МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ДУДНИК ІГОР ОЛЕКСІЙОВИЧ

Допускається до захисту:
Завідувач кафедри
інформаційних технологій,
д.т.н., доцент,
_____Нескородєва Т. В.
«__»_____20__р.

РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ СТАТИЧНИХ АНАЛІЗАТОРІВ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ КОДУ

Спеціальність 125 Кібербезпека
Кваліфікаційна (бакалаврська) робота

Науковий керівник:
Барибін О. І.,
к. т. н., доцент кафедри
інформаційних технологій

_____
(підпис)

Оцінка:___/ ___/ _____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця 2022

АНОТАЦІЯ

Дудник І. О. Рекомендації щодо застосування статичних аналізаторів інформаційної безпеки коду Спеціальність 125 Кібербезпека. Освітня програма «Кібербезпека». Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У кваліфікаційній (бакалаврській) роботі на основі аналізу функціоналу та характеристики статичних аналізаторів коду в забезпеченні ІБ запропоновано та но основі оцінки із запропонованими критеріями показано, що такі статичні аналізатори, як: DeepSource, SonarQube та Cast можна рекомендувати як найбільш застосовні. За результатами тестування на основі модельного коду можна рекомендувати Deepsource як найкращий статичний аналізатор коду серед протестованих.

Ключові слова: інформаційна безпека, статичний аналіз коду, аналізатор коду.

39 с., 13 табл., 10 рис., 21 джерело.

Dudnik IO Recommendations for the use of static code information analyzers Specialty 125 Cybersecurity. Educational program "Cybersecurity". Vasil Stus Donetsk National University, Vinnytsia, 2022.

In the qualification (bachelor's) work based on the analysis of the functionality and characteristics of static code analyzers in IS provision, but based on the evaluation of the proposed criteria, it is shown that static analyzers such as DeepSource, SonarQube and Cast can be recommended as the most applicable. According to the results of model code testing, DeepSource can be recommended as the best static code analyzer among those tested.

Keywords: information security, static code analysis, code analyzer.

39 pages, 13 tables, 10 figures, 21 sources.

ЗМІСТ

ВСТУП

У сучасному світі існує багато різних додатків. При написані їх виникаю багато казусів , як помітних , так і неосяжних очами девелоперів. Із-за цього виникають багато багів , «дірок» у безпеці , сповільнення працездатності додатку та інше. На допомогу приходять статистичні аналізатори. З-за допомого статичних аналізаторів вдається заощадити на часі , та виявити недоробки\помилки та багато багів.

Статичний аналіз коду - це аналіз комп'ютерного програмного забезпечення, який виконується без фактичного виконання програм, створених із цього програмного забезпечення, як протилежне динамічному аналіз (тестування програмного забезпечення за допомогою виконання програм). [1]

Для більшості випадків аналіз виконується на деякій версії вихідного коду, а в інших випадках на деяких форма об'єктного коду. Термін зазвичай застосовується до аналізу, проведеного an автоматизований програмний інструмент, при цьому людський аналіз називається розумінням програми, розуміння програми або перевірка коду. Можна стверджувати, що програмні метрики та зворотний інжиніринг є формами статики аналізу, але таке обговорення не є метою даної роботи.

Метою даної роботи є формування оцінки та рекомендацій деяких статистичних аналізаторів.

Завдання роботи є:

- Аналіз функціоналу та характеристики статичних аналізаторів коду в забезпеченні ІБ.
- Оцінка статичних аналізаторів по запропонованому критерію оцінювання та формування рекомендації з їх використання.

Об’єкт дослідження –статичні аналізатори ІБ коду

Предмет дослідження –застосування статичних аналізаторів ІБ коду

РОЗДІЛ 1 ХАРАКТЕРИСТИКИ СТАТИЧНИХ АНАЛІЗАТОРІВ КОДУ В ЗАБЕЗПЕЧЕННІ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ КОДУ

1.1 Що таке статичний аналіз коду ?

Статичний аналіз коду - це аналіз комп'ютерного програмного забезпечення, який виконується без фактичного виконання програм, створених із цього програмного забезпечення, як протилежне динамічному аналіз (тестування програмного забезпечення за допомогою виконання програм).

Для більшості випадків аналіз виконується на деякій версії вихідного коду, а в інших випадках на деяких форма об'єктного коду.

Цей термін зазвичай застосовується до аналізу, що виконується за допомогою автоматизованого програмного засобу, при цьому людський аналіз називається розумінням програми, розуміння програми або перевірка коду. [2]

1.2 Які проблеми які визначає статичний аналізатор?

Статичний аналіз є потужним інструментом для забезпечення якості та надійності програмного забезпечення, який може виявити ряд проблем у коді перед виконанням.

Деякі з цих категорій проблем:

1)Потенційні вразливості безпеки

2)Ризики помилок і анти-шаблони

3)Порушення вказівок щодо стилю коду

4)Переповнення буферу

5)Проблеми з продуктивністю

6)Мертвий або невикористаний код

7)Уразливості безпеки

1.3 Аналіз для оптимізації програми

Компіляторам оптимізації необхідно знати багато різних властивостей компілюється програма для створення ефективного коду. Кілька прикладів такими властивостями є:

-Чи містить програма мертвий код, або, точніше, це функція f недоступний з головного? Якщо так, розмір коду можна зменшити.

-Чи однакове значення якогось виразу всередині циклу на кожній ітерації? Якщо так, вираз можна перемістити за межі циклу, щоб уникнути зайвого обчислення.

-Чи залежить значення змінної x від введення програми? Якщо ні, можна бути попередньо обчислений під час компіляції.

-Які нижня та верхня межі цілочисельної змінної x? Відповідь може керувати вибором представлення змінної під час виконання.

-Чи вказують p і q на непересічні структури даних у пам'яті? Це може дозволити паралельна обробка. [3]

1.4 Статичний та звичайний аналіз коду

Статичний аналіз коду значно швидше, ніж звичайне тестування, і може виявити будь-які дефекти, видимі в програмі вихідний код. Якщо інструменти для статичного аналізу коду порівняти з ручним оглядом коду розробниками програмного забезпечення, це можливо можна зробити висновок, що перегляд коду за допомогою інструменту також набагато швидший і ефективніший. Однак для статичного коду аналіз для виявлення будь-якого дефекту, він повинен бути видимий у вихідному коді. На додаток до кількості контексту, необхідного для ідентифікації дефекту, багато дефектів можна знайти лише в певному представлення програми. На рисунку 1 показана матриця, сформована за типом і видимістю дефекту.

Проблеми високого рівня часто помітні лише при розробці програми, тоді як помилки реалізації часто можуть бути помічені лише за допомогою перевірка вихідного коду програми. Об’єктно-орієнтовані мови, такі як Java, мають велику кількість бібліотек, що робить легше зрозуміти дизайн, дослідивши вихідний код.[4]

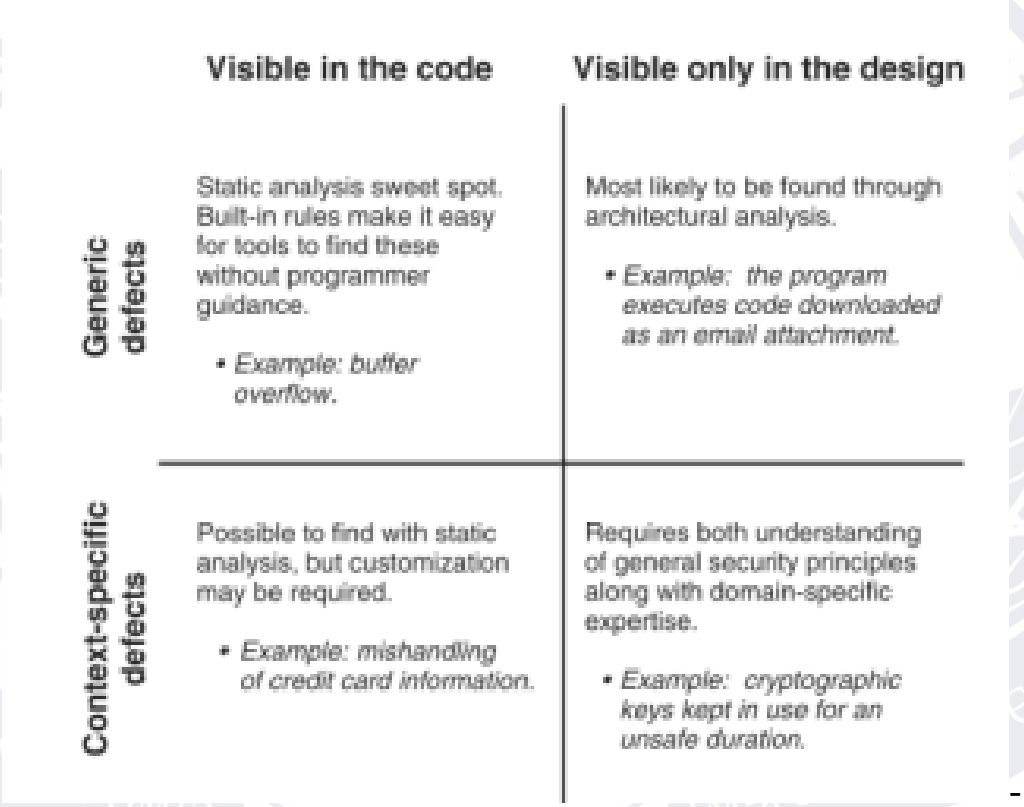

Рис 1 Загальні дефекти та контекстно-специфічні дефекти [2]

1.5 Синтаксичне дерево

Як правило, статичний аналіз виконується на вихідному коді програми за допомогою інструментів, які перетворюють програму в абстрактне синтаксичне дерево , щоб зрозуміти структуру коду, а потім знайти в ньому проблеми.

Абстрактне синтаксичне дерево - це деревовидне представлення вихідного коду комп’ютерної програми, яке передає структуру вихідного

коду. Кожен вузол у синтаксичному дереві представляє конструкцію, що зустрічається у вихідному коді.

Під час перетворення в його абстрактне синтаксичне дерево зберігаються лише структурні та пов’язані з вмістом деталі вихідного коду, а будь-які додаткові деталі відкидаються. Інформацією, яка зберігається і є життєво важливою для призначення синтаксичного дерева, є:

-Типи змінних і розташування кожного оголошення змінної

-Порядок і визначення виконуваних операторів

-Ліва і права складові двійкових операцій

-Ідентифікатори та присвоєні їм значення [5,6]

1.6 Різниця статичного та динамічного аналізу

Динамічний аналіз – це тестування та оцінка програми під час виконання.

Статичний аналіз — це тестування та оцінка програми шляхом перевірки коду без виконання програми.

Багато програмних дефектів, які викликають помилки пам’яті та потоків, можна виявити як динамічно, так і статично. Обидва підходи доповнюють один одного, оскільки жоден підхід не може знайти кожну помилку.

Основна перевага динамічного аналізу: він виявляє незначні дефекти або вразливості, причина яких занадто складна, щоб її можна було виявити за допомогою статичного аналізу. Динамічний аналіз може відігравати важливу роль у забезпеченні безпеки, але його головна мета — пошук і налагодження помилок.

Основна перевага статичного аналізу: він досліджує всі можливі шляхи виконання та значення змінних, а не лише ті, які викликаються під час виконання. Таким чином, статичний аналіз може виявити помилки, які

можуть проявитися лише через тижні, місяці чи роки після випуску. Цей аспект статичного аналізу є особливо цінним для забезпечення безпеки, оскільки атаки на безпеку часто використовують програму непередбаченими та неперевіреними способами.[7]

РОЗДІЛ 2 АНАЛІЗ СТАТИЧНИХ АНАЛІЗАТОРІВ.

2.1 Приклади статичних аналізаторів

Приклади додатків було обрано довільним випадком зі списків найкращих статичних аналізаторів.[8, 9]

А саме були обрані такі статичні аналізатори:

1)SonarQube

SonarQube - це програма для впровадження статистичного аналізу коду в процес розробки програмного забезпечення.

SonarQube дозволяє аналізувати код на неякісне оформлення, наявність багів, помилок та можливих проблем безпеки при написанні коду нової програми.

SonarQube підтримує такі мови програмування:

Java , Kotlin , C# . VB.net , C , C++ , JS, TS, PHP, Python, Terraform, CloudFormation, ABAP , APEX , COBOL , CSS , Flex ,GO ,HTML, Objective-C, COBOL, PL/I, PL\SQL, RPG, Ruby, Scala, Swift , T-SQL , VB6, XML

Функціонал:

Звичайний функціонал:

-Безпека додатків під керівництвом розробників. При виявленні недоліків, ненадійне введення користувача може бути виявлено в коді і оброблено до того, як він скомпрометує вашу програму.

-Звітність Отримайте інформацію про програму з точки зору операційних ризиків та ризиків безпеки.

-Висока розробка та маштабування. Висока доступність досягається за рахунок додавання надмірності до кожного сайту в системі. У поєднанні з функцією горизонтального масштабування Data Center Edition забезпечує швидку і надійну звітність з аналізу коду - навіть коли ваш екземпляр

розростається до глобальних масштабів, в якому розміщуються тисячі користувачів і проектів.

Функціонал стосовно безпеки:

-Гарячі точки (Огляд коду). Знаходить і переглядає гарячі точки безпеки (використання чутливого до безпеки коду)

-Уразливі місця (Зміна/виправлення коду). Автоматично виявляє вразливості (включаючи недоліки ін’єкції)

-Налаштування механізму аналізу SAST (Static Application Security Testing).

Якщо ви використовуєте пропрієтарні фреймворки для захоплення та/або збереження даних, що вводяться користувачем, версія Enterprise Edition дозволяє вам оголошувати їх у механізмі статичного тестування безпеки додатків (SAST). SonarQube гарантує, що таке введення даних перевіряться, перш ніж потрапити в критично важливі частини системи (база даних, файлова система, ОС тощо).

-OWASP / CWE Топ-25:

Спеціальні звіти для відстеження безпеки програм у порівнянні з категоріями стандартів OWASP і CWE Top 25 .

Скорочує цикл зворотного зв’язку про вразливості безпеки та допомагає розробникам швидше виправляти прориви в безпеці.

Експортуйте у форматі PDF найпопулярніші звіти.

Підтримувані компілятори:

-Будь-яка версія компіляторів Clang, GCC та Microsoft C/C++

-Будь-яка версія компілятора Intel для Linux та macOS

-Компілятори ARM5 і ARM6

-Компілятори IAR для ARM, Atmel AVR32, Atmel AVR, Renesas H8, Renesas RL78, Renesas RX, Renesas V850, Texas Instruments MSP430 та для 8051

-Компілятори QNX

-Компілятори Texas Instruments для Windows і macOS для ARM, C2000, C6000, C7000, MSP430 і PRU

-Компілятори Wind River Diab і GCC

-Також підтримуються компілятори, повністю засновані на GCC, включаючи, наприклад, Linaro GCC [10]

2)CodeScene

CodeScene - це інструмент аналізу поведінкового коду, надає візуалізацію коду на основі даних контролю версій і алгоритмів машинного навчання, які визначають соціальні закономірності та приховані ризики в коді.

Які мови підтримує: C,Clojure, CPP,C#,dart,elixir,erland,GO, groovy, java,JS,Kotlin,perl,php,powershell, python, ruby,scala, swift,typescript

Функціонал:

-Виявляє гарячі точки

- складний код, з яким організації доводиться часто працювати .

Гарячі точки бувають такими:

а)Мають низьке покриття коду

б) Мають низьку працездатність коду

в) Мають багато дефектів.

г) Визначає пріоритети технічного боргу на основі того, як розробники працюють з кодом.[11]

3) Embold

Embold - це простий, але ефективний статичний аналізатор коду на основі штучного інтелекту, який може допомогти розробникам аналізувати та покращувати свій код.

Здатний аналізувати вихідний код за чотирма напрямками:

-Проблеми з кодом.

-Проблеми дизайну.

-Показники дублювання

-Проблеми поверхні, які впливають на стабільність, надійність, безпеку та ремонтопридатність.

Оцінка Embold розраховується на основі цих чотирьох параметрів.

Має функцію, яка допомагає зрозуміти зони ризику та полегшує визначення пріоритетів проблем, які потрібно вирішити.

Він легко інтегрується з IDE, контролем версій і системами збирання.

Embold легко інтегрується з колишніми інструментами, які використовують команду розробників, як-от GitHub, Bitbucket і Azure, і отримує міттеві результати за допомогою автоматичного перегляду запитів на витяг.

Embolt підтримує такі мови програмування: C, C++, C#, Objective-C, TypeScript, JavaScript, Python, PHP, Go, Kotlin, Solidity, SQL, Swift(Cloud only), Ruby, Apex, HTML, CSS. [12]

4) CodeSonar

CodeSonar використовується для пошуку та виправлення помилок і вразливостей безпеки у вихідному та двійковому коді. Він виконує повнопрограмний міжпроцедурний аналіз з абстрактною інтерпретацією на C, C++, C#, Java, а також у двійкових виконуваних файлах і бібліотеках x86 і ARM.

CodeSonar зазвичай використовується командами, які розробляють або оцінюють програмне забезпечення, щоб відстежувати їхні недоліки якості або безпеки. CodeSonar підтримує хости Linux, BSD, FreeBSD, NetBSD, MacOS і Windows.

CodeSonar надає інформацію про кожну виявлену слабкість, включаючи трасування через вихідний код, який може викликати помилку, а також візуалізацію дерева викликів, яка відображає, як слабкість пов'язана з більш широким додатком.

CodeSonar підтримує такі стандарти функціональної безпеки:

IEC 61508, ISO 26262, DO-178B/C або ISO/IEC TS 17961.

Підтримувані мови програмування: C, C++, C#, Java, Python.

Також аналізує двійковий код та підтримує Intel x86-32, amd64 і ARM.

Підтримувані компілятори: Apple Xcode, ARM RealView, CodeWarrior, GNU C/C++, Green Hills Compiler, HI-TECH Compiler, IAR Compiler, Intel C++ Compiler, Microsoft Visual Studio, Renesas Compiler, Sun C/C++, Texas Instruments CodeComposer, Wind River. [13]

5) CAST

CAST - додаток аналізу коду спрямований на вирішення двох проблем. По-перше, більшість сучасних IT-систем складається з тисяч компонентів, створених кількома командами та десятками розробників. Вимірювання якості програмного забезпечення в цих системах вимагає ретельного калібрування між різними технологіями, випусками та чітким визначенням меж додатків.

Платформа аналізу додатків

Платформа аналізу додатків призначена для допомоги у виявлені дефектів або вразливостей програмного забезпечення з метою:

- Зменшити ризик
- Зменшити технічний борг
- Керуйте складністю
- Розмір застосування датчика
- Підвищити продуктивність розвитку
- Покращити якість програмного забезпечення

CAST AIP аналізує вихідний код, класифікуючи кожну бізнес-функцію на одиницю вимірювання. Це дозволяє швидше ідентифікувати знижену якість програмного забезпечення, уразливості системи та області, де продуктивність може бути покращена в складній багаторівневій інфраструктурі.

CAST Highlight – Highlight – це набагато легша технологія аналізу коду, яка не вимагає збирання вихідного коду в одному місці. Аналізований

на робочому столі розробника або менеджера проекту, вихідний код ніколи не переміщується в інше місце. Підсвічування потім об’єднує отримані показники розміру, ризику та складності на одній інформаційній панелі для перегляду портфеля.

Інструменти аналізу коду розробника – більшість розробників використовують статичні аналізатори, підключені до їхньої Visual Studio, Eclipse або іншої консолі IDE. Часто це інструменти з відкритим кодом, такі як FindBugs і PMD для Java. CAST AIP об'єднує результати будь-якого відкритого коду або власного набору інструментів аналізу коду в свої загальні панелі керування. Це представляє безперервний погляд на якість структурного коду протягом усього циклу розробки.

Інструменти, здатні класифікувати бізнес-функції як одиниці вимірювання, здатні виявляти дефекти в програмах, які важко визначити.

Інструменти аналізу коду пропонують унікальний погляд на складність, ризик і якість кожної програми.

Підтримує 50+ мов. [14]

6) PMD

PMD - це аналізатор вихідного коду. Він знаходить загальні недоліки програмування.

Він підтримує такі мови програмування: Salesforce.com, PLSQL, Apache Velocity, XML, XSL, Java, C, C++, C#, Groovy, PHP, Ruby, Fortran, JavaScript, PLSQL, Apache Velocity, Scala, Objective C, Matlab, Python, Go, Swift, Apex, Visualforce.

PMD може виявляти недоліки або можливі недоліки у вихідному коді, наприклад:

-Можливі помилки - пусті блоки.

-Мертвий код - невикористані локальні змінні , параметри та приватні методи.

-Порожні оператори if/while. Занадто складні вирази — непотрібні оператори if для циклів, які можуть бути циклами while.

-Неоптимальний код - марнотратне використання.

-Класи з високими вимірюваннями цикломатичної складності . - Повторюваний код - скопійований/вставлений код може означати скопійовані/вставлені помилки та зменшує ремонтопридатність.[15]

7) SourceMeter

SourceMeter - Інструмент, який допомагає аналізувати коди C/C++, Java, C#, RPG та Python.

Базова версія цього інструмента є безкоштовною, але вона має менше функцій.

У випадку C/C++ SourceMeter підтримує міжнародний стандарт ISO/IEC 14882:2011, доповнений декількома новими функціями з ISO/IEC 14882:2014, а також мову C, визначену ANSI/ISO 9899:1990, ISO/IEC 9899: 1999 та стандарти ISO/IEC 9899:2011.

Функціонал:

-Точний і глибокий статичний аналіз, побудова повних семантичних графів, що містять семантичні ребра (виклики, посилання), коментарі тощо.

-Більше 60 показників вихідного коду (складність, зв’язок, згуртованість, успадкування тощо) на різних рівнях (пакет, простір імен, клас, метод тощо)

-Дублювання типу 2 щодо меж синтаксису

-Показники дублювання коду (стабільність, вбудованість, дисперсія тощо)

-Виявлення непостійних змін дублювання

-Перевірка правил кодування (різниці між порушеннями правил PMD та порушеннями правил FaultHunter)

-Виявлення вразливостей безпеки на основі потоку даних (ін’єкція SQL, XSS тощо)

-Перевірка порушень правил на основі показників

-Перевірка певних порушень правил Android

-Виявлення винятків під час виконання за допомогою символічного виконання коду, лише для Java.[16]

8) Polyspace

Polyspace - це інструмент статичного аналізу коду для широкомасштабного аналізу шляхом абстрактної інтерпретації для виявлення або підтвердження відсутності певних помилок під час виконання у вихідному коді для мов програмування C, C++ та Ada.

Polyspace вивчає вихідний код, щоб визначити, де можуть виникнути потенційні помилки під час виконання, такі як:

-Арифметичне переповнення.

-Переповнення буфера.

-Поділ на нул

Інструмент також перевіряє вихідний код на відповідність відповідним стандартам коду.

Короткий опис дефект: неявне перетворення призводить до переповнення.

-Залучені типи даних : ціла змінна без знака (розмір: 32 біти) перетворюється на інший беззнаковий тип цілого числа (розмір: 16 біт).

-Очікувані значення : очікуваний діапазон цілого типу без знака (розмір: 16 біт) становить (0 ... 65535).

-Фактичні значення : фактичні значення, отримані змінною, вищі за очікувані.

-Пов’язані події : проблема з’являється в рядку 138. Інший рядок, що стосується проблеми, є призначенням у рядку 134. Пам’ятайте, що вам може знадобитися виправити помилку в попередньому рядку, а не в безпосередньому місці проблеми. [17]

9) PVS-Studio

PVS-Studio - це статичний аналізатор, який охороняє якість коду, безпеку (SAST) і безпеку коду

Аналізатор узгоджує попередження з загальним перерахуванням слабких місць, стандартами кодування SEI CERT і підтримує стандарт MISRA.

Має такий функціонал:

-Попередня обробка вихідних файлів C і C++ (на основі параметрів компіляції) дозволяє розширити директиви препроцесора, тобто включити заголовні файли та замінити макроси. Аналізатор використовує цю функцію для побудови найбільш повної семантичної моделі аналізованого коду.

- Аналіз на основі шаблонів, який базується на абстрактному синтаксичному дереві, шукає фрагменти у вихідному коді, які подібні до відомих шаблонів коду з помилкою.

- Анотації методів надають більше інформації про використовувані методи, ніж можна отримати, аналізуючи лише їхні підписи.

-Аналіз потоку даних використовується для оцінки обмежень, які накладаються на значення змінних під час обробки різних мовних конструкцій. Наприклад, аналіз потоку даних допомагає оцінити значення, які змінна може приймати всередині блоків if/else.

-Виведення типу , засноване на семантичній моделі програми, надає аналізатору повну інформацію про всі змінні та оператори в коді.

-Символічне виконання оцінює значення змінних, які можуть призвести до помилок, виконує перевірку діапазону значень.

-Аналіз зіпсованих даних виявляє випадки, коли програма використовує неперевірені дані користувача. Надмірна довіра до таких даних може спричинити вразливості (наприклад, SQLI, XSS, обхід шляху).

-Міжмодульний аналіз дозволяє діагностиці врахувати функції, оголошені в інших одиницях перекладу.

-Підтримує такі мови: C\C++

-Підтримує такі компілятори: GNU Arm Embedded Toolchain, Arm Embedded GCC компілятор, C, C++ CLion, Qt Creator, Eclipse, Clang, IntelliJ IDEA, Android Studio, Java Visual Studio, JetBrains Rider, C#, .NET Framework, .NET [18]

10) DeepSource

DeepSource - це швидка та надійна платформа статичного аналізу

Особливості:

- Розширений статичний аналіз Найбільша колекція правил статичного аналізу в галузі

- Центральна панель якості коду Центральний центр вашої команди для відстеження та вжиття заходів щодо справності коду

- Постійна перевірка якості коду Автоматично виконується під час кожного фіксації та витягування

- Трансформатори. Ставе форматування коду на автопілот. Не дозволяє своїй CI зламати через порушення стилю.

- Централізовано відстежуйте ключові показники коду

- Розширений аналіз IaC Виявляє проблеми безпеки та конфігурації для Terraform, Docker тощо.

- Відповідає OWASP Top 10

- Постійне сканування секретів Виявляє жорстко закодовані облікові дані безпеки та конфіденційні дані у вихідному коді.[19]

11) DeepCode : snyk

Ключовими технологічними відзнаками DeepCode є унікальна комбінація програмного аналізу, штучного інтелекту, представлення машинного навчання та великого коду, що пропонує:

-Нескінченне навчання від колективного мозку спільноти розробників

Незалежна від мови платформа, яка дозволяє додавати нові або навіть власні мови протягом кількох тижнів.

Швидкість: негайні результати. Немає потреби в компіляції або тривалій обробці: наш середній час аналізу репозиторію великого розміру

становить приблизно 5 секунд порівняно з нічними запусками, які пропонують аналізатори та лінтери.

Проблеми в будь-якій категорії, як тільки будь-який репозиторій з відкритим вихідним кодом виправить ту саму проблему. У ряді випадків ми автоматично визначаємо конкретну категорію для проблеми, але багато дефектів не можуть бути автоматично класифіковані. Ось деякі з категорій-прикладів, що зазвичай класифікуються:

-Зміни безпеки (наприклад, виправлення неправильно настроєного шифру)

-Виявлення неправильного використання API (загальна категорія, проблеми можуть призвести до проблем з користувачами, проблем з продуктивністю або просто потворного коду).

-Незначні помилки (наприклад, необхідно використовувати LinkedHashMap замість HashMap для збереження порядку).

-Стиль кодування (наприклад, перемикання з функцій зворотного виклику на лямбда-вираз).

Підтримує такі мови програмування : C#, Go, PHP, Java, JavaScript, Python, Ruby, TypeScript. [20]

2.2 Критерії та шкала оцінювання

Використаємо такі критерії оцінювання:

-Підтримка мов програмування

-Функціонал

-Підтримка компіляторів

-Доступність

-Середня оцінка

Оберемо таку шкалу , для подальшої оцінки:

Табл.. 2.1 Критерії оцінювання

Характеристика оцінки	Оцінка
Дуже добре	5
Добре	4
Влаштовує	3
Погано	2
Дуже погано	1

2.3 Оцінка статичних аналізаторів

1) SonarQube

Табл.. 2.2 Оцінювання SonarQube

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**5**
Функціонал	**4**
Підтримка компіляторів	**3**
Доступність	**4**
Середня оцінка	**4**

Підтримка мов програмування – у SonarQube доволі велика кількість підтримуваних мов програмування , тому оцінка 5.

Функціонал - доволі обширний функціонал , починаючи від пошуку неважливих багів , закінчуючи критичними вразливостями системи та багами.

Підтримка компіляторів: Підтримує компілятори , але багато з них вузького направлення

Доступність – в цілому програма безкоштовна , при чому ще є можливість подати заяву , для отримання доступу на 14-денне використання програми

Середня оцінка – (5+4+3+4)/4=4

2) CodeScene

Табл.. 2.3 Оцінювання CodeScene

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**4**
Функціонал	**3**
Підтримка компіляторів	**3**
Доступність	**3**
Середня оцінка	**3.25**

Підтримка мов програмування – підтримує достатню кількість мов програмування.

Функціонал - Функціонал непоганий , але даний статичний аналізатор не сканує проблеми безпеки

Підтримка компіляторів: Підтримує компілятори , але багато з них вузького направлення.

Доступність – програма є платною, але є можливість подати заяву , для отримання доступу на 30-денне пробне використання програми.

Середня оцінка – (4+3+3+3)/4=3.25

3) Embold

Табл. 2.4 Оцінювання Embold

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**1**
Функціонал	**5**
Підтримка компіляторів	**5**
Доступність	**3**
Середня оцінка	**3.75**

Підтримка мов програмування – дуже мала кількість мов програмування.

Функціонал - Функціонал непоганий , але даний статичний аналізатор не сканує проблеми безпеки

Підтримка компіляторів: Підтримує компілятори , але багато з них вузького направлення.

Доступність – програма є платною, але є можливість подати заяву , для отримання доступу пробне використання програми.

Середня оцінка – (4+3+3+3)/4=3.25

4) CodeSonar

Табл. 2.5 Оцінювання CodeSonar

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**2**
Функціонал	**4**
Підтримка компіляторів	**3**
Доступність	**2**
Середня оцінка	**2.75**

Підтримка мов програмування – підтримує малу кількість мов

Функціонал - Функціонал доволі непоганий , але вже трохи застарілий.

Підтримка компіляторів: Підтримує компілятори , але багато з них вузького направлення.

Доступність – програм є платною, можна отримати демо-версію , але це все не автоматизовано та щоб отримати доступ до демо-версії потрібно подати заявку , після чого з вами зв’жуться через 1-3 робочі дні , і тільки тоді буде відомо , чи отримаєте ви доступ , чи ні .

Середня оцінка – (2+4+3+2)/4=2.75

5) CAST

Табл. 2.6 Оцінювання CAST

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**5**
Функціонал	**3**
Підтримка компіляторів	**4**
Доступність	**3**
Середня оцінка	**4**

Підтримка мов програмування – підтримує багато мов (50+)

Функціонал - Функціонал непоганий , але даний статичний аналізатор не сканує проблеми безпеки

Підтримка компіляторів: Підтримує дуже багато компіляторів (100+)

Доступність – програма є платною, але є можливість подати заяву , для отримання доступу пробне використання програми.

Середня оцінка – (5+3+5+3)/4=4

6) PMD

Табл. 2.7 Оцінювання PMD

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**3**
Функціонал	**3**
Підтримка компіляторів	**3**
Доступність	**5**
Середня оцінка	**3.5**

Підтримка мов програмування – підтримує достатню кількість мов.

Функціонал - Функціонал непоганий , але даний статичний аналізатор не сканує проблеми безпеки

Підтримка компіляторів: Підтримує компілятори , але багато з них вузького направлення.

Доступність – програма безкоштовна.

Середня оцінка – (5+3+3+3)/4=3.5

7) SourceMeter

Табл. 2.8 Оцінювання SourceMeter

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**3**
Функціонал	**5**
Підтримка компіляторів	**3**
Доступність	**4**
Середня оцінка	**3.75**

Підтримка мов програмування – підтримує незначну кількість мов.

Функціонал - доволі обширний функціонал , починаючи від пошуку неважливих багів , закінчуючи критичними вразливостями системи та багами.

Підтримка компіляторів: Підтримує компілятори , але багато з них вузького направлення.

Доступність – базова версія програми безкоштовна, але основна та професіональна потребує грошей.

Середня оцінка – (3+5+3+4)/4=3.75

8) Polyspace

Табл. 2.9 Оцінювання Polyspace

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**4**
Функціонал	**4**
Підтримка компіляторів	**3**
Доступність	**4**
Середня оцінка	**3.75**

Підтримка мов програмування – підтримує незначну кількість мов.

Функціонал - доволі непоганий функціонал , починаючи від пошуку неважливих багів , закінчуючи критичними вразливостями системи та багами.

Підтримка компіляторів: Підтримує компілятори , але багато з них вузького направлення.

Доступність – базова версія програми безкоштовна, але основна та професіональна потребує грошей.

Середня оцінка – (4+4+3+4)/4=3.75

9) PVS-Studio

Табл. 2.10 Оцінювання PVS-Studio

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**1**
Функціонал	**5**
Підтримка компіляторів	**4**
Доступність	**4**
Середня оцінка	**3.5**

Підтримка мов програмування – підтримує незначну кількість мов.

Функціонал - доволі непоганий функціонал , починаючи від пошуку неважливих багів , закінчуючи критичними вразливостями системи та багами.

Підтримка компіляторів: підтримує в міру різновидних компіляторів

Доступність – безкоштовно , але не дуже зручне завантаження.

Середня оцінка – (1+5+4+4)/4=3.5

10) DeepSource

Табл. 2. Оцінювання DeepSource

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**4**
Функціонал	**5**
Підтримка компіляторів	**2**
Доступність	**5**
Середня оцінка	**4**

Підтримка мов програмування – доволі не мала кількість підтримуваних мов програмування.

Функціонал - доволі непоганий та великий функціонал , починаючи від пошуку неважливих багів , закінчуючи критичними вразливостями системи та багами.

Підтримка компіляторів: мала кількість підтримуваних компіляторів

Доступність – безкоштовно.

Середня оцінка – (4+5+2+5)/4=4

11) Deepcode : snyk

Табл. 2.12 Оцінювання Deepcode:snyk

Критерій оцінки	**Оцінка**
Підтримка мов програмування	**4**
Функціонал	**3**
Підтримка компіляторів	**2**
Доступність	**5**
Середня оцінка	**4**

Підтримка мов програмування – задовільна кількість підтримуваних мов програмування.

Функціонал - Функціонал непоганий , але даний статичний аналізатор не сканує проблеми безпеки

Підтримка компіляторів: мала кількість підтримуваних компіляторів

Доступність – безкоштовно.

Середня оцінка – (4+3+2+5)/4=3.5

2.4 Рейтинг статичних аналізаторів на основі оцінки

Табл. 2.13 Підсумковий рейтинг статичних аналізаторів

№	Назва Статичного Аналізатора	Середня Оцінка
1	**SonarQube, CAST, DeepSource**	4
2	**Embold, SourceMeter, Polyspace**	3,75
3	**PMD, PVS-Studio,Deepcode:snyk**	3,5
4	**CodeScene**	3,25
5	**CodeSonar**	2,75

РОЗДІЛ 3 РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ СТАТИЧНИХ АНАЛІЗАТОРІВ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ КОДУ .

3.1 Застосований код зі вразливостями та обрані статичні аналізатори

Для початку оберемо python код зі вразливостями sql-ін’єкцій [21]:

```
import sqlite3
from flask import request

def removing_product():
productId = request.args.get('productId')
str = 'DELETE FROM products WHERE productID = ' + productId
return str

def sql_injection():
connection = psycopg2.connect("dbname=test user=postgres")
cur = db.cursor()
query = removing_product()
cur.execute(query)
```

Оберемо під цей код декілька статичних аналізаторів , наприклад , візьмемо: Deepsource, Embold , Deepcode: snyx

3.2 Аналіз коду з-за допомогою Deepsource

1)Після аналізу коду програма вкаже усі недоліки коду

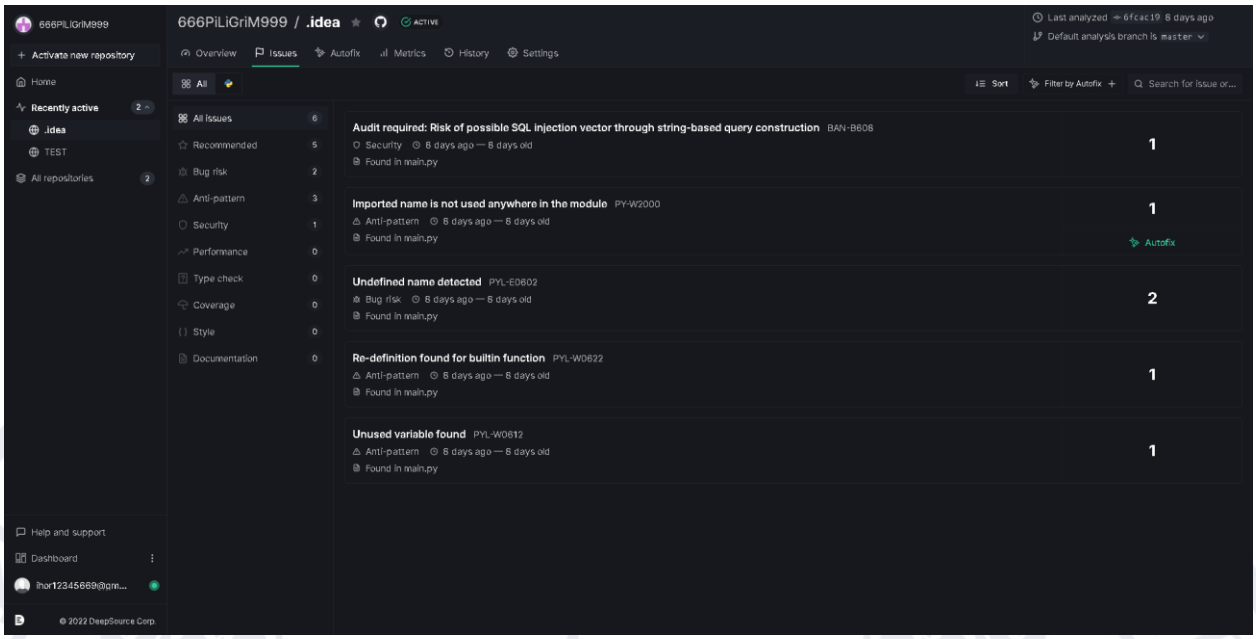

Рис. 3.1Список недоліків коду

2)Після закінчення сканування аналізатор описує проблему , та вказує в який частині коду ця проблема

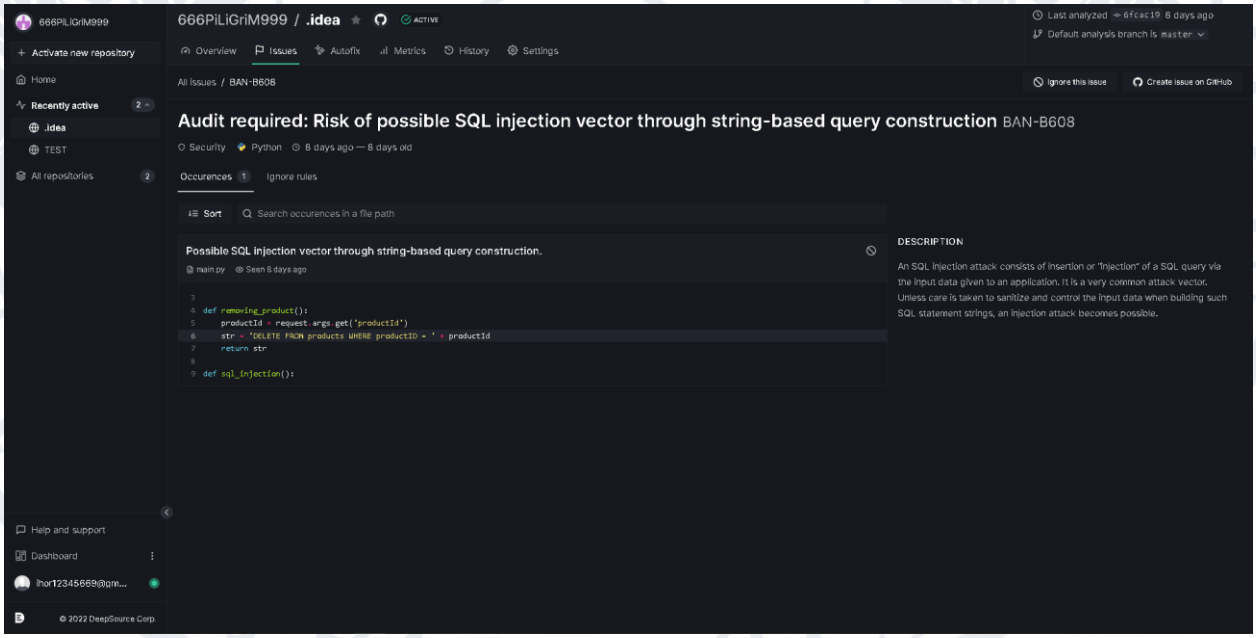

Рис. 3.2Опис проблеми та де саме вона виникла

3)Deepsource, при можливості, надає рекомендаці, щодо виправлення коду

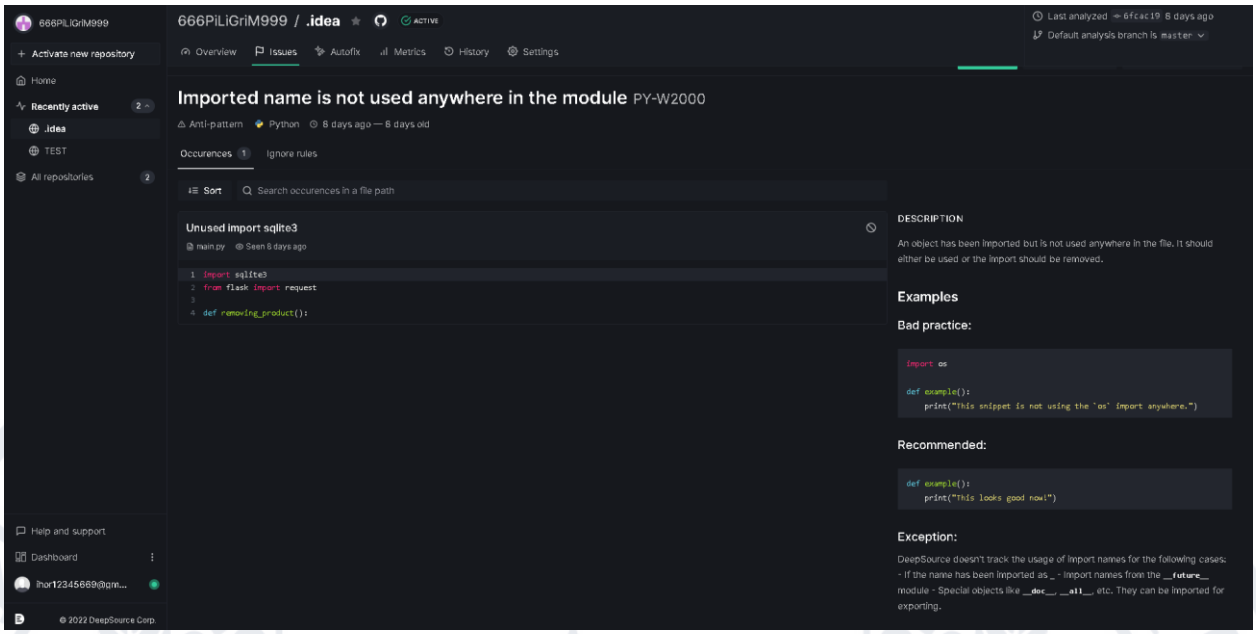

Рис. 3.3 Рекомендації, щодо виправлення вразливості

4)В данному статичному аналізаторі присутня функція автофіксу , що значно спрощує роботу розробнику , після автофіксу коду аналізатор дає можливість імпортувати вже відреадагований код у GitHub (авторедагування працює не в усіх випадках проблем).

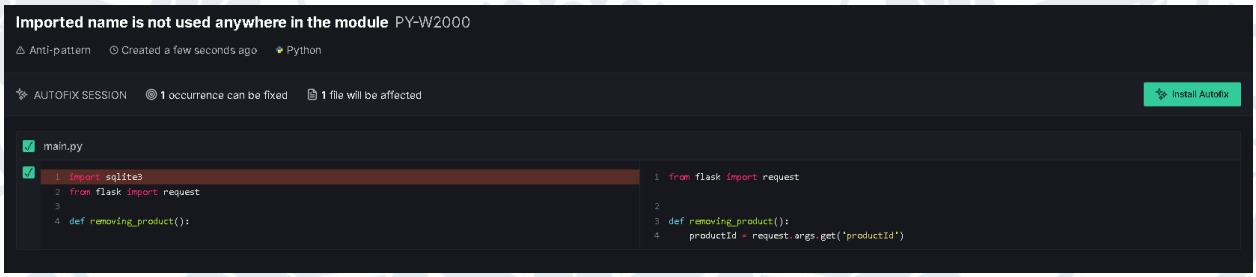

Рис. 3.4Опис проблеми та де саме вона виникла

3.3 Аналіз коду з-за допомогою Embold

1) Одразу після сканування коду ми побачемо зведення ключових показників ефективності

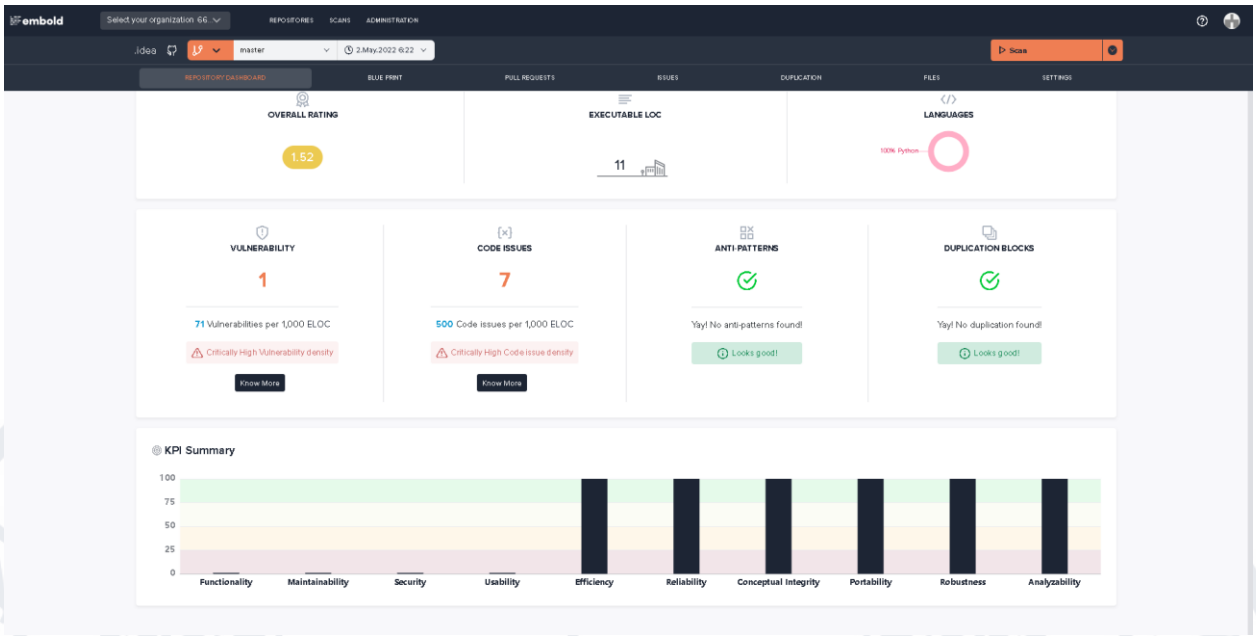

Рис. 3.5 Результати сканування та ключові показники ефективності

2) Після завершення аналізу коду Embold вкаже усі недоліки коду

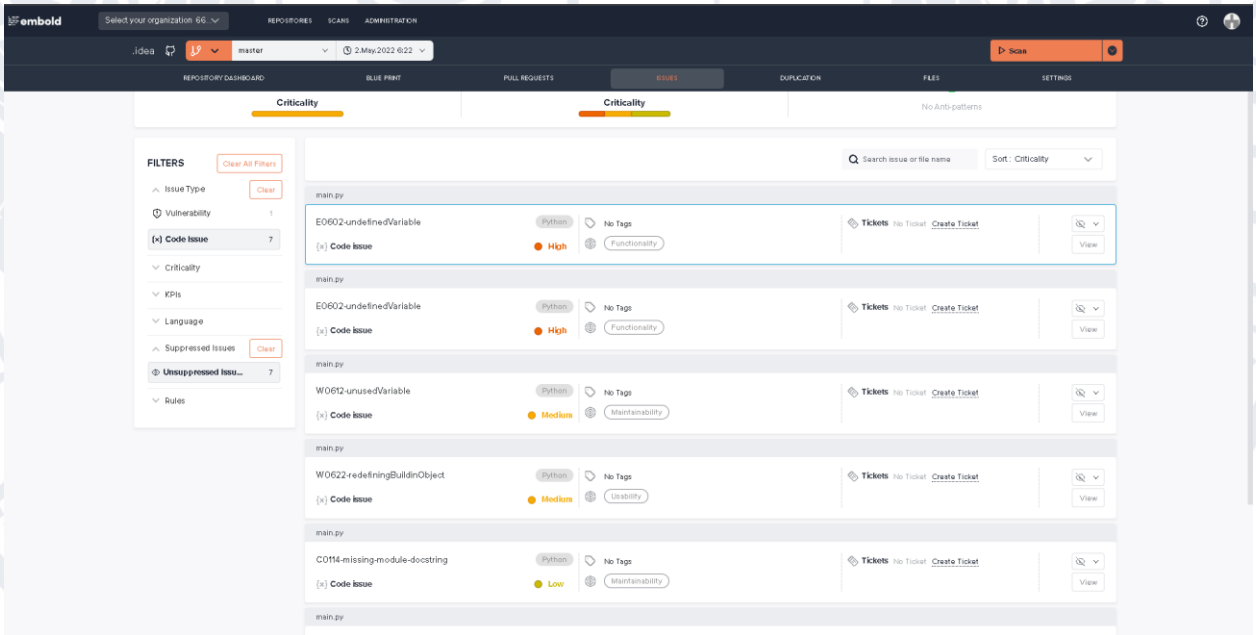

Рис. 3.6 Показ усіх недоліків коду

3) Після сканування ми можемо побачити проблему та в якій частині коду ця проблема

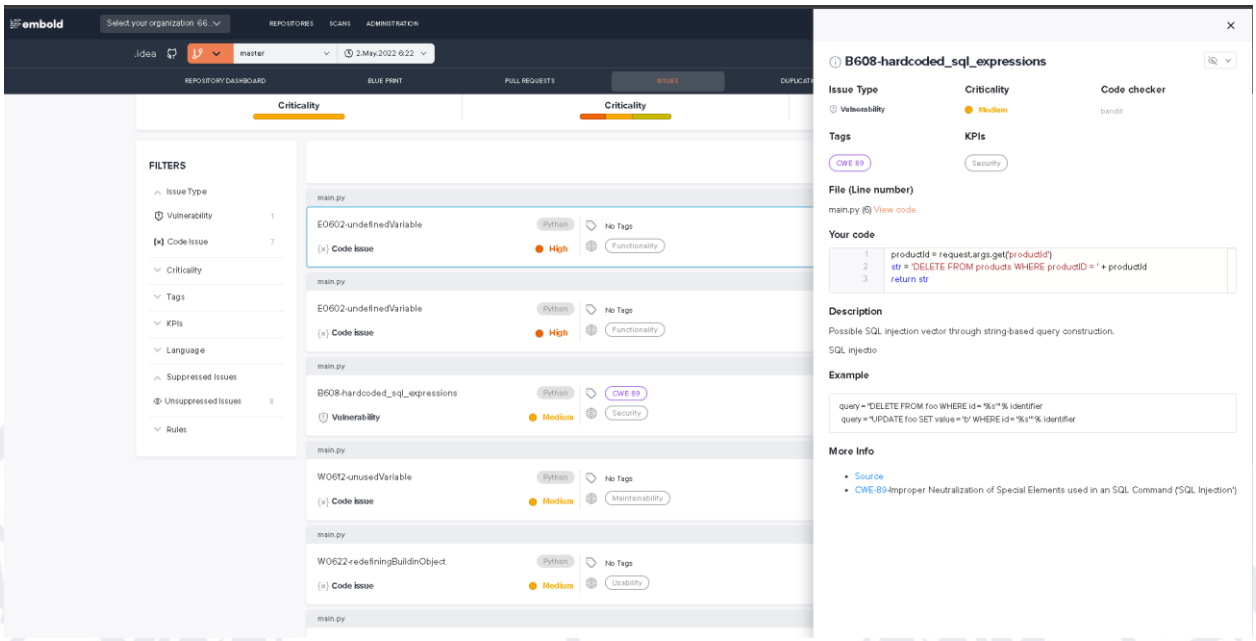

Рис. 3.7 Опис проблеми та де саме вона виникла

3.4 Аналіз коду з-за допомогою Deepcode : snyk

1)Після аналізу коду програма вкаже усі недоліки коду

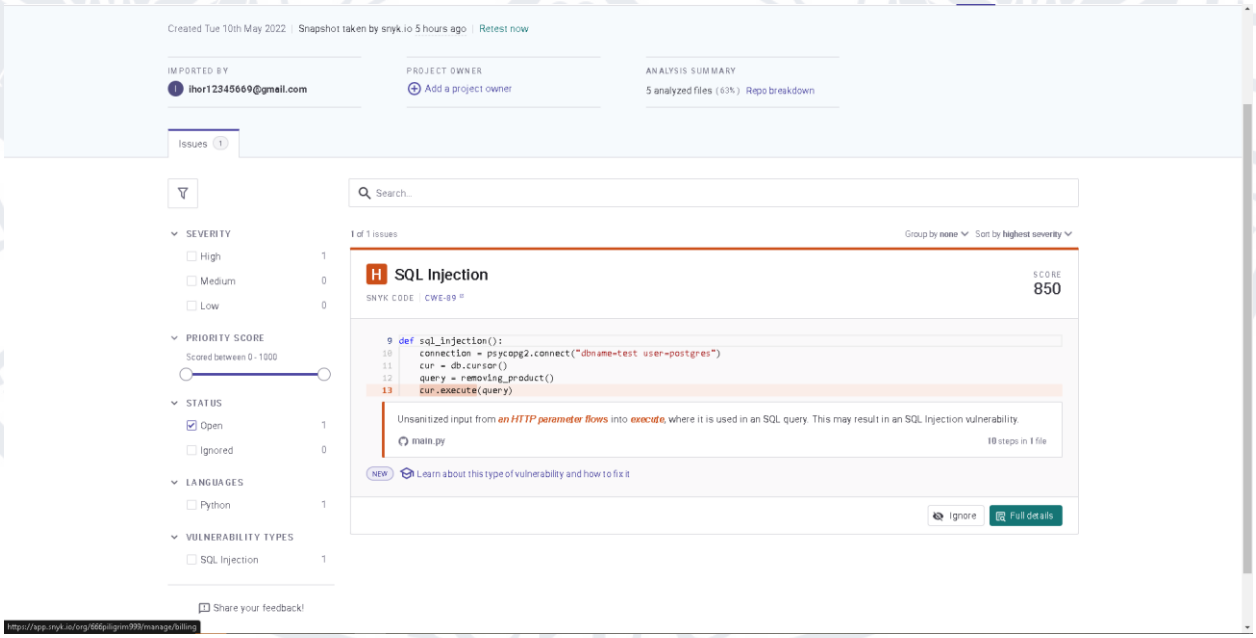

Рис. 3.8 Показ усіх недоліків коду

2)В даному аналізаторі доступна кнопка «Більш детально» . В цій функції можна подивитися більш детально інформацію про вразливість .

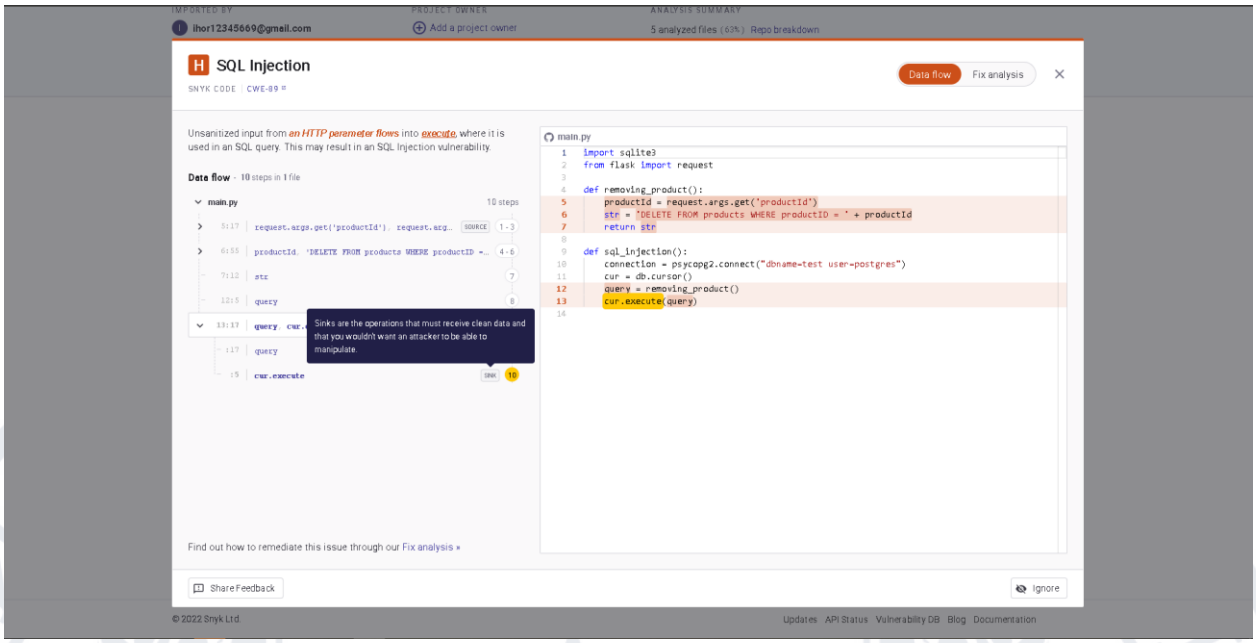

Рис. 3.9 Детальний опис проблеми , та в якій частині коду вона виникла

3)Також у функції «більш детально» можна подивитися детальний опис та методи профілактики, тако программа створює код-вирішення проблеми.

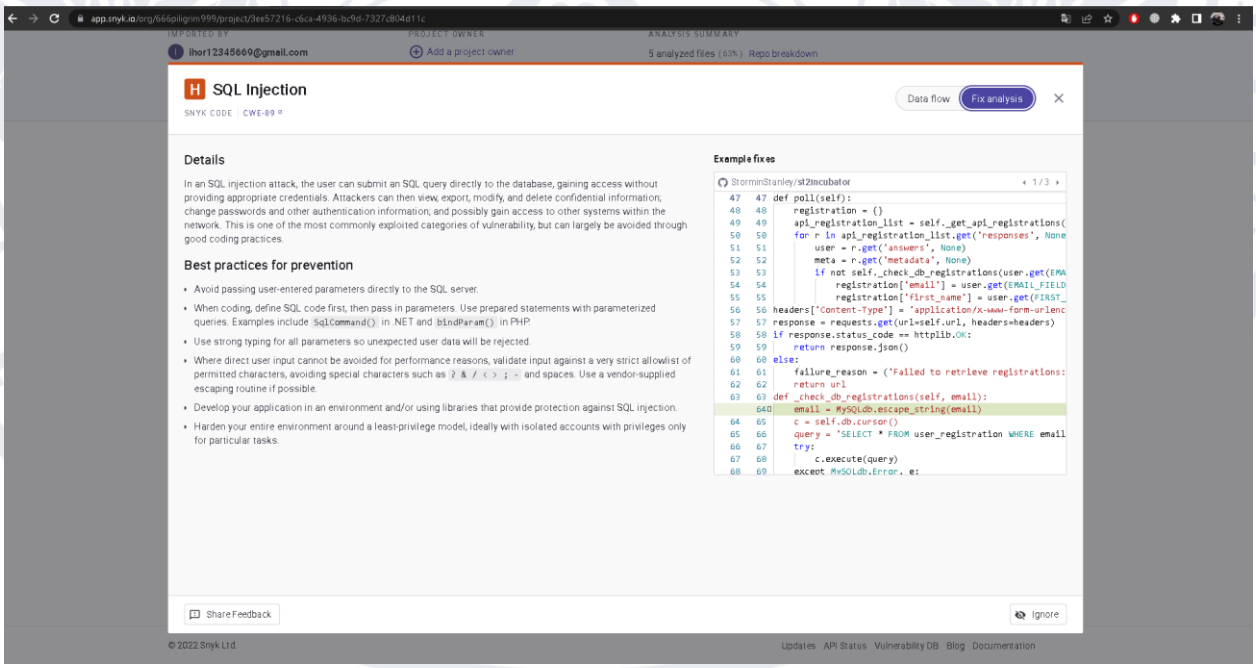

Рис. 3.9 Опис вразливості та спосіб як усунути її

3.5 Рекомендації статичних аналізаторів

У якості основи було використано програмний код зі вразливостями, у таких фреймворках, як : Deepsource, Embold, Deepcode:snyk.

Завершивши тестування деяких статичних аналізаторів, було з’ясовано про переваги статичних аналізаторів для програмних розробників.

У Deepsource ми побачили такі переваги:

- Зручний інтерфейс
- Автовиправлення коду (значно економить час розробника)
- Зручний та зрозумілий опис проблем
- Аналізатор вказує де саме в коді проблема , а саме відокремлює код, де виникла проблема.

У Embold ми побачили такі переваги:

- Зручний інтерфейс
- Зручний та зрозумілий опис проблем
- Аналізатор вказує де саме в коді проблема , а саме відокремлює

У Deepcode: snyk ми побачили такі переваги:

- Зручний інтерфейс
- Сортування по мовам програмування
- Зручний та зрозумілий опис проблем

Після аналізу коду деякими аналізаторами, можна рекомендувати Deepsource як найкращий статичний аналізатор коду серед протестованих.

Embold показав теж непоганий результат , але по функціоналу він віддає перевагу Deepsource.

Deepcode:snyk, має непоганий функціонал але практика показала, що дана програма погано сканує, коли інші аналізатори просканували та виявили 6-7 недоліків коду , Deepcode:snyk – один.

ВИСНОВКИ

За результатами роботи можна зробити наступні висновки.

1. На основі аналізу функціоналу та характеристики статичних аналізаторів коду в забезпеченні ІБ запропоновано такі критерії оцінки :

- Підтримка мов програмування
- Функціонал
- Підтримка компіляторів
- Доступність

2. З’ясовано, що такі статичні аналізатори , як: DeepSource, SonarQube та Cast можна рекомендувати як найбільш застосовні.

3. За результатами тестування деяких статичних аналізаторів із використанням модельного коду, були виявлені переваги статичних аналізаторів для програмних розробників. В цілому можна рекомендувати Deepsource як найкращий статичний аналізатор коду серед протестованих.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. TQSO - An overview on the Static Code Analysis approach in Software Development, 2009, 16 с.

2. Static code analysis tools: a systematic literature review, Darko Stefanović, Danilo Nikolić, Dušanka Dakić, Ivana Spasojević, Sonja Ristić January 2020, 11 с.

3 .Static Program Analysis Anders Møller and Michael I. Schwartzbach February 10, 2022 165 с.

4. J. West and B. Chess, Secure programming with static code analysis, Pearson Education, 2007

5. Software Development Glossary. Static Analysis [Електронний ресурс] Режим доступу: https://deepsource.io/glossary/static-analysis/ (дата звернення: 09.12. 2021)

6. Software Development Glossary. Abstract Syntax Tree [Електронний ресурс] Режим доступу: https://deepsource.io/glossary/ast/ (дата звернення: 09.12. 2021)

7. Dynamic Analysis vs. Static Analysis [Електронний ресурс] Режим доступу: https://www.cism.ucl.ac.be/Services/Formations/ICS/ics_2013.0.028/inspector_xe/documentation/en/help/GUID-E901AB30-1590-4706-94B1-9CD4736D8D2D.htm (дата звернення: 09.12. 2021)

8. Software Testing Help. TOP 40 Static Code Analysis Tools (Best Source Code Analysis Tools) [Електронний ресурс] Режим доступу: https://www.softwaretestinghelp.com/tools/top-40-static-code-analysis-tools/ (дата звернення: 09.12. 2021)

9. Перелік інструментів для статичного аналізу коду - List of tools for static code analysis [Електронний ресурс] Режим доступу:

https://uk.wikijaa.ru/wiki/list_of_tools_for_static_code_analysi (дата звернення: 09.12. 2021)

10. SonarQube [Електронний ресурс] Режим доступу: https://quarta-soft.ru/sonarqube/ (дата звернення: 09.12. 2021)

11. CodeScene [Електронний ресурс] Режим доступу: https://analysis-tools.dev/tool/codescene (дата звернення: 09.12. 2021)

12 .Embold [Електронний ресурс] Режим доступу https://embold.io/ (дата звернення: 09.12. 2021)

13. CodeSonar [Електронний ресурс] Режим доступу: https://www.grammatech.com/codesonar-cc (дата звернення: 09.12. 2021)

14. CAST [Електронний ресурс] Режим доступу: https://www.castsoftware.com/products/engineering-dashboard (дата звернення: 09.12. 2021)

15. PMD [Електронний ресурс] Режим доступу: https://pmd.github.io/ (дата звернення: 09.12. 2021)

16. SourceMeter [Електронний ресурс] Режим доступу: https://sourcemeter.com/ (дата звернення: 09.12. 2021)

17. Polyspace [Електронний ресурс] Режим доступу: https://www.mathworks.com/products/polyspace.html (дата звернення: 10.12. 2021)

18. PVS-Studio [Електронний ресурс] Режим доступу: https://pvs-studio.com/en/pvs-studio/ (дата звернення: 07.12. 2021)

19. DeepSource [Електронний ресурс] Режим доступу https://deepsource.io/features/ (дата звернення: 09.12. 2021)

20. Deepcode: snyk [Електронний ресурс] Режим доступу: https://www.deepcode.ai/ (дата звернення: 04.05. 2022)

21. Detect Python and Java code security vulnerabilities with Amazon CodeGuru Reviewer [Електронний ресурс] Режим доступу: https://aws.amazon.com/ru/blogs/devops/detect-python-and-java-code-security-vulnerabilities-with-codeguru-reviewer/ (дата звернення: 01.05. 2022