поєднує символічний штучний інтелект (аналіз програм) з великими мовними моделями для виявлення складних шаблонів безпеки і навіть пропонує способи їх виправлення. Завдяки такому підходу на базі штучного інтелекту Snyk Code забезпечує дуже низький рівень помилкових спрацьовувань і

високу точність виявлення, що дозволяє розробникам зосередитися на реальних проблемах.

Загальна архітектура Snyk Code побудована на хмарній платформі Snyk, але пропонує гнучкі режими розгортання. У стандартному режимі SaaS Snyk Code інтегрується безпосередньо з репозиторіями Git (GitHub, GitLab, Bitbucket тощо) та веб-інтерфейсом Snyk.

У цьому режимі вихідний код (або його знімки) завантажується до багатокористувацької хмари Snyk для аналізу. Хмарний сервіс запускає AI-двигуни DeepCode на захищених кластерах, автоматично масштабується і повертає результати сканування користувачеві через інтерфейс Snyk або CLI. Крім того, для клієнтів з приватними локальними репозиторіями або більш суворими політиками щодо даних Snyk пропонує два ключові варіанти: Broker і Local Engine. Snyk Broker — це локальний проксі-сервер, який дозволяє хмарі Snyk безпечно отримувати код з внутрішніх SCM (залишаючи облікові дані всередині корпоративної мережі).

Типова архітектурна схема Broker показує клієнт, встановлений в інфраструктурі клієнта, який встановлює тунель HTTPS/WebSocket до Snyk SaaS (схема доступна в документації Snyk). Для найвищого рівня контролю даних Snyk Code пропонує Local Engine: розгортання на базі Kubernetes, яке запускає весь механізм аналізу Snyk Code локально. У режимі Local Engine вихідний код не залишає інфраструктуру клієнта — лише кінцеві результати (проблеми та метадані) завантажуються назад до хмари Snyk для звітності. Схема ілюстрації архітектури Local Engine у кластері Kubernetes зображена нижче на малюнку 2.8.

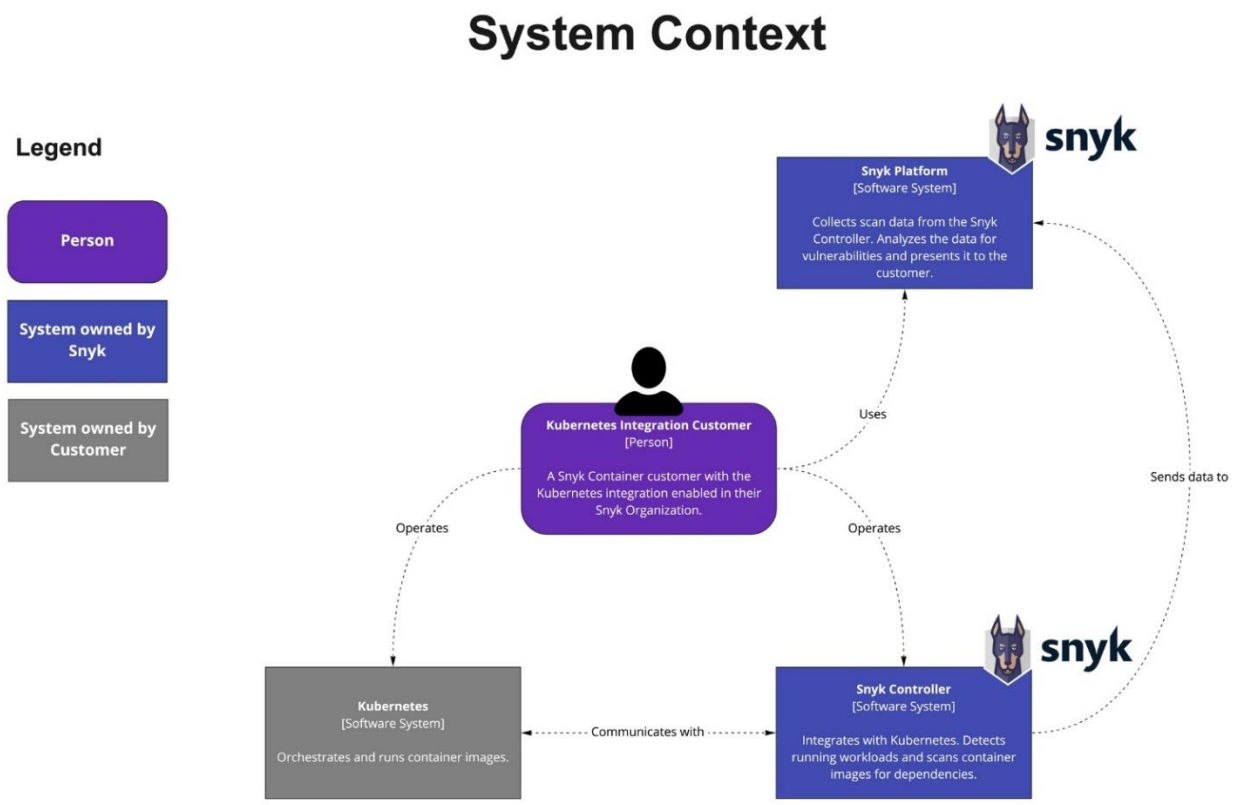

Рис. 2.8. Архітектура Local Engine у кластері Kubernetes [19]

Під цими варіантами розгортання робочий процес аналізу Snyk Code відповідає стандартним пайплайнам статичного аналізу. Snyk Code виконує лексичний та синтаксичний аналіз (побудова AST) вхідного коду, а потім застосовує набір правил і моделей для обходу графіків потоку управління та потоку даних.

Фактично, движок будує семантичну модель коду, щоб відстежувати, як дані переміщуються від джерел до приймачів (аналіз забруднення), як використовується пам'ять і як поводяться структури управління. Наприклад, він перевіряє шаблони використання API на наявність небезпечних викликів або нульових дереференцій, моделює кожну гілку та виклик функції і виводить типи змінних. Потім він відстежує дані від вхідних до вихідних, щоб знайти ризики ін'єкції або небезпечну десеріалізацію, виявляє умови гонки, аналізуючи потік управління, і навіть обчислює діапазони значень (щоб виявити помилки off-by-one або division-by-zero).

Цей багатий статичний аналіз підкріплений великою базою знань: DeepCode

AI від Snyk повідомляє про понад 25 мільйонів випадків потоку даних, отриманих з мільйонів проєктів з відкритим кодом, і підтримує сканування на понад 19 мовах програмування. Список включає всі основні веб- та хмарні мови та фреймворки, наприклад Java, JavaScript/TypeScript, C/C++, C#, Go, Python, Ruby, PHP, Scala, Swift/Objective-C, Rust, Kotlin та інші. (У квітні 2023 року Snyk спеціально додав підтримку C/C++, застосувавши свій семантичний AI-движок до цих мов.) Міжпроцедурний та багатофайловий аналіз («міжфайловий аналіз») доступний майже для всіх підтримуваних мов (єдиним винятком є Ruby).

Що стосується функціональних можливостей, Snyk Code пропонує розробникам комплексний набір функцій. Він забезпечує сканування в IDE за допомогою плагінів (наприклад, для VS Code, IntelliJ/JetBrains, Visual Studio), сканування з командного рядка за допомогою snyk code test та сканування репозиторію в веб-інтерфейсі Snyk. В таблиці нижче наведено деталі для роботи плагінів та відповідних IDE.

Таблиця 2.3

Версії кожної IDE, що підтримуються плагіном або розширенням Snyk

Плагін або розширення Snyk	Підтримувана версія IDE
Розширення Visual Studio Code	Остання версія розширення Snyk для Visual Studio Code підтримує використання з Visual Studio Code версії 1.76.0 і новіших.
Плагін JetBrains	Останній плагін Snyk для JetBrains підтримує всі IDE JetBrains версії 2024.2 або новіші.
	Старіші версії плагіна можуть підтримувати IDE JetBrains версій 2020.3 або новіших.

Продовження таблиці 2.3

Плагін або розширення Snyk	Підтримувана версія IDE
Розширення Visual Studio	Остання версія розширення Snyk для Visual Studio підтримує використання з Visual Studio 2022 (версія 17.5 і вище).
	Старіші версії розширення можуть підтримувати Visual Studio 2015, 2017 та 2019.
Плагін Eclipse	Останній плагін Snyk для Eclipse підтримує використання з Eclipse 2024-03 або новіших версій.
	Старіші версії плагіна можуть підтримувати Eclipse 2023-03 або новіші.

Snyk Code може автоматично перевіряти кожен запит на витягнення на відповідність правилам безпеки, блокуючи злиття, що спричиняють критичні проблеми. Він також інтегрується з CI/CD: до пайплайнів можна додати крок Snyk CLI для сканування коду на кожній збірці. Крім того, Snyk Code надає REST API, щоб спеціальні інструменти або інформаційні панелі могли запитувати проекти та проблеми, а також підтримує експорт до систем відстеження проблем (інтеграція з Jira) для відстеження виправлень.

Після сканування коду інтерфейс Snyk Code відображає результати у вигляді зручних для розробників відомостей. Проблеми фільтруються, сортуються та групуються за такими критеріями, як серйозність, мова та розташування файлу. Інструмент обчислює показник пріоритетності, який враховує поширеність проблеми, легкість її виправлення та можливість використання, допомагаючи командам спершу зосередитися на найважливіших вразливостях.

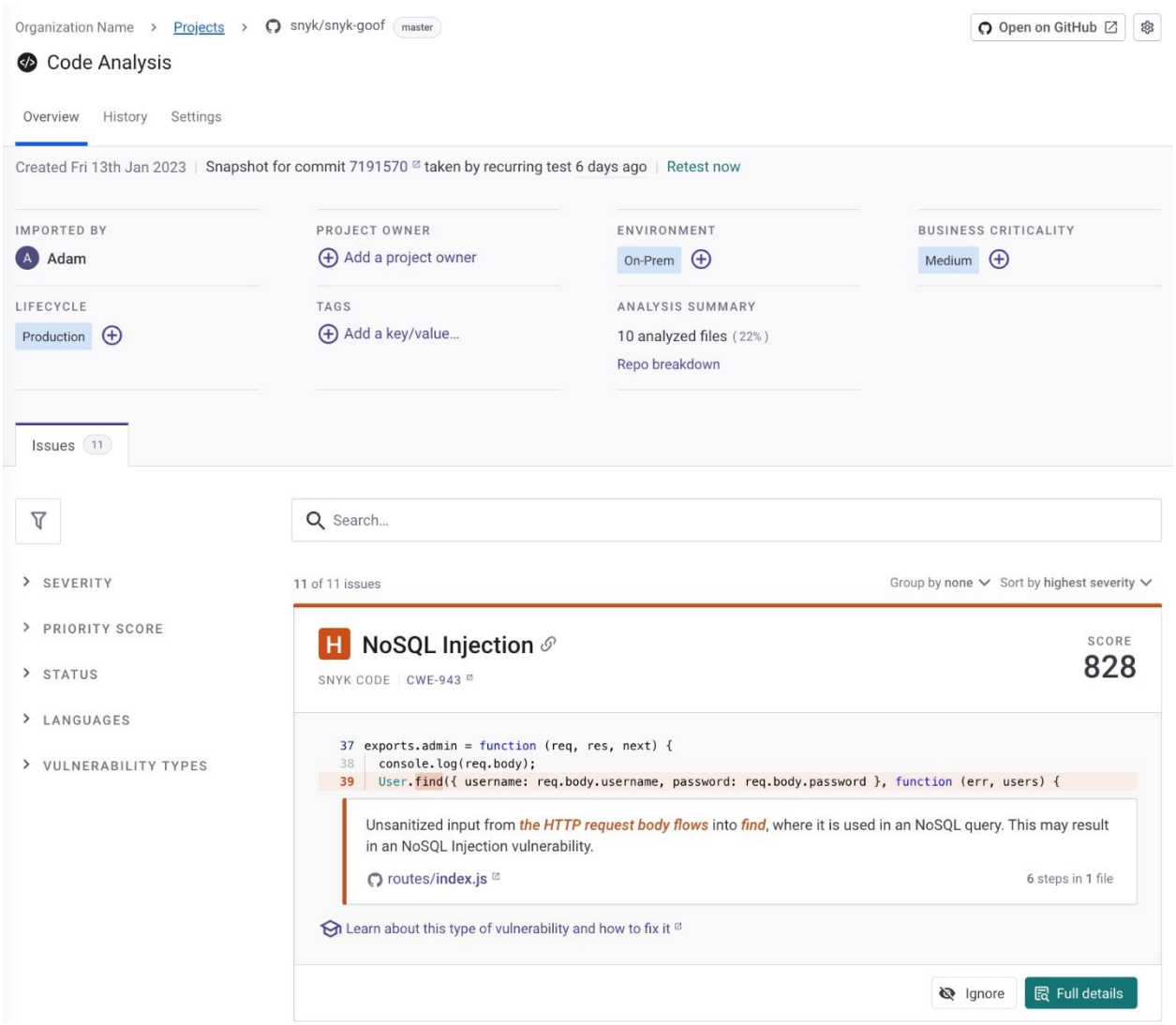

Рис. 2.9. Сторінка «Аналіз коду»

Для кожного виявленого випадку Snyk Code показує шлях потоку даних від проблемного джерела до вразливого приймача, чітко показуючи, як поширюються небезпечні дані. Кожна вразливість пов'язана з відібраною інформацією: поясненням причини недоліку, факторами ризику та типовими заходами щодо його усунення. Важливо, що Snyk Code надає пропозиції щодо виправлення та посилання: він показує приклади того, як виправити проблему (часто за допомогою посилань на подібні виправлені фрагменти коду).

У деяких випадках Agent Fix AI від Snyk може навіть створити патч за лічені секунди, досягаючи до 80% точності в автоматизованих виправленнях. Платформа також підтримує сортування проблем: можна позначити проблеми як ігноровані (для відомих винятків) або автоматично пропускати файли за допомогою файлу

ігнорування .dcignore. Всі історичні знімки сканування зберігаються, а вразливості повторно перевіряються за запитом.

Архітектура Snyk Code побудована як гнучка, масштабована платформа, що інтегрує як хмарні, так і опціональні локальні компоненти, що працюють на базі передового семантичного AI-двигуна.

Вона забезпечує всебічне покриття випадків використання SAST для сучасної веб-розробки, включаючи глибокий багатомовний аналіз, інтеграцію з IDE та CI-пайплайнами, а також детальну, контекстуалізовану звітність з потоками даних, рекомендаціями та автоматизованими виправленнями. Ці можливості підтримують робочі процеси, орієнтовані на розробників, такі як перевірка запитів на витягнення та управління проблемами, забезпечуючи безперервну безпеку коду. Завдяки цій адаптивній архітектурі Snyk Code може обслуговувати як невеликі команди, так і великі підприємства, безперешкодно пристосовуючись до масштабів та вимог безпеки кожного середовища.

Висновки до другого розділу

Другий розділ був присвячений аналізу існуючих методів виявлення та запобігання вразливостей у вебзастосунках. Були розглянуті підходи до статичного та динамічного тестування безпеки додатків, а також визначені їхні переваги та обмеження. Було встановлено, що хоча динамічний аналіз є ефективним для виявлення вразливостей під час виконання, він має обмежену охопність і значною мірою залежить від якості тестових сценаріїв.

Аналіз продемонстрував, що Snyk Code надає розширені функціональні можливості, поєднуючи семантичний аналіз коду з урахуванням контексту вразливостей та безперебійною інтеграцією в середовища розробки та конвеєри CI/CD. Його підхід, орієнтований на розробників, підтримує практики безпечного кодування та відповідає сучасним принципам DevSecOps. Результати цього розділу обґрунтовують вибір Snyk Code як відповідного та ефективного рішення для статичного виявлення вразливостей у вебзастосунках.

3 ТЕХНОЛОГІЯ СТАТИЧНОГО АНАЛІЗУ КОДУ ДЛЯ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ У ВЕБЗАСТОСУНКАХ

3.1 Вимоги, встановлення та налаштування Snyk Code для статичного аналізу безпеки

У цьому розділі розглядається, як налаштувати Snyk Code, інструмент статичного тестування безпеки додатків (SAST) від Snyk, для сканування коду вебзастосунків. Snyk Code – це рішення SAST, орієнтоване на розробників, яке інтегрується в робочий процес, включаючи IDE, репозиторії коду та пайплайни CI/CD, для виявлення проблем безпеки.

Його механізм на основі штучного інтелекту зменшує кількість помилкових спрацьовувань і надає практичні рекомендації щодо виправлення. Snyk Code підтримує більшість мов веб-розробки (JavaScript, TypeScript, Python, PHP, Ruby, .NET тощо) і доступний у безкоштовному плані Snyk (для окремих осіб та невеликих команд), а також у корпоративних пропозиціях з розширеними функціями.

Налаштування Snyk Code передбачає кілька етапів: перевірку попередніх вимог, встановлення необхідних інструментів та конфігурацію Snyk (як в інтерфейсі користувача, так і в пайплайнах збірки). Далі ми розглянемо кроки для налаштування Snyk, його імплементацію, та рекомендації щодо використання та застосування його в корпоративному середовищі.

3.1.1 Вимоги

Перед інтеграцією Snyk Code у робочий процес розробки програмного забезпечення важливо розуміти, що цей продукт є хмарним рішенням для статичного тестування безпеки додатків (SAST). Як наслідок, традиційне локальне розгортання («встановлення на місці») не підтримується. Натомість процес впровадження зосереджується на налаштуванні організаційних робочих процесів, середовищ розробки та пайплайнів автоматизації, щоб забезпечити безпечну

передачу, аналіз та постійний моніторинг вихідного коду хмарним сервісом Snyk Code. Однак компанії та команди з розробки мають відповідати певним вимогам перед використанням.

По-перше, необхідний активний обліковий запис Snyk, а також відповідна роль (зазвичай адміністратор організації), щоб увімкнути функції Snyk Code. Організація повинна бути підключена до підтримуваної системи управління вихідним кодом (SCM) (GitHub, GitLab, Bitbucket, Azure Repos тощо). Обліковий запис Snyk повинен бути інтегрований із SCM, щоб можна було імпортувати репозиторії; Snyk Code тимчасово клонує вибрані репозиторії для сканування, що вимагає відповідних дозволів та доступу HTTPS [19].

Що стосується програмних вимог, Snyk CLI є основним інструментом для сканування коду. CLI можна встановити різними способами (npm, Homebrew, автономний бінарний файл тощо). Якщо встановлюється за допомогою npm, знадобиться остання версія Node.js runtime; в іншому випадку самостійні бінарні файли працюють на Windows, macOS, Linux або навіть Alpine. Після встановлення CLI перед скануванням необхідно пройти аутентифікацію у поточному обліковому записі Snyk (за допомогою snyk auth або API-токену). Для безперервної інтеграції зазвичай встановлюється змінна середовища (наприклад, SNYK_TOKEN) з API-токену Snyk. Крім стандартної робочої станції для розробки, спеціальне обладнання не потрібне; CLI є автономним. (Якщо потрібне локальне сканування, Snyk пропонував Local Engine, що вимагає кластера Kubernetes, але він є застарілим, і в більшості випадків використовується хмарне сканування SaaS, що надається Snyk). В нашому випадку буде використовуватись інтеграція з GitHub, всі показані скріншоти будуть відповідати цьому, однак буде розглянуто і сценарії локального використання – нижче наведені всі.

3.1.2 Встановлення

Для сканування репозиторію GitHub через веб-інтерфейс Snyk не потрібно встановлювати програму на кожну машину. Замість цього вмикається Snyk Code в налаштуваннях організації Snyk та імпортується репозиторії. У налаштуваннях веб-інтерфейсу Snyk → Snyk Code переведемо Snyk Code в стан «Enabled» (Увімкнено).

Потім інтегруємо власну SCM: перейдемо до Налаштування → Інтеграції → Контроль джерел і підключимо GitHub, GitLab тощо, надавши Snyk доступ до поточного облікового запису.

Після підключення імпортуємо потрібні репозиторії в Snyk; Snyk Code автоматично просканує код і створить проекти Snyk з результатами аналізу коду. (Імпортовані репозиторії слід імпортувати повторно, якщо Snyk Code було ввімкнено після початкового імпорту). Нижче наведено серію скріншотів процесу інтеграції SnykCode з наявним репозиторієм.

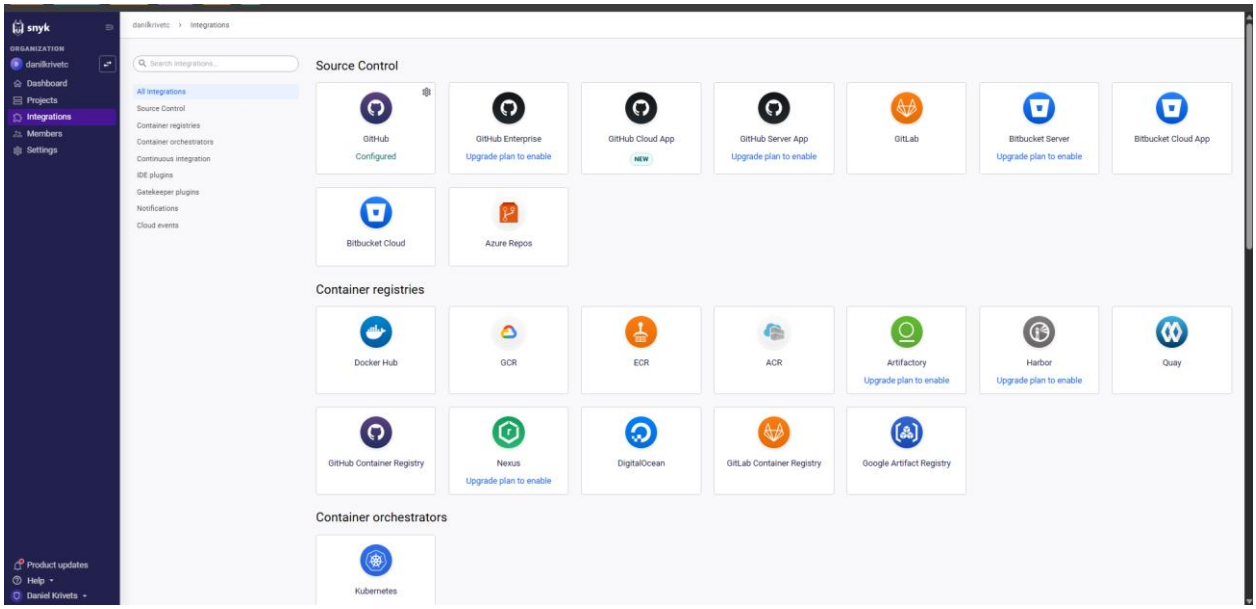

Рис. 3.1 Панель управління інтеграціями

Панель управління що зображена вище допомагає керувати поточними інтеграціями та надає широкий спектр можливостей для користувача ролі організатора.

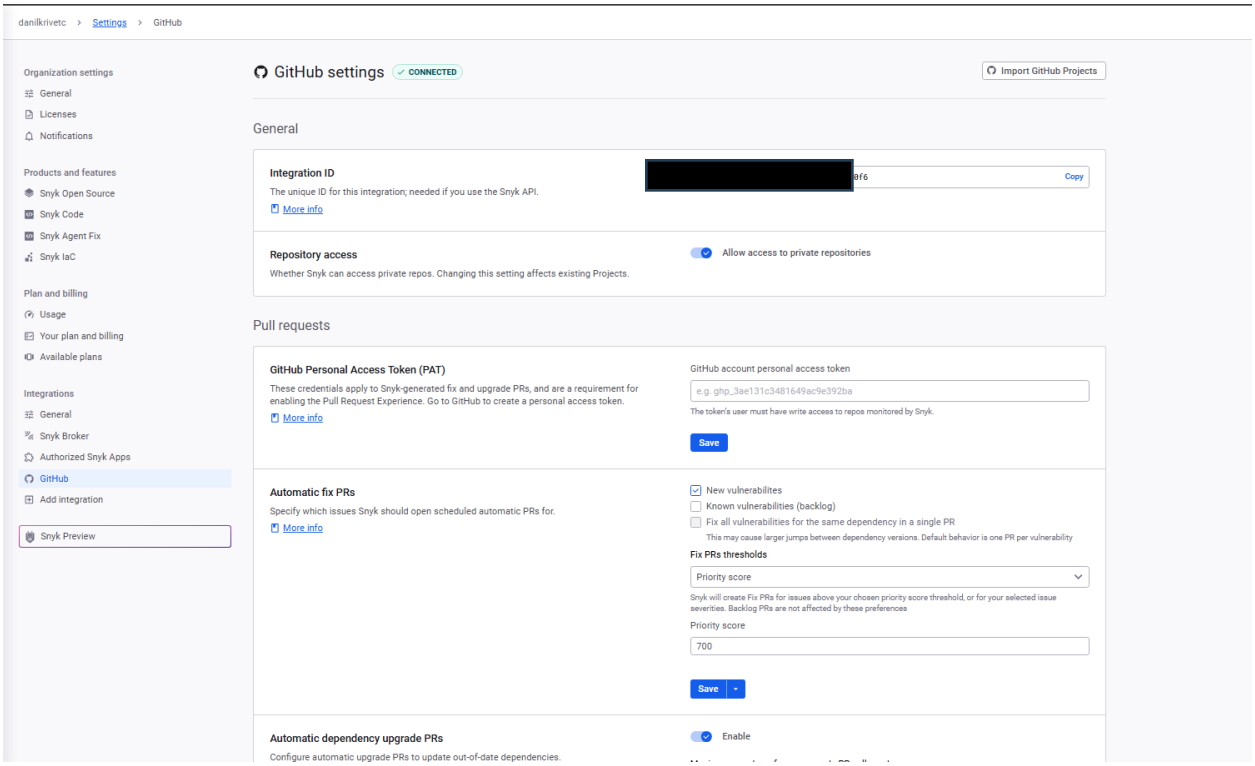

Рис. 3.2 Детальний огляд налаштувань інтеграції з GitHub частина 1

Як можна побачити ця панель керування дає нам чимало можливостей для конфігурацій. Значна частина впровадження Snyk Code для статичного тестування безпеки додатків у середовищах веб-розробки передбачає налаштування інтеграції з системою контролю версій.

Для більшості команд розробників GitHub залишається основною платформою управління вихідним кодом, а Snyk надає широкий спектр параметрів конфігурації для забезпечення точного виявлення вразливостей, автоматизованих робочих процесів виправлення та безперебійної роботи розробників. Коли інтеграція GitHub активована в робочому просторі Snyk організації, стають доступними кілька категорій налаштувань. Ці налаштування впливають на доступ до репозиторію, автоматизовані робочі процеси pull-request, логіку пріоритетності, управління залежностями та виконання перевірок аналізу коду.

Snyk дозволяє організаціям визначати дозволи для доступу до приватних репозиторіїв. Ця функція є необхідною для команд, які керують власними або закритими вебзастосунками. Конфігурація інтеграції GitHub включає перемикач, який визначає, чи може Snyk читати та аналізувати приватні репозиторії. Крім того,

організації можуть надати особистий токен доступу GitHub (PAT), щоб увімкнути запити на витяг, згенеровані Snyk. PAT повинен належати користувачеві GitHub з правами на запис у всіх репозиторіях, що контролюються. Цей токен необхідний для підтримки розширених можливостей, таких як автоматизовані запити на витяг виправлень та функції коментування запитів на витяг GitHub.

Після встановлення інтеграції Snyk може автоматично надсилати запити на виправлення вразливостей у репозиторіях, що перебувають під моніторингом. Параметри конфігурації дозволяють організаціям вибрати, чи повинен Snyk відкривати запити на виправлення тільки для нововиявлених проблем, для вразливостей із затримкою або для обох категорій.

Додаткові елементи керування визначають, скільки вразливостей можна виправити в рамках одного запиту на витягнення – наприклад, об'єднавши кілька виправлень для однієї і тієї ж залежності. Організації також можуть налаштувати порогові значення пріоритетності, вибравши мінімальний рівень серйозності або пріоритетності, необхідний для того, щоб Snyk відкривав автоматичні запити на виправлення. Ці параметри дозволяють командам розробників контролювати частоту оновлень, обсяг змін і стабільність робочого процесу розробки.

Окрім виправлення вразливостей, Snyk надає автоматизовані запити на оновлення залежностей. Ці запити створюються, коли в маніфестах проекту виявляються застарілі версії пакетів. Адміністратори можуть визначити максимальну кількість одночасно відкритих PR для оновлення, щоб запобігти захаращенню репозиторію, вказати залежності, які слід ігнорувати, та вибрати, чи слід включати оновлення основних версій до рекомендацій. Ця функція є особливо цінною для проектів вебзастосунків, які покладаються на часто оновлювані фреймворки, менеджери пакетів (npm, pip, Maven) або ланцюжки побудови фронтенду.

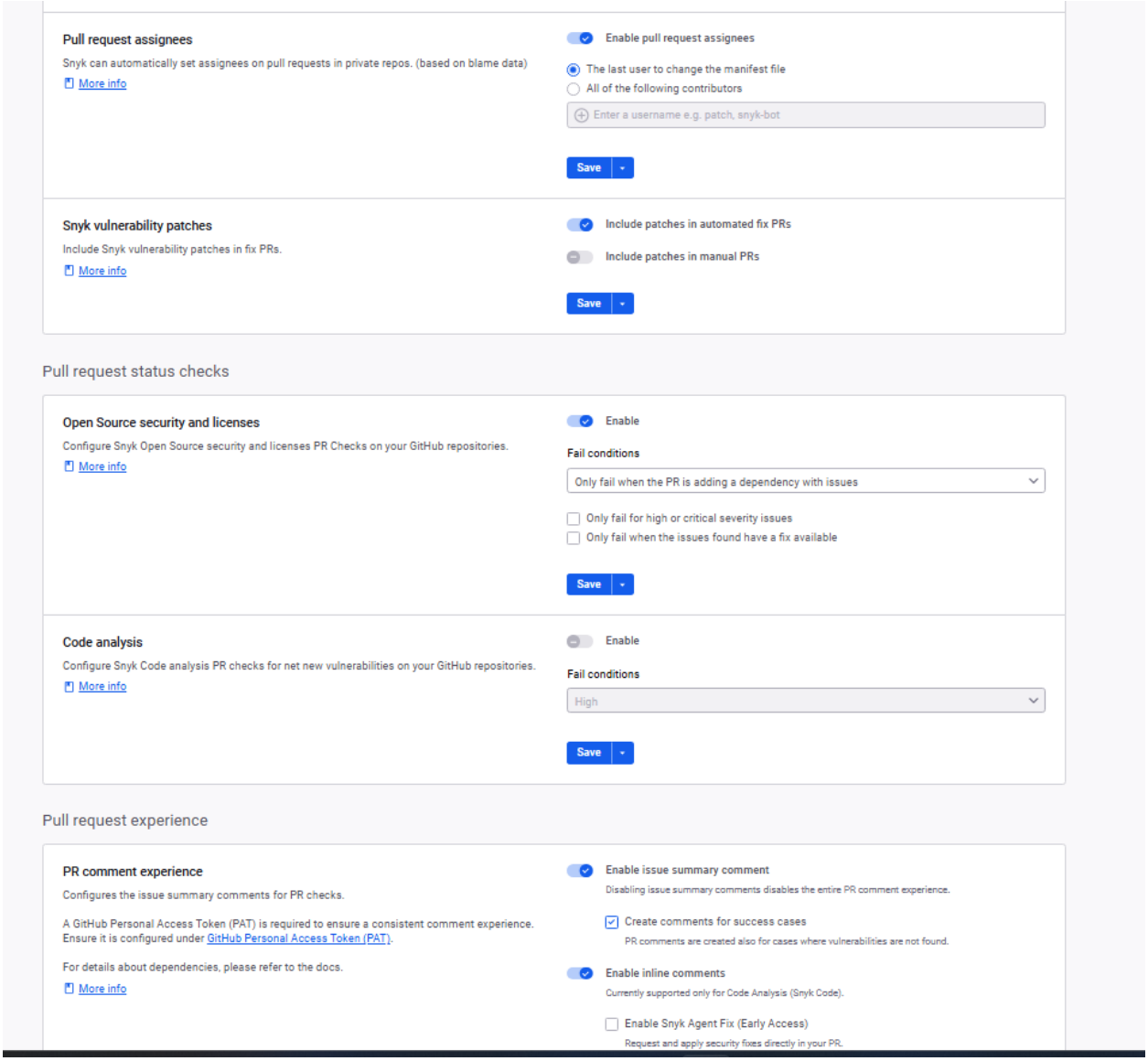

Рис. 3.3 Детальний огляд налаштувань інтеграції з GitHub частина 2

Для підтримки командних робочих процесів розробки Snyk може автоматично призначати розробникам запити на витяг. Система може використовувати аналіз на основі звинувачень, щоб ідентифікувати учасників, пов'язаних із відповідними файлами, або дозволити вибрати всіх учасників як можливих виконавців. Крім того, Snyk підтримує включення виправлень вразливостей Snyk безпосередньо в автоматизовані або запущені вручну запити на витяг. Ці виправлення забезпечують мінімальні корективи, призначені для нейтралізації конкретних вразливостей без необхідності повного оновлення залежностей.

Snyk інтегрує детальні перевірки стану в запити на витяг GitHub, щоб

запобігти появі нових вразливостей під час внесення змін до коду. Адміністратори можуть вмикати або вимикати перевірки, пов'язані з безпекою залежностей відкритого коду, проблемами ліцензування та статичним аналізом Snyk Code. Політики перевірки стану можна налаштувати, наприклад, відхиляти запити на витяг лише в тому випадку, якщо нова залежність створює вразливість, з'являються проблеми високого рівня серйозності або існує виправлення для виявленої проблеми. Спеціально для Snyk Code перевірки статусу PR можна налаштувати для оцінки нововведених вразливостей на рівні коду та блокування злиття, що не відповідають критеріям безпеки.

Snyk також забезпечує покращений досвід коментування запитів на витяг. Коли ця функція увімкнена, Snyk публікує підсумкові коментарі з описом виявлених проблем і може додавати вбудовані коментарі з точним зазначенням місць у коді, де виявлено вразливість. Ця функція значно покращує видимість для розробників шляхів потоку даних і векторів експлуатації, виявлених Snyk Code. Вбудовані коментарі наразі підтримуються лише для статичного аналізу Snyk Code. Крім того, команди можуть увімкнути функцію Snyk Agent Fix (ранній доступ), яка надає автоматичні пропозиції та іноді може генерувати готові до застосування виправлення безпосередньо в інтерфейсі запиту на витяг.

Як можна бачити Snyk надає багато можливостей для тонкого налаштування умов роботи команди, сканування та вза’ємодію Snyk Code та GitHub репозиторію. Нижче буде розглянуто процес інтеграції та додавання репозиторію для аналізу.

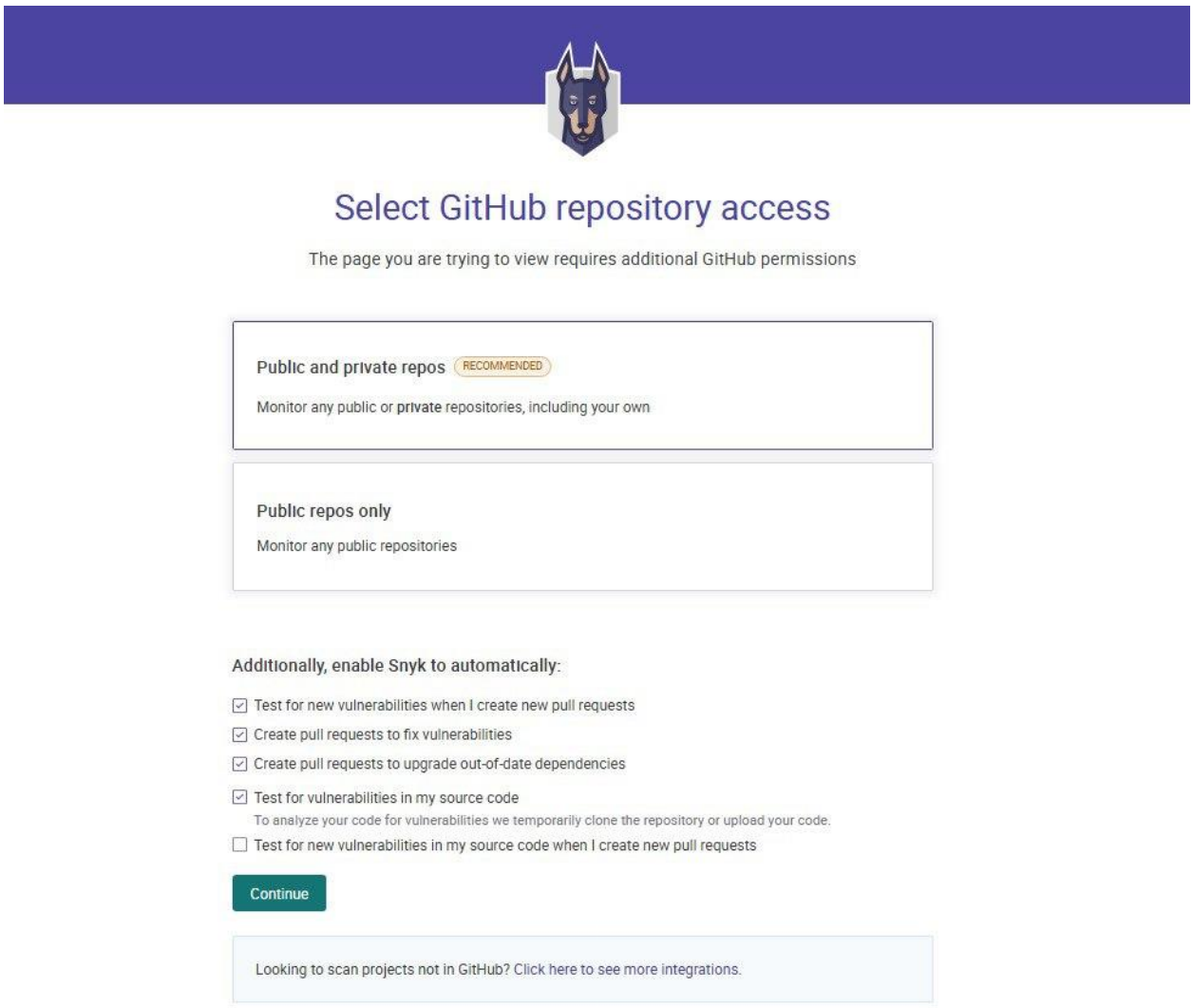

Рис. 3.1. Конфігурація сканування репозиторію

Важливо звернути увагу на пункт налаштування сканування SnykCode. Snyk дозволяє автоматичне сканування за рядом умов, нами був використаний сценарій автоматичного сканування після кожного push або pull запиту, тобто при кожній зміні в репозиторії вони будуть перевіятись одразу. За замовчуванням, після увімкнення, Snyk Code скануватиме кожне новоімпортовані репозиторій. Можна налаштувати, які файли включати або ігнорувати, за допомогою файлу політики .snyk у репозиторії (Snyk дотримується правил .gitignore та .dcignore). На сторінці «Аналіз коду» кожного проекту Snyk можна фільтрувати проблеми (за серйозністю, мовою тощо), сортувати за пріоритетом та переглядати приклади потоку даних або виправлень. Хоча ці налаштування інтерфейсу не вимагають додаткових кроків, вони впливають на те, як результати сканування

відображаються розробникам. Далі нам необхідно підтвердити інтеграцію з боку GitHub.

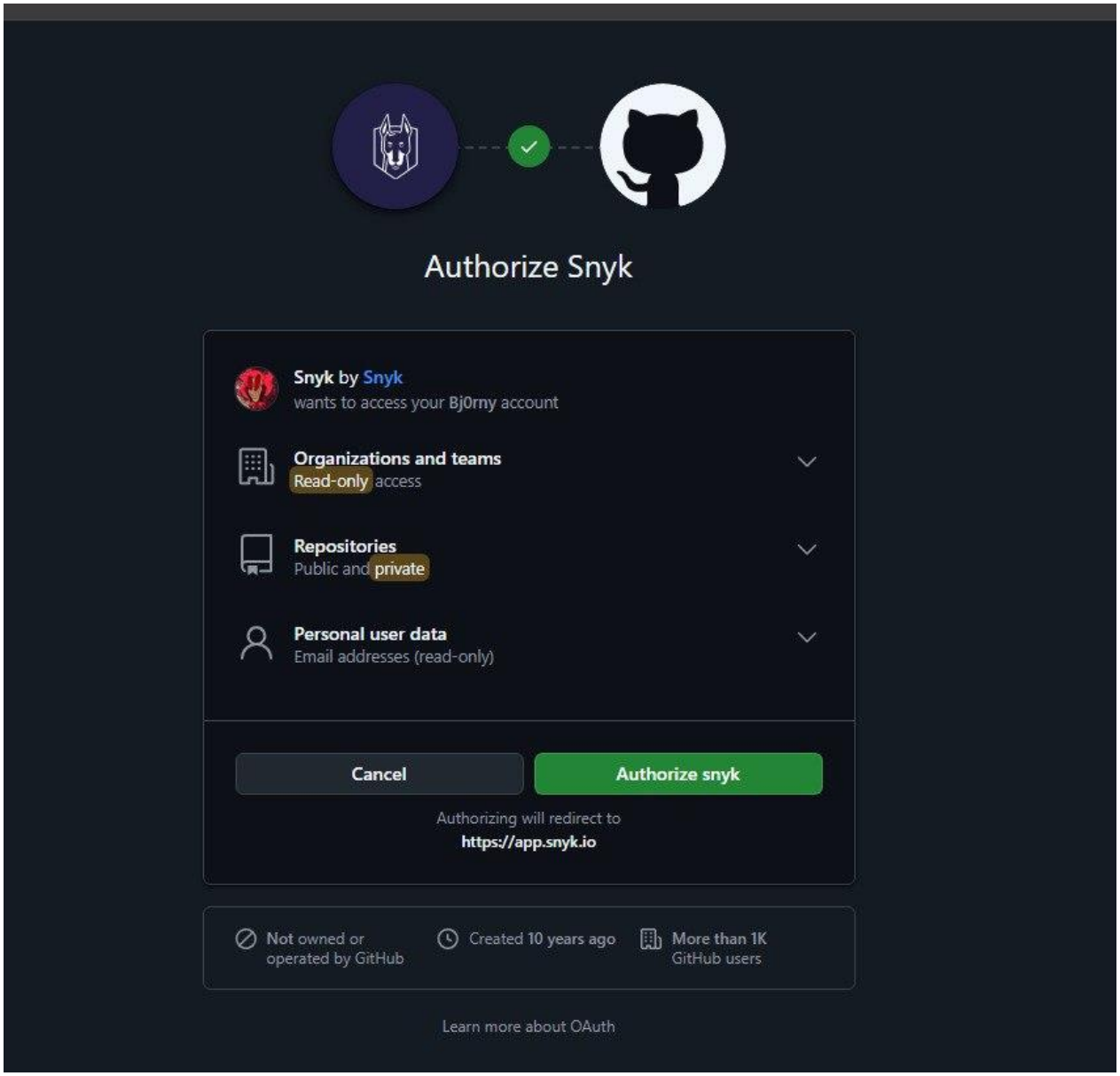

Рис. 3.2. Підтвердження інтеграції Snyk Code та GitHub

Після успішної ітеграції Snyk Code перевірить репозиторії акаунту та запропонує додати їх в якості проєктів. Після підтвердження та вибору репозиторію Snyk створить клон нашої репозиторії на своїй стороні для подальшого сканування.

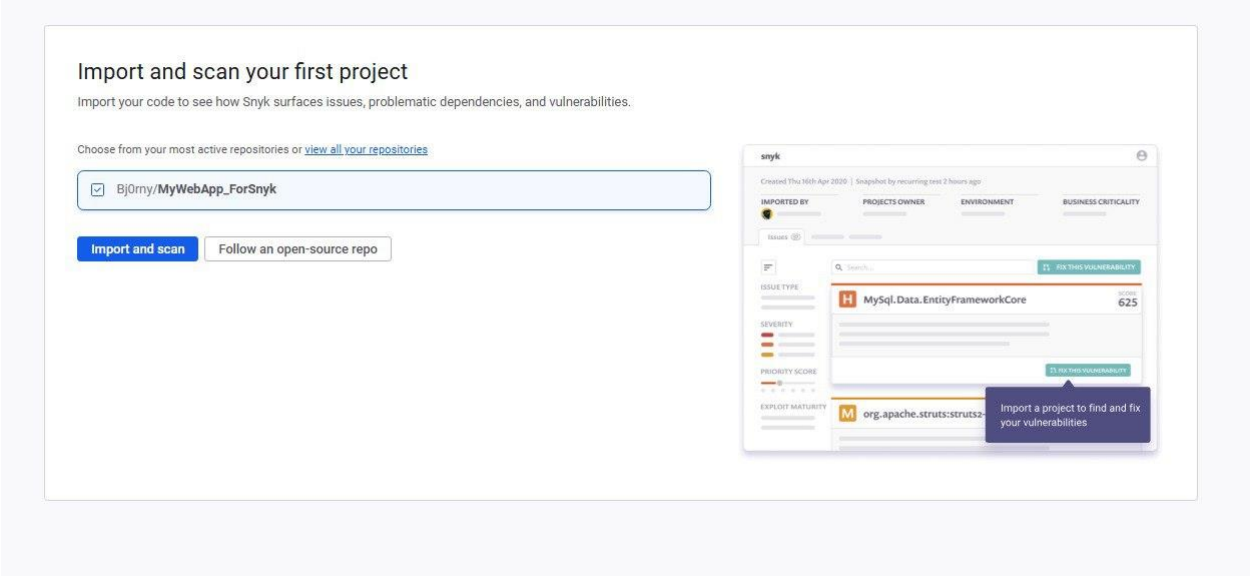

Рис. 3.3. Вибір репозиторію для проєкту

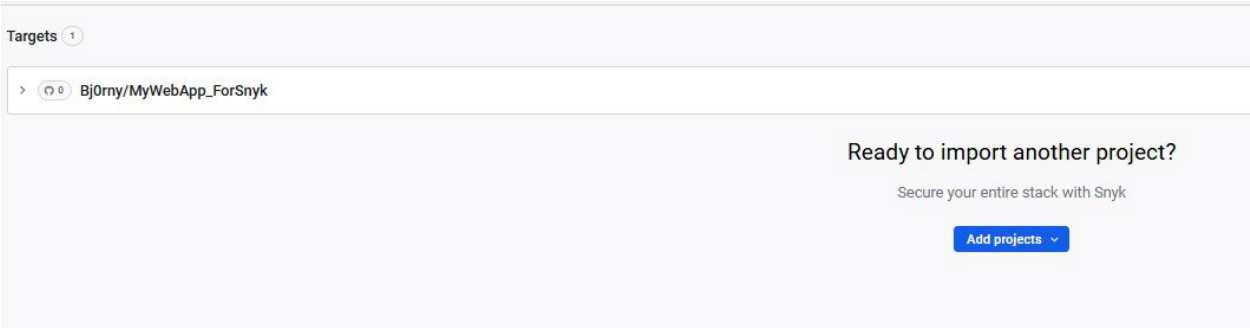

Рис. 3.4. Доданий репозиторій в проєкти Snyk Code

3.2 Застосування технології SAST у життєвому циклі ПЗ (з відображенням результатів)

Snyk Code розроблений для інтеграції в типові DevSecOps-процеси. Існує чотири основні точки інтеграції: плагіни IDE, Snyk CLI, перевірки SCM pull-request та Snyk Web UI/API. Розробники можуть використовувати CLI локально (наприклад, pre-commit hooks) або плагін IDE під час кодування; в CI/CD зазвичай викликають CLI як частину етапу побудови/тестування.

Як зазначено в документації Snyk, якщо певний плагін CI не підтримує Snyk Code (наприклад, деякі плагіни Jenkins наразі не підтримують), команди можуть

просто викликати Snyk CLI в пайплайні. Насправді Snyk рекомендує використовувати CLI для CI/CD, оскільки він забезпечує гнучкість (канали попереднього перегляду, часті випуски) [19].

Snyk забезпечує офіційну інтеграцію з багатьма системами CI/CD. Наприклад, робочий процес GitHub Actions може використовувати дію snyk/actions/setup, а потім дію snyk/snyk для автоматичного сканування при push-запитах або pull-запитах. У GitLab CI або Bitbucket Pipelines Snyk CLI можна додати аналогічним чином. У документації Snyk наведено докладні інструкції для Jenkins, Azure Pipelines, Bitbucket Pipelines, CircleCI, TeamCity тощо. Основна рекомендація від Snyk — використовувати CLI в CI для гнучкості (регулярні оновлення, канали попереднього перегляду).

Протягом усього процесу встановлення та інтеграції в дію вступають унікальні функції Snyk Code. На відміну від традиційних інструментів SAST, Snyk Code робить акцент на зручності використання для розробників: результати містять чіткі рекомендації щодо виправлення з прикладами коду. Він використовує велику базу знань про потоки коду (понад 25 мільйонів змодельованих потоків) та движок на базі штучного інтелекту для зменшення кількості помилкових спрацьовувань.

Для кожного виявленого результату інтерфейс Snyk може показати точний потік даних від джерела до приймача та надати посилання на подібні виправлення коду в відкритому коді. Результати сканування пріоритезуються за допомогою спеціального балу пріоритетності, який враховує легкість виправлення та ймовірність експлуатації, допомагаючи командам сфокусуватися на найризикованіших проблемах.

Інструмент працює дуже швидко (часто за лічені секунди), що відповідає робочим процесам у реальному часі та на вимогу. Розробники можуть навіть виправляти проблеми безпосередньо в своєму IDE за допомогою пропозицій одним кліком або використовувати механізм уразливостей відкритого коду Snyk для відомих шкідливих шаблонів. Коротко кажучи, Snyk Code повністю інтегрується в робочі процеси розробки та побудови: він сканує код при кожному PR та коміті, позначає проблеми в IDE та постійно синхронізується з вебзастосунком Snyk для

відстеження.

В рамках досліджуємого прикладу буде використано пайплайн роботи в віртуальному середовищі Codespace github. Як зазначалось в пункті 3.1 інтеграція була налаштована таким чином що кожен наш пул реквест буде перевірятись, що означає автоматичну перевірку з боку Snyk. Нижче наведено репозиторій веб застосунку, який був створений з відомими вразливостями, які має позначити Snyk. Основними файлами для роботи нашого веб додатку є server.js та routes/users.js.

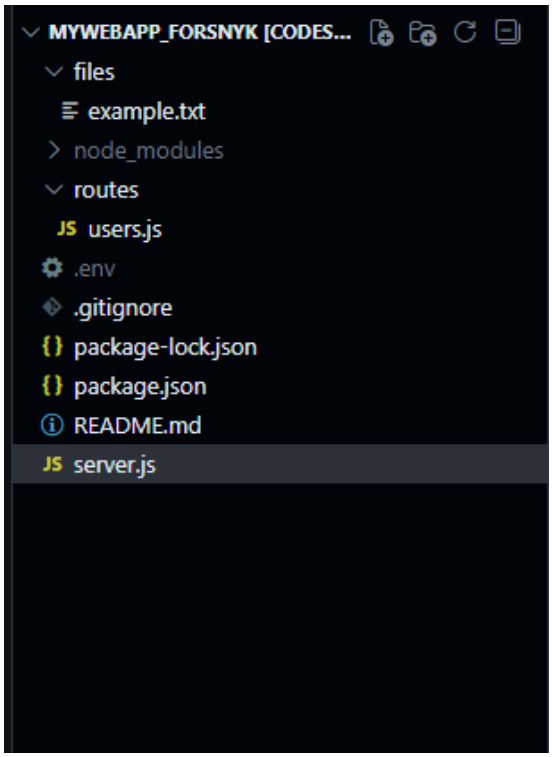

Рис. 3.5. Файли репозиторію вразливого веб застосунку.

Код для server.js:

```
const express = require('express');
const app = express();
const users = require('./routes/users');

// Hardcoded secret that Snyk will flag
const AWS_KEY = "AKIAIOSFODNN7EXAMPLE";

// Hardcoded private key
const PRIVATE_KEY = `-----BEGIN PRIVATE KEY-----
MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBALFakeKeyFakeKey
-----END PRIVATE KEY-----`;

app.use(express.json());

//Insecure HTTP only
```

```
app.get('/', (req, res) => {
    res.send('Vulnerable Test App is running!');
});

app.use('/users', users);

// No security headers (Helmet disabled)
app.listen(3000, () => {
    console.log('App running on http://localhost:3000');
});
```

Код для users.js:

```
const express = require('express');
const router = express.Router();
const fs = require('fs');
const { exec } = require('child_process');

// Fake DB
let users = [
    { id: 1, username: "admin", role: "admin" },
    { id: 2, username: "test", role: "user" }
];

// SQL Injection
router.get('/sql/:username', (req, res) => {
    const username = req.params.username;

    // Vulnerable SQL query pattern
    const query = \`SELECT * FROM users WHERE username = '\${username}'\`;

    res.send("Executed query: " + query);
});

// Command Injection
router.get('/ping/:host', (req, res) => {
    const host = req.params.host;

    exec(\`ping -c 1 \${host}\`, (err, output) => {
        if (err) return res.send("Error");
        res.send(output);
    });
});

// Path Traversal
router.get('/file/:name', (req, res) => {
    const name = req.params.name;

    const filePath = __dirname + '/../files/' + name;

    res.sendFile(filePath);
});

module.exports = router;
```

Після успішного завантаження проєкту до репозиторію Snyk виконав автоматичне сканування та визначив наявні вразливості у кодовій базі веб-застосунку. Проведений аналіз дозволив отримати загальну статистику знайдених проблем безпеки, а також перейти до детального дослідження кожної з них.

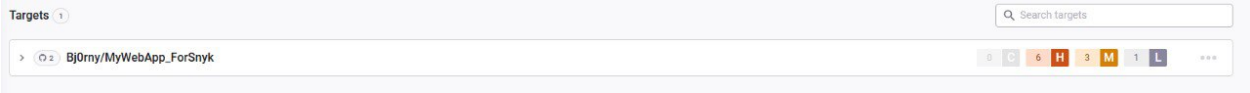

Рис. 3.6. Статистика виявлених вразливостей

На основі результатів сканування була виявлена низка потенційних ризиків, серед яких – доступність незашифрованих чутливих даних у файлах проєкту та можливість ін’єкції команд у веб-застосунок. Це підкреслює важливість використання автоматизованих інструментів статичного аналізу на ранніх етапах життєвого циклу програмного забезпечення.

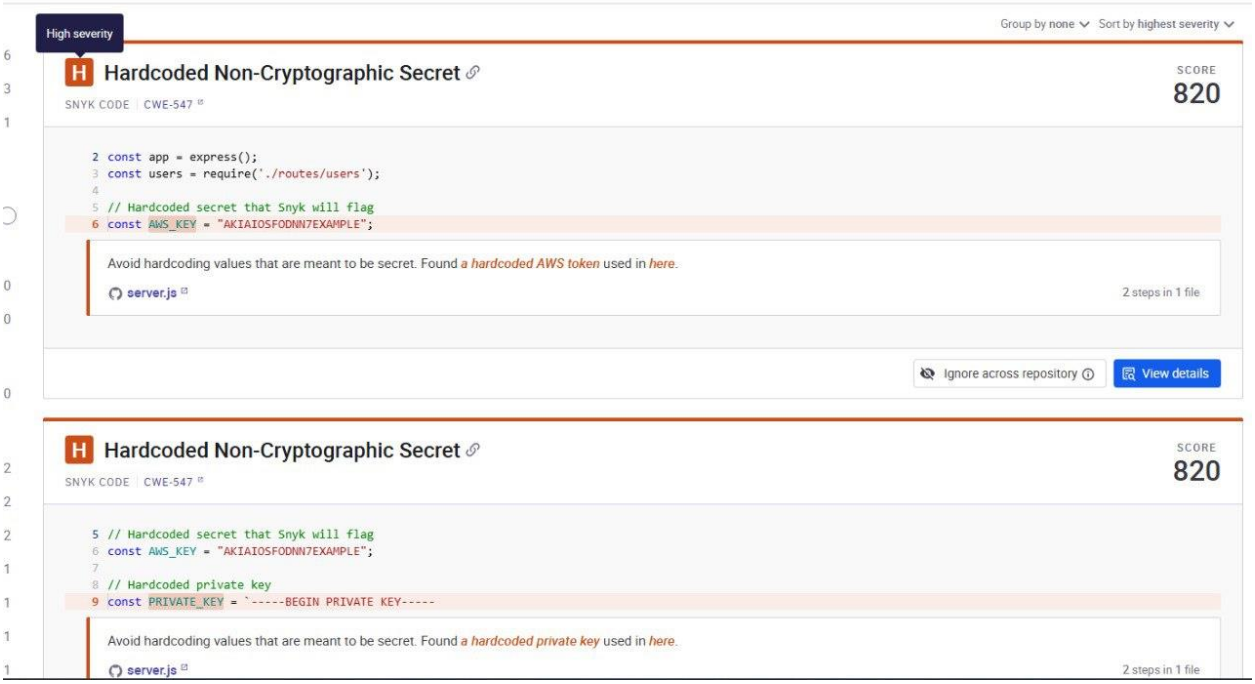

Рис. 3.7. Виявлені вразливості доступності чутливих даних в репозиторії

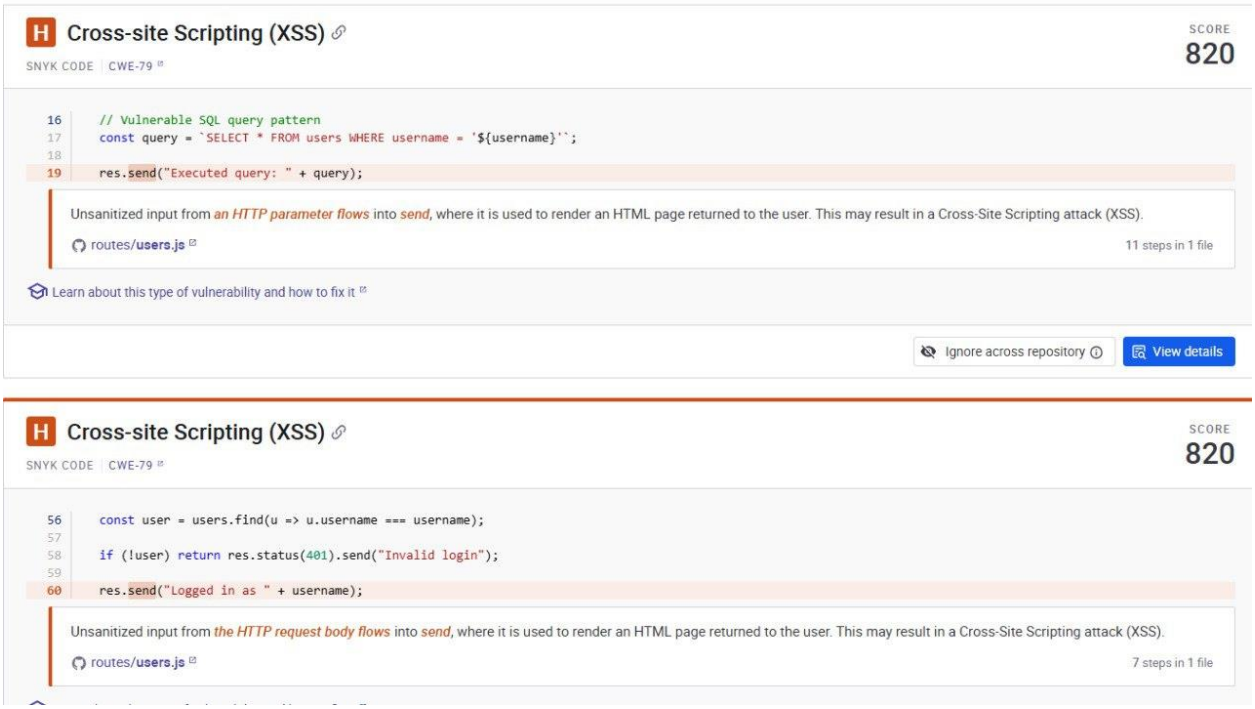

Рис. 3.8. Виявлені вразливості які дозволяють виконання несанкціонованих команд

Як видно з результатів, Snyk Code успішно ідентифікував вразливості та надав детальні рекомендації щодо їх усунення. Отримавши відповідні дані в рамках пайплайну, розробник може оперативно виконати необхідні зміни та повторно запустити перевірку. Інструмент також надає корисні посилання, пояснення природи вразливості, рівня ризику та можливих шляхів її усунення, що значно підвищує ефективність процесу виправлення.

Після внесення змін до застосунку та повторного сканування спостерігається суттєве покращення показників безпеки, що підтверджує результативність застосованих рекомендацій. Відсутність критичних та високоризикових вразливостей свідчить про підвищення рівня захищеності веб-застосунку та успішне інтегрування SAST-практик у процес розробки.

Фінальний код для server.js:

```
const express = require('express');
const helmet = require('helmet');
const users = require('./routes/users');

const app = express();

app.use(express.json());
```

```
app.use(helmet());
app.disable('x-powered-by');

app.use('/api', users);

app.listen(3000, () => console.log("Server running on port 3000"));
```

Фінальний код для users.js:

```
const express = require('express');
const router = express.Router();

// Example users (replace with DB later)
const users = [
  { username: "admin", password: "1234" },
  { username: "john", password: "abcd" }
];

// Login endpoint
router.post('/login', (req, res) => {
  const { username, password } = req.body;

  // Validate input
  if (typeof username !== "string" || typeof password !== "string") {
    return res.status(400).json({ error: "Invalid input format" });
  }

  const user = users.find(u => u.username === username && u.password ===
password);

  if (!user) {
    return res.status(401).json({ error: "Invalid credentials" });
  }

  // Safe JSON response
  return res.json({
    message: `Logged in as ${user.username}`
  });
});

module.exports = router;
```

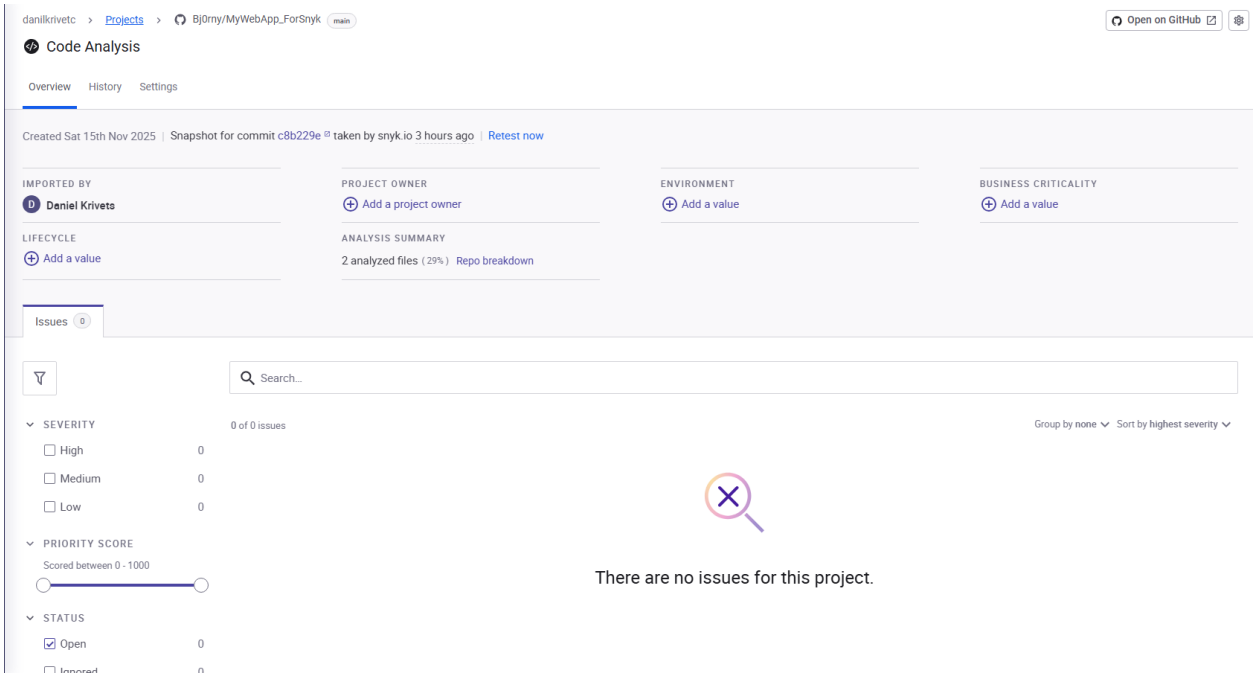

Рис. 3.9. Фінальний результат сканування вразливого репозиторію після внесення змін виправлення

Здійснена інтеграція Snyk Code та подальший аналіз результатів продемонстрували практичну застосовність технології SAST у реальних умовах розробки. Проведені дії засвідчили, що включення статичного аналізу на ранніх етапах життєвого циклу програмного забезпечення дає змогу оперативно виявляти та усувати вразливості ще до переходу системи в експлуатацію. Отримані результати підтвердили ефективність поєднання автоматизованих механізмів перевірки з робочими процесами DevSecOps та створили підґрунтя для подальшого розгляду практик безперервного підвищення рівня безпеки програмного забезпечення у наступних підрозділах.

3.3 Рекомендації щодо застосування технології статичного аналізу коду для виявлення вразливостей у вебзастосунках на базі рішення Snyk Code

Застосування інструментів статичного тестування безпеки додатків (SAST) стало важливою складовою безпечної розробки програмного забезпечення. На основі аналізу вразливостей вебзастосунків згідно розділу 1 «Дослідження

Проблеми Виявлення Вразливостей У Вебзастосунках» та оцінки методів виявлення згідно розділу 2 «Аналіз Методів Виявлення Та Запобігання Вразливостям У Вебзастосунках» а також практичного впровадження Snyk Code згідно пунктів 3.1 та 3.2, у цьому розділі наведено рекомендації щодо ефективної інтеграції статичного аналізу коду в життєвий цикл розробки програмного забезпечення (SDLC). Рекомендації адаптовані до можливостей Snyk Code і стосуються організаційних, технічних та процедурних аспектів.

Інтеграція Snyk Code у робочий процес розробки

Статичний аналіз коду є найбільш ефективним, коли він безпосередньо вбудований у повсякденну діяльність розробників. Snyk Code не слід розглядати як окремий інструмент для періодичного аудиту, а як постійний засіб захисту, інтегрований у життєвий цикл розробки програмного забезпечення (SDLC).

Розробники повинні встановити плагіни Snyk Code до IDE інтегрованих середовищах розробки (IDE) який впроваджено в компанії або команді. Це дозволяє виявляти вразливості в момент кодування, що зменшує витрати на виправлення порівняно з усуненням проблем на пізніших етапах.

Важливо зауважити що ця рекомендація підходить і для будь якого іншого типу інтеграції, яку надає Snyk, оскільки її суть це автоматичне завантаження бажаного коду на перевірку. Snyk Code необхідно налаштувати на автоматичний запуск під час етапів побудови та розгортання. Це гарантує, що небезпечний код не потрапить непоміченим у виробництво.

Автоматизовані перевірки діють як «ворота безпеки», що забезпечують дотримання організаційних стандартів. Організації повинні визначити правила, які блокують побудову, якщо виявлено критичні вразливості. Це запобігає випуску небезпечних версій і забезпечує відповідальність усіх команд.

Пріоритетність вразливостей

Не всі вразливості становлять однаковий рівень ризику. Ефективне використання Snyk Code вимагає стратегій пріоритетності. Узгодження вразливостей з категоріями OWASP та CWE забезпечує відповідність галузевим стандартам і допомагає командам зрозуміти ширший контекст кожної проблеми.

Окрім технічної серйозності, організації повинні оцінювати, як вразливості впливають на конфіденційність, цілісність та доступність систем. Наприклад, вразливість середньої серйозності в платіжній системі.

Різні галузі стикаються з різними ризиками, і рекомендації повинні відображати цю реальність. Наприклад, для фінансового сектору рекомендується надавати пріоритет криптографічним збоям та вразливостям ін'єкцій, оскільки вони безпосередньо впливають на цілісність та конфіденційність транзакцій.

Для електронної комерції краще зосередитися на порушенні контролю доступу та незахищених конфігураціях, оскільки несанкціонований доступ може поставити під загрозу дані клієнтів і призвести до фінансових втрат.

А для сфери охорони здоров'я рекомендується забезпечити суворе дотримання правил захисту даних шляхом впровадження TLS та безпечних методів зберігання. Дані пацієнтів повинні залишатися конфіденційними та захищеними від порушень. Це можна застосувати до кожної галузі окремо, тому рекомендується зрозуміти, які ризики є специфічними для вашого додатка або команди.

Постійний моніторинг та оновлення

Безпека не є статичною, вразливості еволюціонують у міру появи нових загроз. Репозиторії слід сканувати за розкладом, щоб виявляти вразливості, що з'являються з часом. Правила виявлення Snyk Code повинні бути актуальними відповідно до списків OWASP та CWE, що постійно оновлюються. Застарілі правила можуть пропустити критичні вразливості. Пов'язування результатів Snyk Code з комітами забезпечує простежуваність, дозволяючи командам визначити, коли були введені вразливості та ким. Це сприяє підзвітності та ефективному усуненню вразливостей.

Навчання та інформування розробників

Сама по собі технологія не може гарантувати безпеку, відповідно компаніям варто приділити увагу навчанню власного персоналу, розробники повинні бути навчені та мотивовані ефективно використовувати інструменти.

Навчальні програми повинні акцентувати увагу на поширених вразливостях, таких як SQL-ін'єкція, XSS та небезпечна десеріалізація. Розробники повинні

розуміти, як Snyk Code виявляє ці проблеми та як їх усунути. Розробників слід заохочувати переглядати рекомендації Snyk Code та вчитися на прикладах виправлення. Це перетворює виявлення вразливостей на можливість для навчання. Організації можуть відстежувати зменшення вразливостей з часом та винагороджувати команди за поліпшення якості коду. Це сприяє формуванню культури усвідомлення безпеки та постійного вдосконалення.

Висновки до третього розділу

У третьому розділі були розглянуті практичні аспекти застосування технології статичного аналізу коду для виявлення вразливостей у вебзастосунках з використанням Snyk Code як основного рішення. Були проаналізовані вимоги до встановлення, конфігурації та інтеграції Snyk Code в життєвий цикл розробки програмного забезпечення, підкресливши його гнучкість і простоту впровадження для команд розробників. На основі проведеного дослідження були сформульовані практичні рекомендації щодо ефективного використання технології статичного аналізу коду в розробці вебзастосунків. Ці рекомендації наголошують на ранній інтеграції інструментів SAST, постійному моніторингу безпеки та узгодженні з практиками безпечної розробки. Результати підтверджують, що Snyk Code є ефективним рішенням для поліпшення стану безпеки вебзастосунків завдяки проактивному виявленню вразливостей.

ВИСНОВКИ

Ця кваліфікаційна робота була присвячена проблемі виявлення вразливостей у вебзастосунках та дослідженню застосування технології статичного аналізу коду як ефективного засобу безпеки. Дослідження підтвердило, що сучасні вебзастосунки в значній мірі піддаються ризикам безпеки через свою складність, швидкі цикли розробки та широке використання компонентів сторонніх розробників.

У дослідженні проаналізовано типові вразливості вебзастосунків на основі визнаних галузевих стандартів та класифікацій, підкресливши важливість раннього виявлення та запобігання. Комплексне порівняння статичних та динамічних методів тестування безпеки продемонструвало, що статичний аналіз відіграє вирішальну роль у виявленні вразливостей на рівні вихідного коду перед розгортанням.

Рішення Snyk Code було проаналізовано з точки зору архітектури, функціональності та можливостей інтеграції. Дослідження показало, що Snyk Code ефективно підтримує практики DevSecOps, забезпечуючи безперервний аналіз безпеки та надаючи розробникам практичні рекомендації щодо усунення недоліків. За результатами дослідження були розроблені рекомендації щодо застосування статичного тестування безпеки додатків у розробці вебзастосунків. Отримані результати підтверджують, що інтеграція Snyk Code в життєвий цикл розробки програмного забезпечення підвищує безпеку вебзастосунків, зменшує ризики, пов'язані з уразливістю, та сприяє розробці більш безпечних і надійних програмних систем.

ПЕРЕЛІК ПОСИЛАНЬ

1. Britannica. Web application. [Electronic resource] URL: https://www.britannica.com/topic/Web-application (дата звернення: 10.11.2025).

2. Facts and Factors. Study on Global Progressive Web Application Market Size to Hit US$ 7,600 Mn, at a CAGR of 34% by 2026. Facts and Factors. [Electronic resource] URL: https://www.fnfresearch.com/news/global-progressive-web-application-market-revenueprojected-around (дата звернення: 10.11.2025).

3. Amazon Web Services. What is a Web Application? [Electronic resource] URL: https://aws.amazon.com/what-is/web-application/ (дата звернення: 10.11.2025).

4. Bhatt, Tuhin. Web Applications Types: Examples, Benefits and Challenges. Intelivita (2025). [Electronic resource] URL: https://www.intelivita.com/blog/types-of-web-applications/ (дата звернення: 10.11.2025).

5. MITRE. Common Weakness Enumeration (CWE). [Electronic resource] URL: https://cwe.mitre.org/ (дата звернення: 10.11.2025).

6. OWASP. OWASP Top Ten Web Application Security Risks. [Electronic resource] URL: https://owasp.org/www-project-top-ten/ (дата звернення: 10.11.2025).

7. Fortinet. Follina – Another Zero Day Attack in the Wild. [Electronic resource] URL: https://www.fortinet.com/blog/threat-research/msdt-follina-vulnerability (дата звернення: 10.11.2025).

8. Rapid7. CVE 2022 30190 – Microsoft Support Diagnostic Tool Vulnerability. [Electronic resource] URL: https://www.rapid7.com/blog/post/2022/05/31/cve-2022-30190-follina/ (дата звернення: 10.11.2025).

9. Semperis. Takeaways from Zerologon: The Latest Domain Controller Attack. [Electronic resource] URL: https://www.semperis.com/blog/takeaways-from-zerologon-the-latest-domain-controller-attack/ (дата звернення: 10.11.2025).

10. SOCRadar. Most Exploited Vulnerabilities of 2023 and Beyond. [Electronic resource] URL: https://socradar.io/top-exploited-vulnerabilities/ (дата звернення: 10.11.2025).

11. . Microsoft. HAFNIUM Targeting Exchange Servers with 0 Day Exploits. [Electronic resource] URL: https://www.microsoft.com/security/blog/2021/03/02/hafnium-targeting-exchange-servers/ (дата звернення: 10.11.2025).

12. CircleCI. SAST vs DAST: What they are and when to use them [Electronic resource] URL: https://circleci.com/blog/sast-vs-dast-when-to-use-them/ (дата звернення: 10.11.2025).

13. Checkmarx. SAST vs. DAST: Comparing Appsec Testing Methods [Electronic resource] URL: https://checkmarx.com/learn/sast/sast-vs-dast/ (дата звернення: 10.11.2025).

14. GitHub. The architecture of SAST tools: An explainer for developers. [Electronic resource] URL: https://github.blog/enterprise-software/secure-software-development/the-architecture-of-sast-tools-an-explainer-for-developers/ (дата звернення: 10.11.2025).

15. Splunk. SAST vs. DAST vs. RASP: Comparing Application Security Testing Methods [Electronic resource] URL: https://www.splunk.com/en_us/blog/learn/sast-vs-dast.html (дата звернення: 26.08.2025).

16. Semgrep Official Documentation. Semgrep Code overview [Electronic resource] semgrep.dev. – URL: https://semgrep.dev/docs/semgrep-code/overview (дата звернення: 26.08.2025).

17. Padioleau Y., Jin E. Semgrep: a static analysis journey [Electronic resource] – Semgrep Blog (Nov. 9, 2021). – URL: https://semgrep.dev/blog/2021/semgrep-a-static-analysis-journey (дата звернення: 26.08.2025).

18. Snyk. Snyk Code – SAST Code Scanning Tool [Electronic resource] – snyk.io. – URL: https://snyk.io/product/snyk-code (дата звернення: 26.08.2025).

19. Snyk. Snyk Code. Snyk User Documentation. (2024). URL: https://docs.snyk.io/scan-with-snyk/snyk-code (дата звернення: 10.11.2025).