МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н. Каразіна

Факультет: **ННІ Каразінський банківський інститут**
Кафедра: **Інформаційних технологій та математичного моделювання**
Спеціальність: **125 Кібербезпека**
Освітня програма: **Кібербезпека у фінансових технологіях**

Група: АБ-41б **денна форма навчання**

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

на тему:

«СИСТЕМА АВТОМАТИЗОВАНОГО ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ КОДУ НА ОСНОВІ ШІ»
ЗА НАКАЗОМ № 4601-5/335 ВІД 07 ЛЮТОГО 2025 РОКУ

здобувача вищої освіти **Федоренка Сергія Ігоровича**

Робота допущена до захисту в ЕК
протокол кафедри ІТММ №13 від 31.05.2025р.

Завідувач кафедри ІТММ
к.п.н., доцент
_____ Н.І. Стяглик

Науковий керівник
к. т. н.
_____ О. В. Соболєв

м. Харків 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В. Н. Каразіна

Факультет навчально-науковий інститут "Каразінський банківський інститут"
Кафедра інформаційних технологій та математичного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки та інформаційні технології в бізнесі

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ **Н. І. Стяглик**
Підпис ініціали, прізвище

"08" лютого 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЄКТ)

Федоренка Сергія Ігоровича
(прізвище, ім'я, по батькові студента)

1. Тема роботи система автоматизованого виявлення вразливостей коду на основі ШІ

керівник роботи к. т. н. О. В. Соболєв
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "08" лютого 2025 року № 4601-5/335

2. Строк подання студентом роботи 15 травня 2025 року

3. Перелік питань, які потрібно розробити:
У розділі 1: Дослідити актуальність кібербезпеки в сучасному бізнесі

У розділі 2: Розглянути наявні стандарти та класифікації кіберзагроз. Проаналізувати конкурентів та вивчити історію розвитку кібербезпеки.

У розділі 3: Вибрати технологої для розробки та розробити рішення система автоматизованого виявлення вразливостей коду на основі ШІ

4. План роботи

№ з/п	Назви етапів роботи
1	Вибір здобувачем теми кваліфікаційної бакалаврської роботи
2	Затвердження плану і завдання кваліфікаційної бакалаврської роботи
3	Здача кваліфікаційної бакалаврської роботи керівнику
4	Підпис кваліфікаційної бакалаврської роботи керівника
5	Підпис кваліфікаційної бакалаврської роботи у нормоконтролера
6	Допуск завідувачем кафедри до захисту кваліфікаційної бакалаврської роботи
7	Захист кваліфікаційної бакалаврської роботи

5. Дата видачі завдання 08 лютого 2025 року

Студент _____ С. І. Федоренко
підпис ініціали, прізвище

Керівник роботи _____ О. В. Соболєв
підпис ініціали, прізвище

РЕФЕРАТ

НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

«СИСТЕМА АВТОМАТИЗОВАНОГО ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ КОДУ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ»

Федоренка Сергія Ігоровича

Кваліфікаційна бакалаврська робота містить: 75 сторінок без урахування додатків, 2 таблиці, 29 рисунків, немає формул, список літератури з 33 найменувань.

Об’єкт дослідження — методи статичного аналізу вихідного коду.

Предмет дослідження — застосування алгоритмів машинного навчання для класифікації вразливостей у коді.

Мета роботи — розробити прототип системи для автоматизованого виявлення вразливостей коду на основі штучного інтелекту.

Завдання роботи: розглянути стандарти статичного аналізу; обґрунтувати архітектуру системи; реалізувати модулі парсингу коду, навчання моделі та оцінювання результатів; провести експериментальне тестування. Актуальність дослідження зумовлена зростанням складності програмних продуктів та необхідністю швидкого виявлення вразливостей для забезпечення кібербезпеки.

Результати дослідження — реалізовано прототип системи та проведено експериментальне тестування.

Практична новизна — поєднання статичного аналізу з ШІ для зниження хибнопозитивних спрацьовувань та прискорення аналізу.

Сфера застосування результатів — інтеграція у CI/CD конвеєри розробки ПЗ.

Ключові слова: вразливості коду, статичний аналіз, машинне навчання, кібербезпека, автоматизація.

ABSTRACT
ON THE QUALIFYING BACHELOR'S WORK

"SYSTEM OF AUTOMATED CODE VULNERABILITY DETECTION BASED ON ARTIFICIAL INTELLIGENCE"
by Fedorenko Serhii Ihorovych

Qualification bachelor's thesis contains: 75 pages excluding appendices, 2 tables, 29 figures, no formulas, a list of references of 33 titles.

Object of research - methods of static analysis of source code.

The subject of research is the use of machine learning algorithms for classifying vulnerabilities in code.

Purpose: to develop a prototype system for automated detection of code vulnerabilities based on artificial intelligence.

Tasks: to consider static analysis standards; to justify the system architecture; to implement modules for code parsing, model training, and results evaluation; to conduct experimental testing.

The relevance of the study is due to the increasing complexity of software products and the need to quickly identify vulnerabilities to ensure cybersecurity.

Results of the study - a prototype of the system was implemented and experimental testing was conducted.

Practical novelty - combining static analysis with AI to reduce false positives and speed up analysis.

Scope of application - integration into CI/CD software development pipelines.

Keywords: code vulnerabilities, static analysis, machine learning, cybersecurity, automation.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК, СИМВОЛІВ І ТЕРМІНІВ

ШІ – штучний інтелект

AI – Artificial Intelligence

API – Application Programming Interface

CWE – Common Weakness Enumeration

HTTP – HyperText Transfer Protocol

IBM – International Business Machines

IoT – Internet of Things

ML – Machine Learning

OWASP – Open Web Application Security Project

SQL – Structured Query Language

(…) – розширення абревіатури або уточнення терміна

« … » – цитування або назви розділів, блоків документу

– – діапазон значень або зв’язок між елементами

• – пункт списку

→ – послідовність кроків або функціональне відображення

Атака на Інтернет речей (IoT) – використання вразливостей пристроїв «розумного дому», сенсорів тощо

Вразливість – властивість ПЗ або середовища, що може бути використана зловмисником

Експлойт нульового дня – код або засіб, що реалізує атаку на раніше невідому (незафіксовану) вразливість

Класифікація вразливостей – систематизація типів та підтипів вразливостей за певними ознаками

Патерн – усталена структура коду або поведінки, яку може використовувати алгоритм виявлення вразливостей

Сканування коду – автоматизований процес пошуку потенційних вразливостей у вихідному коді

Соціальна інженерія – набір методів психологічного впливу для отримання доступу чи інформації

Фішинг – спосіб соціальної інженерії, спрямований на викрадення даних користувача

Хакер – особа, що володіє технічними навичками і використовує їх з різною метою

ВСТУП

Кінець історії не відбувся! Сучасний світ перебуває на грані дуже серйозних геополітичних змін, і ці зміни настають шляхом військових, економічних та, найголовніше, цифрових війн. Хакери підпорядковані політичними та терористичними угрупуваннями проводять успішні атаки. Наслідками атак є витік чутливих даних, або вплив на політичний устрій держави. Причини, через які атаки стають успішними, завжди різні. Одні з головних проблем це недбалість програмістів при написанні коду, або, що найгірше, спеціально залишені бекдори, які складно виявити при не автоматизованому аналізі коду. Додатково до складності перевірки додається кількість джерельного коду сучасних програмних систем, які можуть сягати до сотень незалежних блоків коду. Це робить ручну перевірку коду дуже складною задачею, а іноді взагалі неможливою! Але вихід є! Сучасні системи машинного навчання та штучного інтелекту є рішенням в сучасній системі автоматизації перевірки та аналізу коду. На момент написання документу, вже досліджувалась тема сканування коду за допомогою штучного інтелекту. В статті «Інструменти штучного інтелекту для автоматизації тестування на проникнення» були виділені відповідні переваги та обмеження використання штучного інтелекту (ШІ) в аналізі коду.

Переваги використання ШІ при аналізі коду:

1) швидкість сканування, яка є набагато швидшою за ручний перегляд коду;
2) повна відсутність людського фактору;
3) легка масштабованість системи, що дозволяє без значних затрат аналізувати велику кодову базу;

4) прогнозування майбутніх вразливостей на основі сформованих патернів коду.

Обмеження використання ШІ при аналізі коду:

1) Складність навчання та оновлення штучного інтелекту відповідно вимогам;
2) Помилкові спрацювання;
3) Високі вимоги до потужності обладнання та до безпеки самої моделі ШІ;

Мета роботи: Розробити систему автоматичного сканування кодової бази на наявність поточних вразливостей та прогнозування майбутніх вразливостей на основі існуючої кодової бази. Забезпечити формування звітності із вказанням критичності, типу, та характеру вразливості та запропонувати способів їх викорінювання.

Завдання:

1) огляд та класифікація існуючих методів аналізу коду;
2) Визначення вимог до системи тестування коду на вразливості;
3) Розробка концепції та архітектури системи тестування;
4) Вибір та впровадження методів штучного інтелекту;
5) Перевірка системи на існуючій кодовій базі;

Об’єктом дослідження є автоматизація процесу виявлення вразливостей у коді використовуючи методи штучного інтелекту Предметом дослідження стане програмне забезпечення яке реалізуватиме автоматичне сканування та виявлення вразливостей в коді засобами штучного інтелекту

РОЗДІЛ 1
АКТУАЛЬНІСТЬ КІБЕРБЕЗПЕКИ В СУЧАСНОМУ БІЗНЕСІ

1.1. Кібербезпека не потрібна?

За останні 30 років підхід до розробки програмного забезпечення кардинально змінився. Технології просунились від асемблера, де програміст повинен був контролювати кожен біт пам’яті в системі, до мов високого рівня як C# де програміст може зосередитись на реалізації бізнес логіки без прив’язки до процесора, архітектури та інших унікальностей системи, де вся логіка керування системою вбудована «під капотом» мови програмування та керується автоматично. Роль програміста зменшилась але не зменшилась загроза ззовні. Досі зберігається активна проблема «щита та меча» а саме проблема хакерів та захисту даних.

Програмісти сьогоднішнього часу переважно використовують готові модулі та сервіси, які вже надають послуги базового захисту від кіберзагроз. Через це може укорінюватися ідея відмови від достатньої уваги до сектора кібербезпеки продукту. На перший погляд ці питання звучать актуально і раціонально «Навіщо мені перейматись стосовно шифрування даних, що передаються якщо https гарантує безпечне з’єднання?» «Навіщо мені перейматись що хтось влізе в адресний простір застосунку якщо це контролює менеджер пам’яті?» Або актуальне посилання на особистий досвід лідерів команд: «Я думаю, що роль кібербезпеки трохи перебільшена. Про неї багато говорять, але по факту назвати якісь реальні кейси кіберзагроз мало хто може. Наведу простий приклад. Коли ми прийшли в Офіс президента, it-команда показала дашборди з тисячею атак на день, перевантаженням сайтів і т.д. Через два тижні ми їх звільнили, і нічого не відбувалося впродовж декількох місяців, поки ми збирали нову команду», - розповів Федоров — віцепрем’єр-міністр та

міністр цифрової трансформації України [5]. Маленькі підприємства розроблюють програмне забезпечена без урахування захисту через економію бюджету, більше зосередившись на розробці та маркетингу а не на безпековій цілісності продукту. Маючи надію лише на дисциплінованість та відповідальність самих фахівців. Але хоча програміст як людина може надати певним чином захищений код з появою мовних моделей ситуація крайнє погіршилась. На поточний момент кожен бажаючий може за пару днів за допомогою ChatGPT зробити прототип продукту. Цей продукт може досягнути фінансового успіху і успішно розвиватись за допомогою генерації коду через ChatGPT. Але настане момент коли продукт буде підтверджено кібератаці, то котрої не буде готовим програмний продукт, який був нашвидку написаний за пару днів.

1.2. Глобальна кібервійна та загрози хакерських атак

Незважаючи на зміну ролі програмістів у написанні програмних продуктів їх відповідальність за безпеку програмного забезпечення не зменшується. В епоху глобальних війн кожен продукт може бути зачеплений і опинитись під впливом кібератак. В 2025 році 21 лютого популярна крипто площадка Bybit була атакована північнокорейськими хакерам. Збитки склали близько 1.5 млрд доларів [6]. У той же час повномасштабне військове вторгнення росії на територію України супроводжується підривом суверенітету зсередини країни. Збільшилась загальна кількість кібератак на державні сайти та реєстри. Спровоковані витоки особистих даних громадян. І це все відбувається в час коли йде заяво про «переоцінену кібербезпеку». На фоні глобальних кіберзагроз, не зважаючи на недбале ставлення до кібезахисту, попит на фахівців стрімко зростає. За останні 8 років ринок кібербезпеки виріс в 8 разів!

1.3. Ціна питання

За даними ресурсу Work.ua середня заробітна плата фахівця з кібербезпеки становить 25 тисяч гривень, що може вважатися високими затратами для малого та середнього бізнесу [7]. Гостро стає питання зменшення вартості базових послуг кібербезпеки та автоматизація процесу для зменшення використання людського ресурсу.

1.4. Брак кадрів

Ще однією проблемою в організації кіберзахисту є брак фахівців. На поточний момент відомо про 55 державних університетів які пропонують освітні програми кібербезпеки та щонайменше 16 приватних IT-шкіл які надають спеціалізовані програми професійного розвитку. [4]

Рис. 1.1 Порівняння кількості державних вузів та приватних шкіл

Але при наявності щонайменше 71 навчального закладу суттєвим питанням залишається якість підготовки фахівців в державних університетах. Однією з глобальних проблем є відсутність практичного досвіду в ході навчання. Багато університетів не надають практичних навичок з кібербезпеки студентам, що призводить до браку знань та навичок які можуть знадобитись в ході професійної діяльності студента. В ході чого маємо маленький висновок що кількість закладів не відповідає якості. Дуже частим явищем є що після закінчення навчання студенти мігрують в загальний IT сектор, такі як back-end та front-end, де не потребуються спеціалізовані професійні навички. Частково прогалини заповнюють комерційні компанії за допомогою приватних курсів де враховано вимоги сучасних стандартів та вимоги ринку.

РОЗДІЛ 2
СУЧАСНИЙ СТАН ГАЛУЗІ КІБЕРБЕЗПЕКИ

2.1. Історія розвитку кібербезпеки

Кібербезпека — це область, яка стала невід'ємною частиною нашого сучасного цифрового життя. З кожним роком загрози в онлайн-світі стають дедалі складнішими і витонченішими, тому захист інформації є критично важливим завданням не тільки для організацій, а й для приватних осіб [21].

В роках першого розвитку програмованих систем та перших прототипах мережевого з’єднання між комп’ютерами та мейнфреймами фахівці почали стикатись з першими проблемами захисту комп’ютера від зовнішнього впливу. Одним з перших зафіксованих комп’ютерних вірусів став Creeper який почав заражувати комп’ютери в 1970х роках. Вірус був створений Бобом Томасом в рамках експериментального дослідження комп’ютерних мереж.

Рис. 2.1. Боб Томас - розробник Creeper

Вірус не ніс руйнівного характеру, а був демонстраційним. Єдине що він робив це виводив на головний екран напис «I’m the creeper, catch me if you can!» («Я привид, спіймай мене, якщо зможеш!») але вже на той момент він став великим викликом для тогочасних фахівців [22].

Рис. 2.2. Демонстрація напису, що виводив Creeper

Вірус поширювався в мережі яка ставала прототипом першого інтернету – військова мережа ARPANET. Вірус був розрахований на зараження під’єднаних до мережі мейнфреймів та комп’ютерів які працювали на операційні системі TENEX. Не дивлячись не сугубо експериментальний характер вірусу, була поставлена задача на розробку програмного забезпечення для запобігання зараження. Так з’явився перший антивірус Reaper. Антивірус був розроблений Реєм Томлінсоном. Єдина функція яка була запрограмованою це виявлення екземпляру програми Creeper та видалення його

З часів першого комп’ютерного вірусу зацікавленість фахівців в кібербезпеці лише зростала. Зростала і кількість хакерів та не лише кількість їх атак та ще й складність і хитрість. Найпопулярнішою хакерською атакою

часів 1980-1990 років є атака на ARPANET з використанням вірусу Morris Worm. Morris Worm є першим вірусом типу «хробак», який за час активації зміг поразити до 6000 інтернет вузлів США. В музеї науки в Бостоні досі зберігається вихідний код хробака. Це була наймасовіша кібератака за весь попередній час існування комп'ютерної мережі. За час існування хробак наніс збитки в 96,5 мільйонів доларів. Також набрали популярності вірусне програмне забезпечення націлене на комп'ютери на системі Microsoft DOS. Одними з найвідоміших цілей шкідливого ПО було шифрування даних та вимагання грошового викупу за послугу розшифрування.

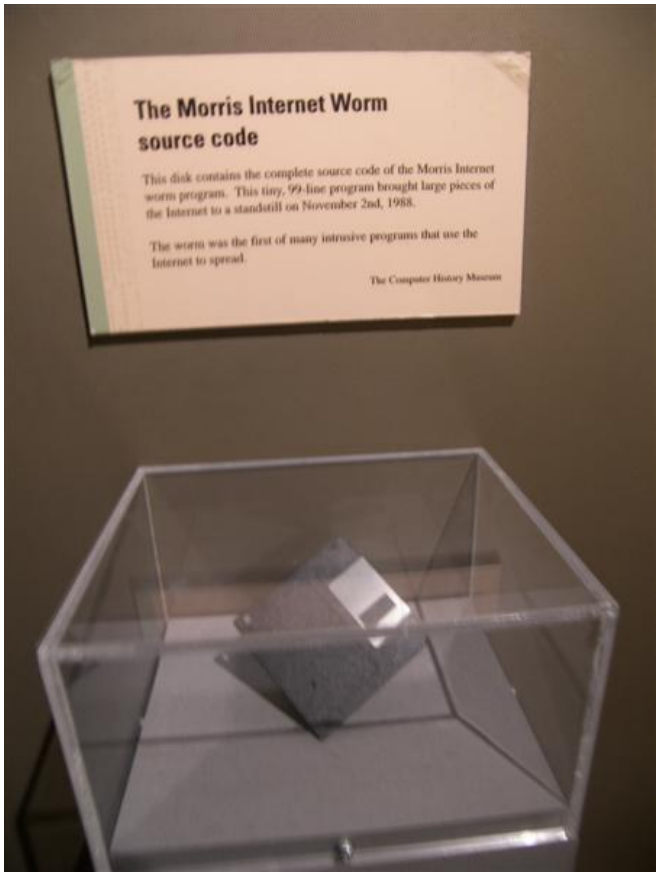


Рис. 2.3. Вихідний код Morris Worm в музеї США

На початку 2000х років роль комп'ютерів та глобальної мережі набула великої значимості. Цивільна інфраструктура, така як авіакомпанії, перейшли

на цифрове управління бізнес-процесами. Державні установи, інститути, великий та малий бізнеси почали вводити в свою роботу комп'ютери і програмне забезпечення. Роль програмного коду та вартість помилки зросла не просто в рази, а на порядки. Тим часом у 2000х роках відбулися різні великі кібератаки на популярні мережі [23]:

- Yahoo! Зламано та викрадено понад 500 аккаунтів користувачів
- Amazon. Масштабна DDOS атака виводить з ладу сервери.
- eBay. Тривала DDOS атака.
- Dyn. DDOS атака що вивела з ладу DNS сервери і перекрила доступ користувачів до будь яких популярних платформ
- Sony PlayStation Network. Взлом що поламав доступ гравців до мережі та заблокував можливість мережевої гри на декілька днів

В ці роки були розроблені та застосовані шкідливі програми масового ураження, прикладами яких є віруси ILOVEYOU, Code Red та використовують шифрування та шантаж для вимагання грошей з потерпілих бізнесів та окремих користувачів.

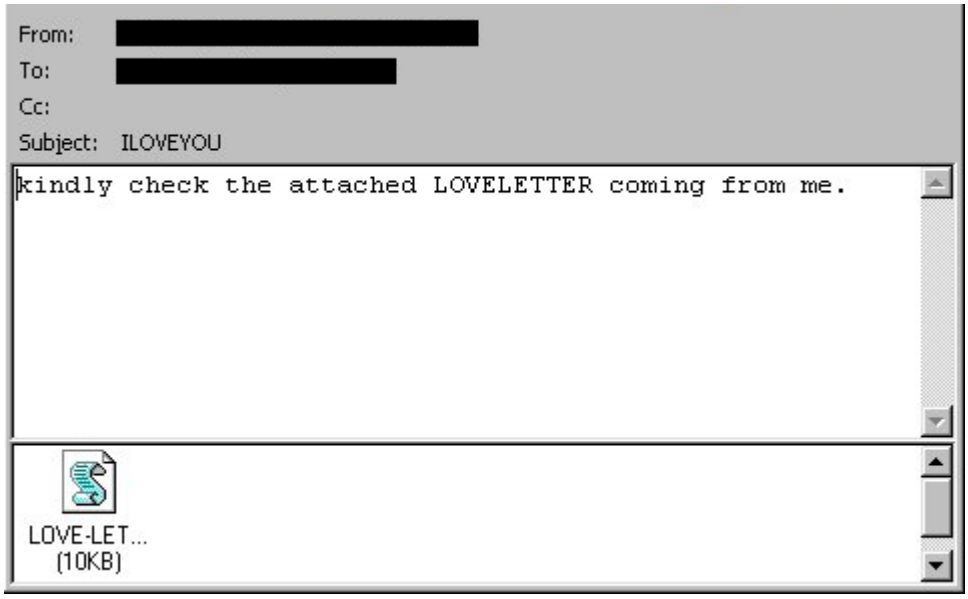


Рис. 2.4. Електронний лист який розповсюджував ILOVEYOU

У роки буму зростання соціальних мереж, та модернізації та широкому поширенні мобільних пристроїв почалися нові величезні складнощі з організацією захисту установ та бізнесу. Нові атаки ставали більш хитрими та менш помітними. Зазвичай використовувались такі слабкі місця як непродумані паролі які були встановлені на мобільні пристрої та їх аккаунти. Також набрала популярність така методика як спрямована атака на певну організацію, установу чи бізнес. Окремою проблемою стало навчання співробітників мінімальному об’єму знань з кібербезпеки, які були введені в корпоративну культуру та змінені принципи роботи з персоналом. Основною причиною вдалих кібератак були робітники які не відповідально віднеслись до культури кібербезпеки та ігнорували методи кібергігієни при роботі з корпоративною поштою або соціальною мережею, чим наразили власну компанію на злам захисту та, як наслідки – величезні збитки

Наші дні є найскладнішими по масштабам кіберзагроз. Атаки здійснюються на уряди, корпорації, цивільну та військову інфраструктури. Змінюються технології, методи, алгоритми та філософія атак. Також частим явищем є цілі команди кіберзлочинці або навіть окремі структури фінансовані державами-терористами.

В цей же час фахівці з кібербезпеки теж вдосконалюють програмне забезпечення, оновлюють антивірусне програмне забезпечення та вносять інновації в методи вирішення задач та протидії кіберзагрозам та надійного захисту конфіденційних даних. Проте, не зважаючи на поліпшення заходів безпеки, ризики буди атакованим злочинною організацією з роками стає все більше.

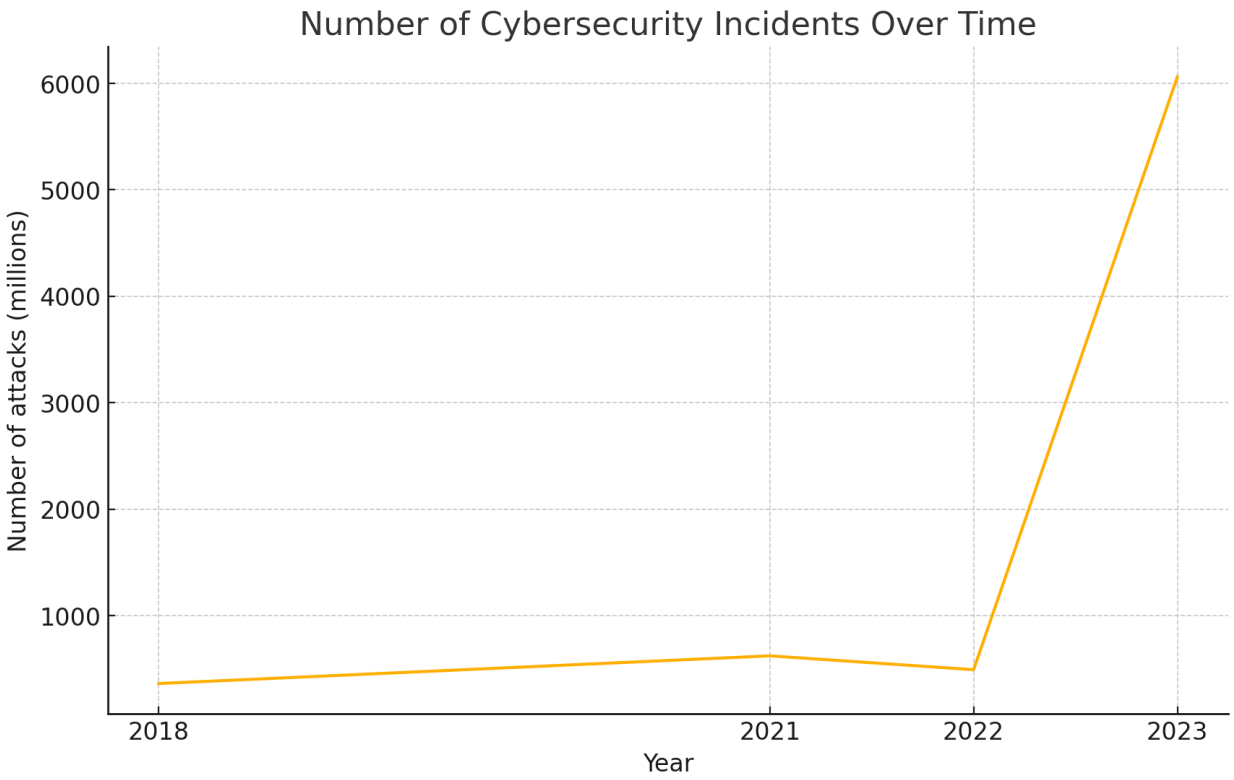


Рис. 2.5. Графік частоти кібератак з часом

2.2. Визначення та класифікація вразливостей та загроз

Протягом розвитку IT вектору і збільшення фактору кіберзагроз кожна організація намагалась визначити та класифікувати кіберзагрози і розділити їх на типи. Такий підхід надає великому бізнесу конкретні визначення проблеми та дає змогу організувати пошук методу закриття вікна можливості реалізувати кіберзагрози.

Все зазвичай починається з інциденту. Менеджер або лідер команди помічає аномалії в журналах подій системи системи або повідомлення від перших потерпілих. Наприклад це можуть бути повідомлення про несанкціоновані зміни даних аккаунта або про відсутність доступу до сайту чи іншу не характерну поведінку сервісу/застосунку. Інцидент фіксується і передається на аналіз.

До аналізу проходить процес збору всіх даних аналітики, артефактів та скарг користувачів. Збираються всі можливі докази інциденту. Копіюються системні журнали подій, формуються робочі звіти, допитується персонал та потерпіла сторона. Робляться знімки системи, списки запитів та дампи оперативної пам’яті.

Після збору даних проводиться ручний аналіз. Використовуються програми для декомпілювання для перегляду коду Assembler. Продивляються системні журнали подій, записуються підозрілі дії в цих журналах, створення файлів, запуск команд та процесів. Додатково допитується персонал. Перевіряються бази даних, нові записи та зміни існуючих. Під час аналізу окремо виділяються ознаки поведінки атаки та її симптоми.

Після статичного аналізу записів системи йде моделювання атаки в контрольованому середовищі. Використовуються тестові середовища розгортання або віртуальні машини. Фахівець намагається відтворити атаку своїми силами та фіксувати пророблені дії та виявлені слабкі зони системи.

Після успішного моделювання йде запис перших таксономій. Проводиться порівняння загрози із вже існуючими видами загроз та вірусів. Записуються спільні ознаки та відмінності. У разі, якщо фахівець зіткнеться із невідомим видом загрози то вводиться новий вид загрози де записуються ознаки притаманні атаці або вірусу.

Після успішної класифікації загрози проводиться обговорення в спільноті фахівців. Обговорюються методи атаки, на що схоже та методи запобігання. Кожен фахівець висловлює свою експертну думку основану на знаннях та досвіді. Група фахівців приходить до спільних висновків.

В лабораторія вивчаючих віруси та кібербезпеку проводиться повторний аналіз вірусу або зовнішньої загрозі. Створюються перші сигнатури та спроби протидіяти загрозі. Потім оновлені алгоритми та правила реалізується в коді антивірусу.

Відома компанія IBM виділяє таки типи кіберзагроз:

1) Шкідливе програмне забезпечення: скорочення від «зловмисне програмне забезпечення» - це програмний код, який написаний навмисно, щоб завдати шкоди комп'ютерній системі або її користувачам. Майже кожна сучасна кібератака використовує певний тип шкідливого програмного забезпечення. Зловмисники використовують шкідливе програмне забезпечення для отримання несанкціонованого доступу та виведення заражених систем з ладу, знищення даних, викрадення конфіденційної інформації та навіть знищення файлів, важливих для роботи операційної системи. До найпоширеніших типів шкідливих програм належать Програми-вимагачі блокують дані або пристрій жертви і погрожують зберегти їх заблокованими або оприлюднити, якщо жертва не заплатить викуп зловмиснику. Згідно зі звітом IBM Security X-Force Threat Intelligence Index 2024, у 2022 році атаки з використанням програм-вимагачів становили 17% від усіх кібератак. Троянський кінь - це шкідливий код, який обманом змушує людей завантажити його, виглядаючи як корисна програма або ховаючись у легальному програмному забезпеченні. Прикладами є троянські програми віддаленого доступу (RAT), які створюють секретний чорний хід на пристрої жертви, або троянські програми-«крапельниці», які встановлюють додаткове шкідливе програмне забезпечення після того, як закріплюються в цільовій системі або мережі. Шпигунські програми - це дуже секретні шкідливі програми, які збирають конфіденційну інформацію, наприклад, імена користувачів, паролі, номери кредитних карток та інші персональні дані, і передають її зловмиснику без відома жертви. Хробаки - це самовідтворювані програми, які автоматично поширюються в програмах і пристроях без участі людини.

2) Соціальна інженерія та фішинг. Соціальна інженерія, яку часто називають «людським хакінгом», маніпулює цілями, щоб змусити їх

вчинити дії, які розкривають конфіденційну інформацію, загрожують їхньому фінансовому добробуту або організації, або іншим чином ставлять під загрозу особисту чи організаційну безпеку. Фішинг - найвідоміша і найпоширеніша форма соціальної інженерії. Фішинг використовує шахрайські електронні листи, вкладення, текстові повідомлення або телефонні дзвінки, щоб обманом змусити людей поділитися особистими даними або обліковими даними для входу в систему, завантажити шкідливе програмне забезпечення, відправити гроші кіберзлочинцям або здійснити інші дії, які можуть зробити їх вразливими до кіберзлочинів. До найпоширеніших видів фішингу належать:

3) Списовий фішинг: вузькоспрямовані фішингові атаки, які маніпулюють конкретною особою, часто використовуючи дані з публічних профілів жертви в соціальних мережах, щоб зробити шахрайство більш переконливим.

а) Китовий фішинг: фішинг зі списом, націлений на керівників компаній або заможних людей.

б) Компрометація ділової електронної пошти (BEC): шахрайство, в якому кіберзлочинці видають себе за керівників, постачальників або довірених ділових партнерів, щоб обманом змусити жертву переказати гроші або поділитися конфіденційними даними.

в) Ще одним поширеним видом соціальної інженерії є підміна доменних імен (також відома як підміна DNS), коли кіберзлочинці використовують підроблений веб-сайт або доменне ім'я, яке видає себе за справжнє - наприклад, «applesupport.com» замість support.apple.com, - щоб обманом змусити людей ввести конфіденційну інформацію. У фішингових листах часто

використовують підроблені доменні імена відправників, щоб зробити лист більш достовірним і легітимним.

4) Атака “Людина посередині”. Під час MITM-атаки кіберзлочинець підслуховує мережеве з'єднання, щоб перехопити і передати повідомлення між двома сторонами та викрасти дані. Незахищені мережі Wi-Fi часто є щасливим місцем полювання для хакерів, які прагнуть здійснити MITM-атаки.

5) Атака на відмову в обслуговуванні. Атака на відмову в обслуговуванні (DoS) - це кібератака, яка перевантажує веб-сайт, додаток або систему великими обсягами шахрайського трафіку, що робить їх надто повільними для використання або повністю недоступними для законних користувачів. Розподілена атака на відмову в обслуговуванні, або DDoS-атака, схожа на DoS-атаку, за винятком того, що вона використовує мережу підключених до Інтернету, заражених шкідливим програмним забезпеченням пристроїв або ботів, відомих як ботнети, для виведення з ладу або аварійного завершення роботи цільової системи.

6) Експлойти нульового дня. Експлойт нульового дня - це тип кібератаки, який використовує вразливість нульового дня - невідому або ще не усунену чи невиправлену вразливість у комп'ютерному програмному забезпеченні, апаратному чи програмно-апаратному забезпеченні. «Нульовий день» означає, що у виробника програмного забезпечення або пристрою немає часу на виправлення вразливостей, оскільки зловмисники вже можуть використовувати їх для отримання доступу до вразливих систем. Однією з найвідоміших вразливостей нульового дня є Log4Shell - вада в широко використовуваній бібліотеці журналювання Apache Log4j. На момент виявлення в листопаді 2021 року вразливість Log4Shell існувала у 10 відсотках світових цифрових активів, включаючи багато веб-додатків, хмарних сервісів та фізичних кінцевих точок, таких як сервери.

7) Атака на пароль. Як випливає з назви, ці атаки полягають у тому, що кіберзлочинці намагаються вгадати або викрасти пароль або облікові дані для входу в обліковий запис користувача. Багато атак на паролі використовують соціальну інженерію, щоб обдурити жертв і змусити їх мимоволі поділитися цими конфіденційними даними. Однак хакери також можуть використовувати атаки грубої сили для викрадення паролів, багаторазово перебираючи різні комбінації паролів, поки одна з них не буде успішною.

8) Атака на Інтернет речей. Під час атаки на Інтернет речей (IoT) кіберзлочинці використовують вразливості в пристроях IoT, таких як розумні домашні пристрої та промислові системи управління, щоб заволодіти пристроєм, викрасти дані або використати його як частину ботнету для інших зловмисних цілей.

9) Ін'єкційні атаки. Під час цих атак хакери впроваджують шкідливий код у програму або завантажують шкідливе програмне забезпечення для виконання віддалених команд, що дозволяє їм читати або модифікувати базу даних чи змінювати дані на веб-сайті. Існує кілька типів ін'єкційних атак. Два з найпоширеніших включають

а) SQL-ін'єкції: коли хакери використовують синтаксис SQL, щоб підмінити ідентифікаційні дані, викрити, підробити, знищити або зробити недоступними існуючі дані або стати адміністратором сервера бази даних.

б) Міжсайтовий скриптинг (XSS): ці типи атак схожі на SQL-ін'єкції, за винятком того, що замість вилучення даних з бази даних вони зазвичай заражають користувачів, які відвідують веб-сайт.

Також IBM окрім самого визначення кіберзагроз вивчає джерела. В основному це йдуть кіберзлочинці які прослідують цілі фінансової вигоди або політичних поглядів. Існує дуже багато кіберзлочинців ціль яких є витягування

логіну/паролю адміністративних юзерів, особистих даних та незаконного доступу до індивідуальних або корпоративних фінансових рахунків:

1) Кіберзлочинці: Ці особи або групи вчиняють кіберзлочини, здебільшого заради фінансової вигоди. Серед поширених злочинів, які скоюють кіберзлочинці, - атаки з вимогами викупу та фішингові шахрайства, які обманом змушують людей здійснювати грошові перекази або розголошувати інформацію про кредитні картки, облікові дані для входу в систему, інтелектуальну власність або іншу приватну чи конфіденційну інформацію.

2) Хакери. Хакер - це людина, яка володіє технічними навичками для компрометації комп'ютерної мережі або системи. Майте на увазі, що не всі хакери є загрозою або кіберзлочинцями. Наприклад, деякі хакери - так звані етичні хакери - по суті видають себе за кіберзлочинців, щоб допомогти організаціям і державним установам перевірити свої комп'ютерні системи на вразливість до кібератак.

3) Національно-державні актори. Національні держави та уряди часто фінансують суб'єктів загроз з метою викрадення конфіденційних даних, збору конфіденційної інформації або підриву критично важливої інфраструктури іншого уряду. Ці зловмисні дії часто включають шпигунство або кібервійну і, як правило, добре фінансуються, що робить загрози складними і важкими для виявлення.

4) Внутрішні загрози. На відміну від більшості інших кіберзлочинів, інсайдерські загрози не завжди є результатом дій зловмисників. Багато інсайдерів завдають шкоди своїм компаніям через людські помилки, наприклад, ненавмисне встановлення шкідливого програмного забезпечення або втрату пристрою, виданого компанією, який кіберзлочинець знаходить і використовує для доступу до мережі. Тим не менш, зловмисні інсайдери існують. Наприклад, незадоволений працівник може зловживати привілеями доступу заради грошової вигоди

(наприклад, оплати від кіберзлочинця або національної держави), або просто зі злості чи помсти.

IBM провела детальний аналіз та класифікувала багато видів кіберзагроз та їх джерел. На меті поточного проекту є зменшення впливу від джерел загроз які класифіковані як «Внутрішні загрози» та зменшення вірогідності витоку таких загроз як «Ін'єкційні атаки», «Атака на Інтернет речей» та «Експлойти нульового дня». Всі ці види загроз пов’язані з прогалинами в програмному коді які не вдалося відслідкувати та виналагодити в ранніх та пізніх етапах розробки. Для більш детального визначення та виявлення системою кібервразливостей слід їх декомпозитувати на більш дрібні класифікації та підтипи. Одним із популярних рішень для класифікації кібервразливостей було створена класифікація CWE (Common Weakness Enumeration) яка класифікує та оновлює види кибервразливостей та їх популярність. В таблиці приведені найпопулярніші 25 вразливостей на момент 2024 року [3].

Таблиця 2.1

Перелік вразливостей за класифікацією CWE

Код	Назва	Опис
1	2	3
CWE-79	Неправильна нейтралізація вхідних даних під час генерації веб-сторінок («міжсайтовий скриптинг»)	Продукт не нейтралізує або неправильно нейтралізує вхідні дані, контрольовані користувачем, до того, як вони будуть розміщені у вихідних даних, які використовуються у вигляді веб-сторінки, що надається іншим користувачам.
CWE-787	Записи за межами обмежень	Пристрій записує дані в кінець або перед початком призначеного буфера.

Продовження таблиці 2.1

1	2	3
CWE-89	Неправильна нейтралізація спеціальних елементів, що використовуються в SQL-команді (“SQL-ін'єкція”)	Продукт повністю або частково будує команду SQL, використовуючи вхідні дані, на які впливає зовнішній вплив з попереднього компонента, але він не нейтралізує або неправильно нейтралізує спеціальні елементи, які можуть змінити передбачувану команду SQL, коли вона надсилається до наступного компонента. Без достатнього видалення або взяття в лапки синтаксису SQL у вхідних даних, контрольованих користувачем, згенерований SQL-запит може призвести до того, що ці дані будуть інтерпретовані як SQL, а не як звичайні користувацькі дані.
CWE-352	Підробка міжсайтових запитів (CSRF)	Веб-додаток не перевіряє або не може в достатній мірі перевірити, чи правильно сформований, дійсний, послідовний запит був навмисно наданий користувачем, який подав запит.

Продовження таблиці 2.1

1	2	3
CWE-22	Неправильне обмеження імені шляху до обмеженого каталогу («обхід шляху»)	Продукт використовує зовнішні вхідні дані для створення імені шляху, призначеного для ідентифікації файлу або каталогу, розташованого в обмеженому батьківському каталозі, але продукт не нейтралізує належним чином спеціальні елементи в імені шляху, які можуть призвести до того, що ім'я шляху розпізнає розташування за межами каталогу.
CWE-125	Зчитування за межами	Пристрій зчитує дані після кінця або перед початком призначеного буфера.

Продовження таблиці 2.1

1	2	3
CWE-78	Неправильна нейтралізація спеціальних елементів, що використовуються в команді ОС («ін'єкція команди ОС»)	Продукт конструює всю або частину команди операційної системи, використовуючи вхідні дані, що надходять ззовні від попереднього компонента, але він не нейтралізує або неправильно нейтралізує спеціальні елементи, які можуть модифікувати заплановану команду операційної системи, коли вона надсилається до наступного компонента.
CWE-416	Використання після звільнення	Продукт повторно використовує або посилається на пам'ять після її звільнення. У якийсь момент після цього пам'ять може бути виділена знову і збережена в іншому покажчику, тоді як початковий покажчик посилатиметься на місце розташування десь у новому розподілі. Будь-які операції з використанням початкового вказівника більше не є дійсними, оскільки пам'ять "належить" коду, який працює з новим вказівником.

Продовження таблиці 2.1

1	2	3
CWE-862	Відсутня авторизація	Продукт не виконує перевірку авторизації, коли актор намагається отримати доступ до ресурсу або виконати дію.
CWE-434	Необмежене завантаження файлів небезпечного типу	Продукт дозволяє завантажувати або передавати небезпечні типи файлів, які автоматично обробляються в його середовищі.
CWE-94	Неналежний контроль генерації коду («Ін'єкція коду»)	Продукт конструює весь або частину сегмента коду, використовуючи вхідні дані з попереднього компонента, але не нейтралізує або неправильно нейтралізує спеціальні елементи, які можуть змінити синтаксис або поведінку передбачуваного сегмента коду.
CWE-20	Неправильна перевірка вхідних даних	Продукт отримує вхідні дані, але не перевіряє або неправильно перевіряє, що вхідні дані мають властивості, необхідні для безпечної та правильної обробки даних.

Продовження таблиці 2.1

1	2	3
CWE-77	Неправильна нейтралізація спеціальних елементів, що використовуються в команді («Ін'єкція команди»)	Продукт будує всю команду або її частину, використовуючи вхідні дані з попереднього компонента, але не нейтралізує або неправильно нейтралізує спеціальні елементи, які можуть змінити заплановану команду, коли вона надсилається на наступний компонент.
CWE-287	Неправильна автентифікація	Коли актор стверджує, що має певну особу, продукт не доводить або недостатньо доводить, що це твердження є правильним.
CWE-269	Неправильне керування привілеями	Продукт не призначає, не змінює, не відстежує та не перевіряє привілеї для актора, створюючи ненавмисну сферу контролю для цього актора.
CWE-502	Десеріалізація ненадійних даних	Продукт десеріалізує ненадійні дані без достатньої гарантії того, що отримані дані будуть дійсними.
CWE-200	Розкриття конфіденційної інформації неавторизованій особі	Продукт розкриває конфіденційну інформацію особі, яка не має явного дозволу на доступ до цієї інформації.

Продовження таблиці 2.1

1	2	3
CWE-918	Підробка запитів на стороні сервера (SSRF)	Веб-сервер отримує URL-адресу або подібний запит від попереднього компонента і отримує вміст цієї URL-адреси, але це не гарантує, що запит надсилається до очікуваного місця призначення.
CWE-119	Неправильне обмеження операцій у межах буфера пам'яті	Виріб виконує операції з буфером пам'яті, але читає з нього або записує в комірку пам'яті за межами призначеної для цього межі буфера. Це може призвести до операцій читання або запису в непередбачувані ділянки пам'яті, які можуть бути пов'язані з іншими змінними, структурами даних або внутрішніми даними програми.
CWE-476	NULL Pointer Dereference	Продукт скасовує посилання на вказівник, який, як він очікує, є дійсним, але є NULL.
CWE-798	Використання жорстко закодованих облікових даних	Продукт містить жорстко закодовані облікові дані, такі як пароль або криптографічний ключ.

Продовження таблиці 2.1

1	2	3
CWE-190	Цілочисельне переповнення або обхід	Продукт виконує обчислення, які можуть призвести до цілочисельного переповнення або обходу, коли логіка передбачає, що результуюче значення завжди буде більшим за початкове значення. Це відбувається, коли ціле значення збільшується до значення, яке є надто великим для зберігання у відповідному представленні. Коли це відбувається, значення може стати дуже малим або від'ємним числом.
CWE-400	Неконтрольоване споживання ресурсів	Продукт не контролює належним чином розподіл та обслуговування обмеженого ресурсу, що дозволяє суб'єкту впливати на кількість спожитих ресурсів, що в кінцевому підсумку призводить до вичерпання наявних ресурсів.

Продовження таблиці 2.1

1	2	3
CWE-306	Відсутня автентифікація для критично важливої функції	Продукт не виконує автентифікацію для функцій, які вимагають підтвердженої ідентичності користувача або споживають значну кількість ресурсів.
CWE-863	Неправильна авторизація	Продукт виконує перевірку авторизації, коли актор намагається отримати доступ до ресурсу або виконати дію, але робить це неправильно.

Джерело: [3]

2.3. Сучасні методи виявлення вразливостей

Перед початком роботи треба розглянути вже наявні технології сканування продукту на вразливості та бекдори. Буде розглянуте багато методів від статичного аналізу коду до fuzz-тестування. [1]

Першою групою інструментів є інструменти для статичного аналізу коду. Їх перевагами є швидке виявлення вразливостей ще на ранніх етапах розробки. Йде аналіз кодової бази проекту на наявність таких класичних помилок як потенційні SQL-ін’єкції, XSS, buffer overflow. Зазвичай ці інструменти розроблені з інтеграцією в середовища розробки або в процеси автоматичного тестування та компіляції для вчасного і зручного попередження про вразливості. В крайніх випадках це супроводжується примусовим припиненням компіляції та публікації при виявленні критичних помилок.

Компанія SonarSource розробила рішення SonarCube для статичного аналізу коду. Продукт розрахований на роботу з мовами C#, Java, Python з можливостями розширеннями на інші мови через відповідні плагіни. SonarCube надає послуги аналізу всієї кодової бази та надання звіту о стані коду та о наявних проблемах. У продукту є відкритий код та інтеграція з різними середовищами розробки [24].

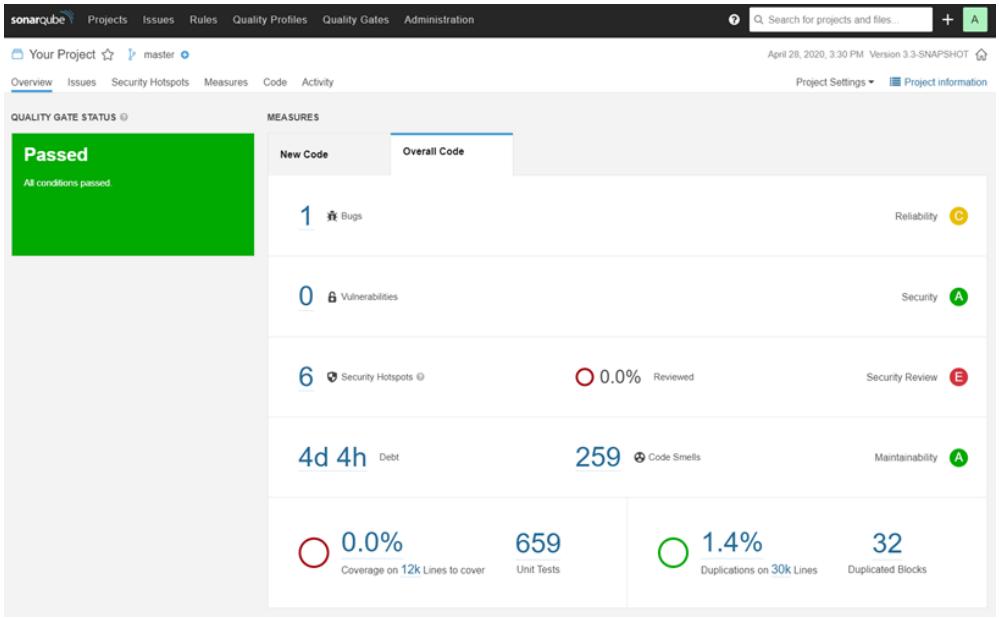


Рис. 2.6. Інтерфейс SonarCube

Іншим відомим рішенням є програмне забезпечення від компанії Checkmarx – CxSAST. Компанія заявляє що це унікальне рішення для аналізу вихідного коду, яке надає інструменти для виявлення, відстеження та виправлення технічних і логічних недоліків у вихідному коді, таких як вразливості безпеки, проблеми з дотриманням нормативних вимог і бізнес-логіки. Програма надає результати аналізу у вигляді графів та відповідних таблиць для зручного сприйняття та аналізу ситуації. Програма має можливості інтеграції в такі середовища розробки Eclipse, Visual Studio, Visual Studio Code Extension, та IntelliJ [25].

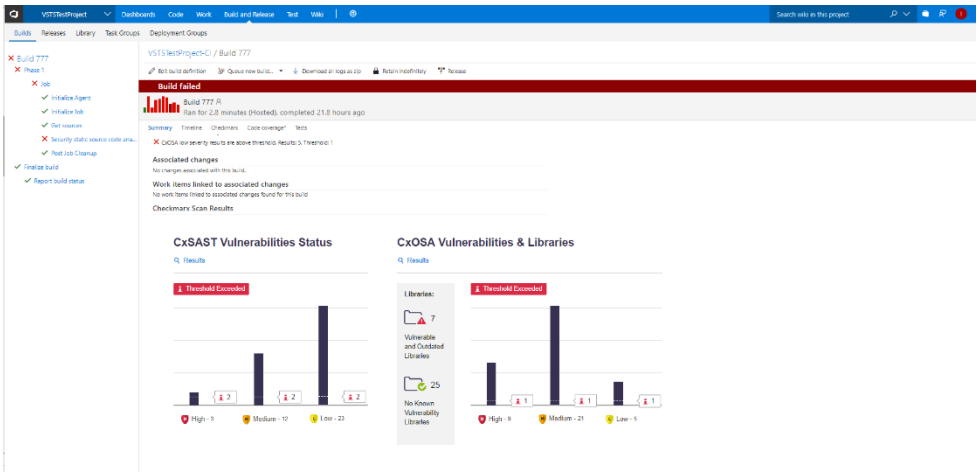


Рис. 2.7. Інтерфейс CxSAST

Організація OpenText може похизуватись своїм засобом для статичного аналізу коду Fortify Static Code Analyzer. Компанія обіцяє прискорення розробки, виявлення та усунення вразливості на ранніх стадіях розробки, отримуючи точні результати на основі бенчмарку OWASP 1.2b, підвищуючи швидкість та якість.

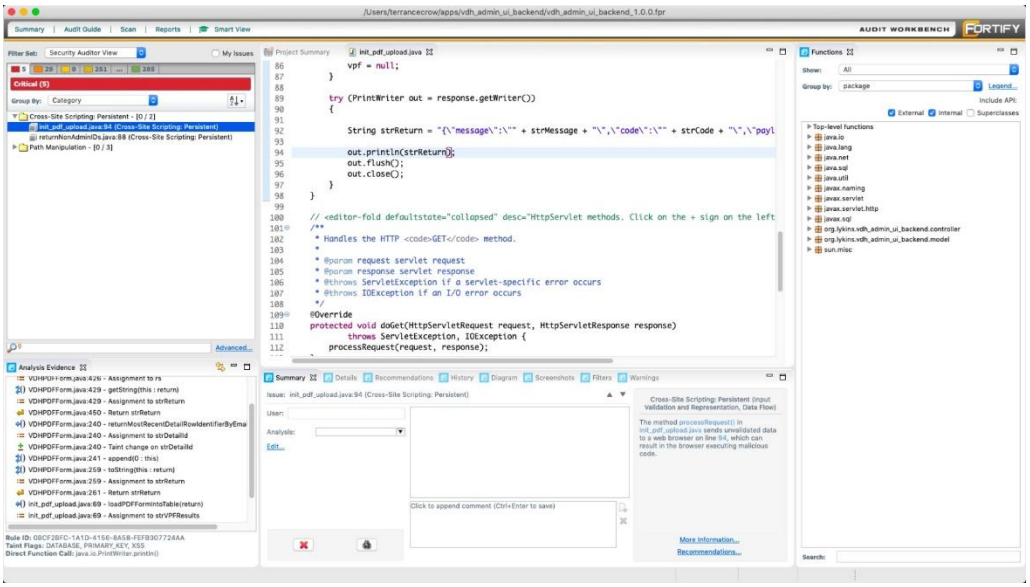


Рис. 2.8. Інтерфейс Fortify Static Code Analyzer

OWASP Benchmark - це повнофункціональний веб-додаток з відкритим вихідним кодом, який містить тисячі придатних для використання тестових кейсів, кожен з яких прив'язаний до певних CWE, які можна проаналізувати за допомогою Fortify Static Code Analyzer [26].

Всі програми, що перелічені вище, були розроблені з однією метою – виявлення вразливостей у кодовій базі продукту. Вони роблять розробку безпечною та швидкою, звертаючи увагу на вразливості. Проте у цих інструментів є значні обмеження. В більшості випадків алгоритми та методи розпізнавання плагінів можуть створювати дуже незручний шум, а саме велику кількість хибних спрацювань. Через нестабільні спрацювання фахівець буде схильний до ігнорувань деяких сигналів що призведе до більш халатного ставлення до безпеки в коді. Також алгоритми аналізують код без runtime-контексту, що зменшує коло виявлення потенційних вразливостей які можуть виникати саме під час роботи програми.

Наступною групою софту для аналізу є інструменти динамічного аналізу безпеки. Цей софт працює з вже запущеною програмою та аналізує її стан через такі канали як HTTP, API, або UI. В основі лежить перевірка запущеного екземпляра програми. Ідеально демонструє перевірку SQL-ін’єкції на практиці та виявлення XSS, CSRF, неправильного налаштування серверу, небезпечні HTTP-заголовки. Здібний перевірити конфігураційну інфраструктуру таку як WAF, CORS, TLS.

Яскравим представником є OWASP ZAP - надзвичайно потужний інструмент для наскрізного тестування. Його часто використовують люди, які хочуть поглиблено вивчити веб-додаток. ZAP пропонує багато функцій, включаючи активне і пасивне сканування та можливості тестування API. По суті, це проксі, який діє як «маніпулятор посередині». Він дозволяє бачити всі запити, які ви робите до веб-додатку, і всі відповіді, які ви отримуєте від нього, що дає змогу виявляти вразливості та потенційні вектори атаки в режимі

реального часу. Це багатовимірний інструмент, який часто використовується тестувальниками проникнення, мисливцями за багами та розробниками для сканування веб-додатків на наявність ризиків безпеки під час процесу тестування додатків [28].

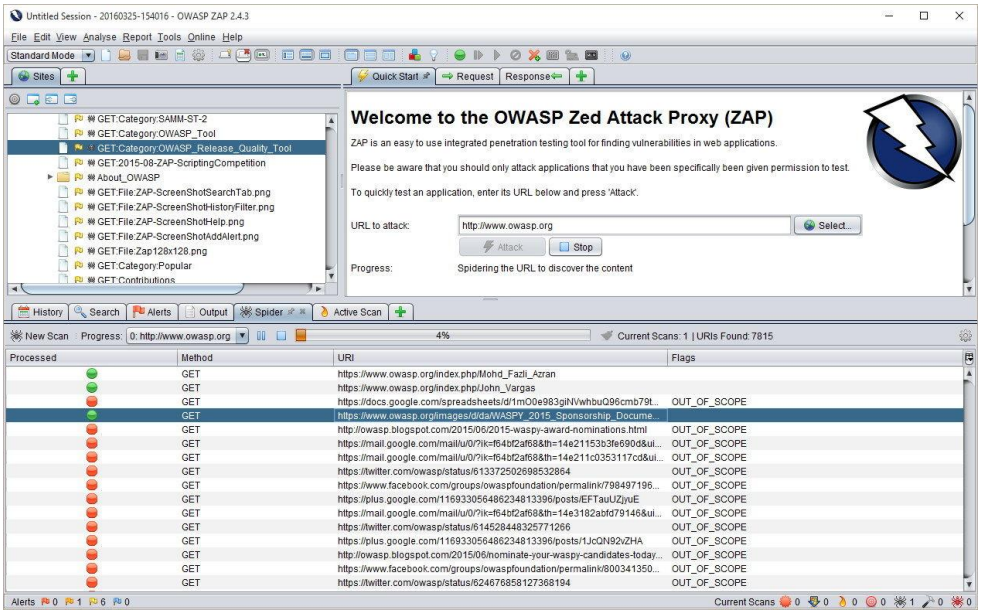


Рис. 2.9. Інтерфейс OWASP ZAP

Наступним інструментом який ми розглянемо є Burp Suite Professional - це інструментарій для тестування на проникнення, розроблений для тестувальників веб-безпеки. Він поєднує в собі автоматизовані повторювані тестові завдання, розроблені експертами ручні та напівавтоматичні інструменти тестування безпеки для швидкого та ефективного тестування на вразливості та методи злому. Burp Scanner чудово допомагає користувачеві знаходити широкий спектр вразливостей у веб-додатках. Основні характеристики: функції ручного тестування на проникнення, розширені або кастомні автоматизовані атаки, автоматизоване сканування на вразливості, інструменти для підвищення продуктивності та доступ до бібліотеки розширень, завантаження сканера [29].

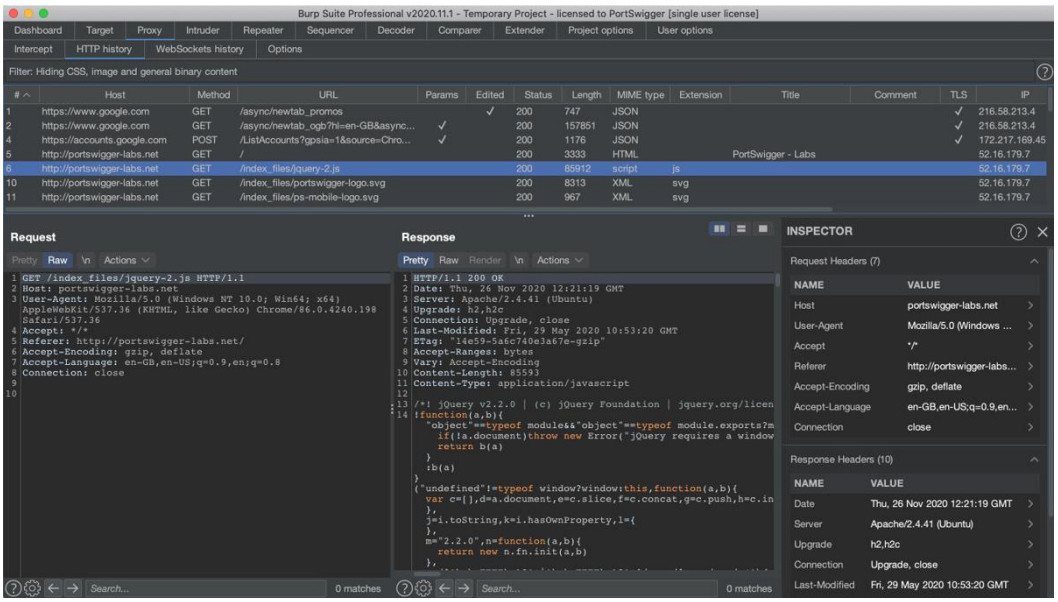


Рис. 2.10. Інтерфейс Burp Suite Professional

Не дивлячись на те що інструменти працюють з продуктом через пряме втручання в реальному режимі часу,, ці методики мають значні обмеження у вигляді наявності стабільного тестового середовища для коректного спрацювання тестів та неможливість виявлення помилок які пов’язані лише з внутрішнім станом системи та джерельним кодом, які не виявляються методами користувацького інтерфейсу або програмного інтерфейсу API.

Взявши до уваги відмінності, сильні та слабкі сторони кожного х методів тестування була розроблена третя група інструментів для тестування – Інтерактивний аналіз. Сильною стороною такого підходу є поєднання статичного аналізу коду та динамічного аналізу продукту. Тобто одночасно сканується як код так і API/UI. Такий підхід дає менше хибних спрацювань за рахунок фактичного тестування «на льоту». Також цей метод відомий глибоким контекстом даних, тобто більш детальною інформацією які данні прийшли, звідки та як були оброблені програмним кодом та де був результат.

Компанія Contrast Security може продемонструвати своє рішення Contrast Assess - це інструмент тестування безпеки додатків, який поєднує в собі статичне, динамічне та інтерактивне тестування безпеки додатків підходи для надання високоточної та постійної інформації про вразливості безпеки у додатках. Assess підходить для таких середовищ, як тестові, QA або серверні сервери. Він також може бути застосований до робочих станцій розробників. У поєднанні з інтеграціями Contrast, такими як Visual Studio, розробники можуть знаходити та виправляти вразливості, не виходячи з інтегрованого середовища розробки [30].

Рис. 2.11. Інтерфейс Contrast Assess

Також фахівцям відомий такий інструмент як Hdiv Security. Hdiv - провідний постачальник програмного забезпечення з відкритим вихідним кодом для самозахищених додатків, що працюють у режимі реального часу. Рішення Hdiv вбудовуються в додатки під час розробки, щоб забезпечити найсильніший доступний самозахист додатків під час виконання від 10 загроз OWASP. Hdiv є піонером у розробці програмного забезпечення для кібербезпеки із самозахистом, і сьогодні його рішеннями користуються провідні комерційні постачальники програмного забезпечення та глобальні

підприємства в банківській, державній, роздрібній, технологічній та аерокосмічній галузях. Компанія заявляє що Hdiv захищає від 90% ризиків безпеки додатків, що входять до Топ-10 OWASP [27] - широкого переліку найбільш критичних недоліків безпеки веб-додатків, таких як SQL-ін'єкції, міжсайтовий скриптинг, підробка міжсайтових запитів, фальсифікація даних та атаки грубої сили. Hdiv пропонує вищу ефективність, ніж будь-яке з доступних на даний момент рішень для боротьби з ризиками безпеки веб-додатків. Можливість забезпечити надійний захист без необхідності знати і розуміти всі поточні загрози безпеки. Можливість забезпечити надійний захист без шкоди для продуктивності додатків або якості роботи користувачів. Можливість вбудувати безпеку в додатки під час розробки, замість того, щоб потім повертатися і вносити виправлення і налаштування в додатки [31].

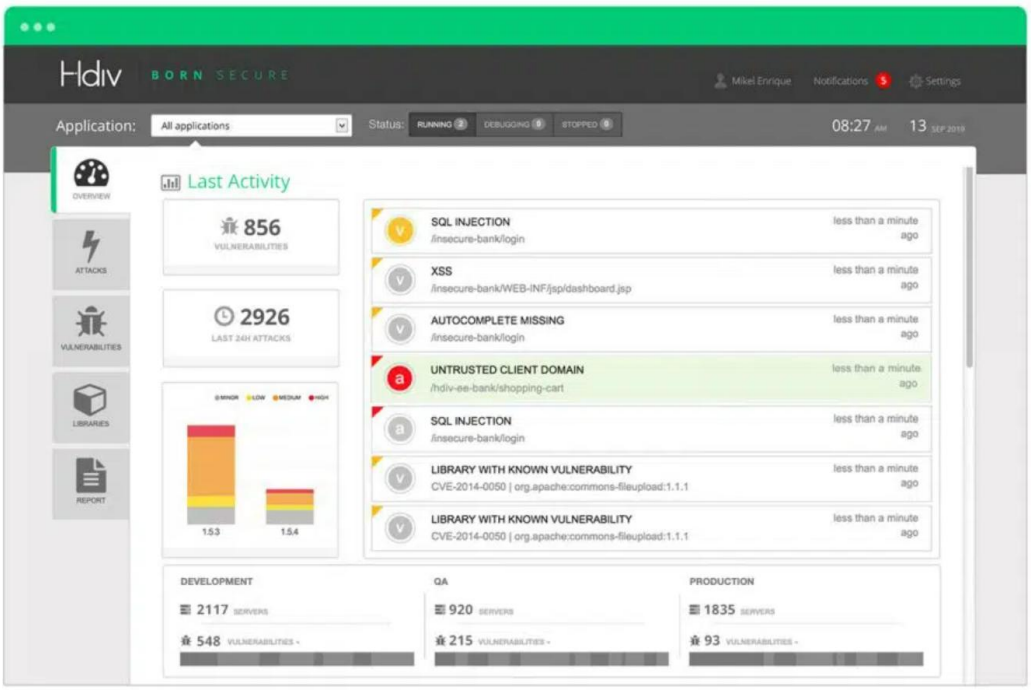


Рис. 2.12. Інтерфейс Hdiv

Додатковою методикою тестування з відомих ще є Fuzz-тестування. За принципом роботи тестування схоже на імітацію DDoS атаки. Софт генерує

тисячі та мільйони мутованих або заздалегідь налаштованих даних та передає їх в тестуємий софт методами API та аналізує данні в пам’яті, креші, та витоки пам’яті та інші види незапланованої поведінки програмного продукту. Софт може як і пророблювати мутації вже вказаних зразків даних так і генерувати данні з повного нуля в залежності від виду поставленої задачі. З плюсів такого тестування є виявлення різних помилок крайніх випадків такі як перегрузка буфера.

Яскравим прикладом софту для подібного роду тестування є American Fuzzy Lop. AFL вимагає, щоб користувач надав приклад команди, яка запускає програму, що тестується, і принаймні один невеликий приклад вхідних даних. Вхідні дані можна подавати до програми, що тестується, або через стандартний ввід, або у вигляді вхідного файлу, вказаного у командному рядку процесу. Наразі мережеві програми не підтримуються безпосередньо, хоча в деяких випадках існують реальні рішення цієї проблеми. Наприклад, у випадку аудіо плеєра, можна наказати відкрити за допомогою нього короткий звуковий файл. Потім fuzzer намагається виконати вказану команду, і якщо це вдається, він намагається зменшити вхідний файл до найменшого, який викликає таку саму поведінку. Після цієї початкової фази AFL починає власне процес фаззингу, застосовуючи різні модифікації до вхідного файлу. Коли програма, що тестується, падає або зависає, це зазвичай означає виявлення нової помилки, можливо, уразливості в системі безпеки. У цьому випадку модифікований вхідний файл зберігається для подальшої перевірки користувачем. Для того, щоб максимізувати продуктивність фаззингу, American fuzzy lop очікує, що програма, яка тестується, буде скомпільована за допомогою утиліти, яка оснащує код допоміжними функціями, що відстежують потік управління. Це дозволяє фаззеру виявити, коли поведінка об'єкта змінюється у відповідь на вхідні дані. У випадках, коли це неможливо, також підтримується тестування «чорного ящика» [32].

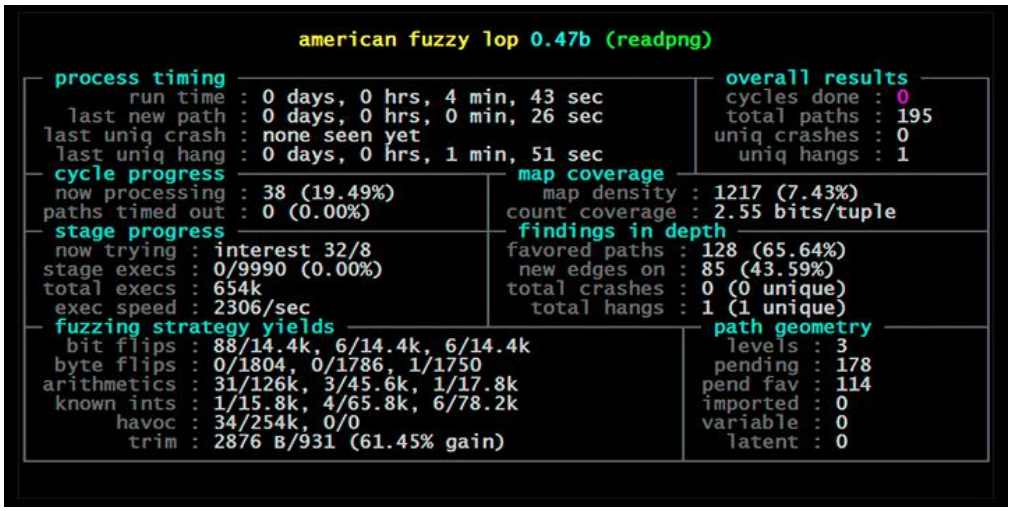


Рис. 2.13. Консольний інтерфейс American Fuzzy Lop

Додатково для тестування можна використовувати libFuzzer - це еволюційний рушій нечітких обчислень, який керується покриттям у процесі роботи. LibFuzzer пов'язаний з бібліотекою, що тестується, і подає нечіткі вхідні дані до бібліотеки через певну точку входу (так звану «цільову функцію»); потім фаззер відстежує, які ділянки коду досягнуто, і генерує мутації на корпусі вхідних даних для того, щоб максимізувати покриття коду. Інформація про покриття коду для libFuzzer надається інструментом LLVM SanitizerCoverage [33].

Не дивлячись на досить потужні методики тестування, ці методи не обділені обмеженнями та недоліками. Найперше з чим буде стикатись фахівець це довгий процес налаштування та надання тестових даних для мутаційних та згенерованих тестів. Також даний метод тестування не завжди здатен до тестування глибинних помилок та знаходження внутрішніх вразливостей.

Тож дивлячись на переваги та недоліки різних систем аналізу та тестування найкращим рішенням буде поєднання та використання всіх видів тестування. З цим можуть допомогти інструменти оркестрації, менеджменту та формування звітності. Тобто інструменти які поєднують в собі функціонал одночасного керування всіма видами тестових продуктів та збору єдиного

зручного звіту по результату роботи аналізу. Найвідомішими інструментами є інструменти автоматизованої збірки такі як GitLab CI/CD, GitHub Actions, Jenkins. Також йде інтеграція з такими інструментами як DefectDojo, ThreadFix для збору результатів та формування звітності о виявлених уразливостях.

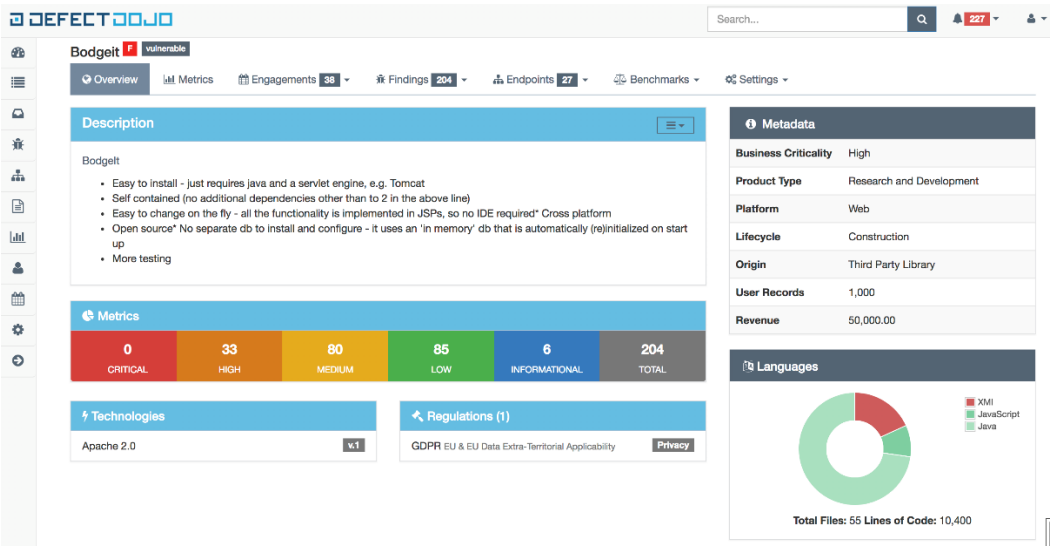


Рис. 2.14. Вид звітності в сервісі DefectDojo

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ

3.1. Проблема використання ручного труда

Кожен розробник в ході розробки програмного забезпечення створює унікальний і неповторний код, який потім використовується іншими розробниками в унікальних і неповторних проектах. В ході написанні коду навіть у дисциплінованих та навчених розробників сильно впливає людський фактор що є дуже потужним джерелом кібервразливостей в коді. Залучення фахівців для аналізу кодової бази через ручний перегляд програмного коду може частково полегшити ситуацію, але з збільшенням кодової бази зростають два дуже важливих фактору: час перевірки та похибки під час перевірки. Чим більше код тим більше його треба аналізувати і тим більше шанс на те що фахівець упустить з виду важливі дрібниці та деталі.

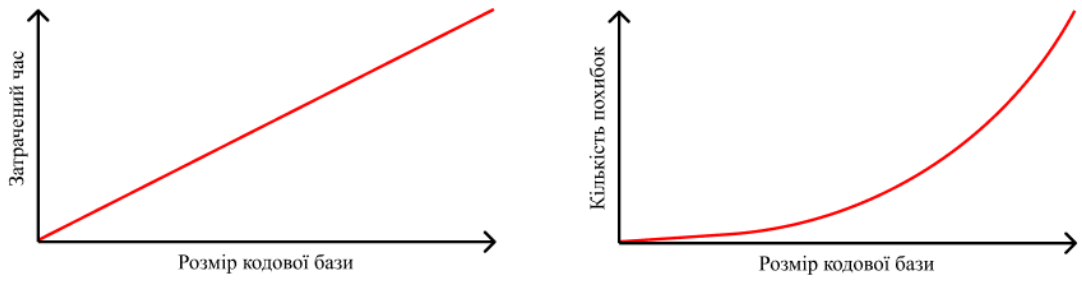


Рис. 3.1. Умовні графіки затрат часу та кількості похибок із збільшенням кодової бази при використанні річної перевірки коду

Кількість затраченого часу буде рости в арифметичній прогресії з збільшенням кількості коду в проекті а шанс похибки буде збільшуватись геометричній прогресії що дуже шкодить великим проектам і сильно знижує ефективність ручного підходу до перевірки проекту на вразливості.

Тому найкращим виходом з ситуації буде автоматизація підходу, програмне забезпечення яке буде аналізувати код знаходити в ньому прогалини, класифікувати їх, формувати звіт та надавати інформацію оператору або розробнику. Автоматизований процес зменшить фінансове навантаження на бюджет середніх та маленьких підприємств та зробить практику захищеного програмного забезпечення більш поширеною через маленьку вартість реалізації.

3.2. Новий принцип розробки: SDD

Найкращим рішенням впровадження політики розробки захищеного програмного забезпечення є інтеграція перевірки на вразливості в середину основного циклу розробки. На поточний момент кожен розробник створює програмний код за принципом «Зроби щоб працювало», тобто просто пишуть реалізацію певного функціонала. Професійні фахівці в свою чергу схильні використовувати методологію Test Driven Development, де розробник спочатку пише автоматизований приймальний тест, який в обов’язковому порядку не повинен проходитись відразу, потім пише той об’єм коду який потрібен для проходження приймального тести і в фіналі завершує цикл рефакторінгом. В випадку коли потребується не тільки робочий код але і захищений, треба доповнювати ітерацію перевіркою на вразливості, для цього буде сформований новий цикл написання коду, і назва йому Secure Driven Development (Керована безпекою розробка). В новому принципі розробки додаються два дуже важливих кроки: «Просканує» та «Захисти від вразливостей».

Рис. 3.2. Цикл розробки при використанні методу SDD

Починає цикл розробки найважливіший етап «Зроби щоб працювало». Цей етап є базовим і найважливішим для бізнеса, бо найперше що треба бізнесу від розробника це реалізація поставлених задач та ідей в тій мірі в якій це можливо. На першому етапі розробник пише код не звертаючи увагу на майбутні проблеми та лазівки для взлому програми, надаючи максимальну концентрацію реалізації ідей замовника. У разі комбінації з Керованою Тестами Розробкою перший етап розділятиметься на три під етапи «Червоний», «Зелений», «Рефакторінг». Наступний етапом йде етап під назвою «Проскануй». На другому етапі вже буде задіяна система автоматизованого виявлення вразливостей в коді. Розробник повинен отримати від системи перелік проблем, та їх класифікацію. На основі отриманих системою даних розробник проводить аналіз написаного коду та виправленню проблем. Тим самим розробник переходить до останнього етапу під назвою «Захисти вразливості». На третьому етапі розробник на основі даних, що він отримав від системи аналізу вводить зміни в кодову базу для виправлення вразливостей в коді програмного забезпечення. У висновку з відповідальним ставленням до кожного етапу бізнес очікує отримання функціонального, дешевого, швидкого, та найголовніше безпечного програмного продукту без залучення додаткових значних витрат на найми відповідних фахівців з кібербезпеки. Подібна система дозволяє безперервно оновлювати кодову базу додаючи новий функціонал та не перейматись за безпеку застосунку.

Розробник матиме «магічну кнопку» яка відображатиме стан проекту та сигналізуватиме якщо щось піде не так, тим самим розробнику відкривається вікно можливостей до швидких та легких змін у системи без зайвих турбувань та страхів щось поламати в проекті або залишити якусь небезпечну діру в захисті програмного забезпечення.

3.3. Проблема розпізнавання вразливостей в коді

Розробники всіх часів, технологій і методологій завжди писали унікальний код що може стати на заваді для систематизації розпізнавання вразливостей в текстовому форматі, бо комп’ютер приймає рішення на основі чисел та вбудованих алгоритмів а не на основі контексту кодової бази та неявних проблемах в командах які описані людськими буквами та словами. Для більш детального аналізу треба розглянути декілька кейсів проявлення однієї проблеми в різних реалізаціях коду. Розглянемо проблеми пов’язані з CWE-89 а саме «SQL-ін'єкція».

```php
<?php
$id = $_GET['id'];
$sql = "SELECT name, price FROM products WHERE id = '$id'";
$result = mysqli_query($conn, $sql);
?>
```

Рис. 3.3. Приклад коду з SQL-ін’єкцією на мові PHP

В приведеному вище коді реалізована типова проблема з неконтрольованим доступом до зміни SQL запиту, а саме через відсутності

фільтрації змінної $id. В якості значення в цю змінну може прийти від запиту не просто унікальний ідентифікатор а ще й небажане «продовження» команди в якому можуть бути запити на видалення, запис або зміну даних. Прикладом буде що замість звичайного id=5 хакер може передати спеціальний текст який сформує команду id=5' OR 1=1; -- і тоді запит стане у вигляді SELECT name, price FROM products WHERE id = '5' OR 1=1; -- що може просто запустити певний алгоритм для всіх елементів бази даних. Для того щоб виявити вразливість буде достатньо алгоритмічного виявлення ключових слів та запрограмованих структур. Наприклад можна знаходити кодом вставки SQL синтаксису методом розпізнавання ключових слів такі як SELECT або FROM. Також в самій вставці шукати методи конкатенації і за таких умов написати алгоритм який буде перевіряти один випадок порушення безпеки для певної мови програмування здається реальним.

Роздивляючись наступний випадок відразу бачимо нову проблему. Інша мова програмування додає додаткові перепони у вигляді найочевиднішого – нового синтаксису і нових текстових реалізацій логіки. Відразу потребується новий алгоритм для виявлення нового синтаксису конкатенації.

```
2
3  string keyword = Request.QueryString["q"];
4  string sql = "SELECT * FROM documents WHERE title LIKE '%" + keyword + "%'";
5
6  using (var cmd = new SqlCommand(sql, conn))
7  {
8      var reader = cmd.ExecuteReader();
9      // …
10 }
11
```

Рис. 3.4. Приклад коду з SQL-ін’єкцією на мові C#

Окрім нового синтаксису в коді ще присутня нова проблема - Blind (boolean-based) SQL-ін’єкція. Хакер може підставити команду q=' OR SUBSTRING((SELECT password FROM users WHERE user_id=1),1,1)='a що дає зловмиснику можливість отримати доступ у випадку якщо встановлений пароль в базі якому починається на символ «a».

Тобто окрім виявлення виду проблеми (в поточних випадках SQL-ін'єкція) ще треба вручну програмувати окремі алгоритми для виявлення підвидів кожної вразливості.

Наступним випадком йде код на мові програмування Python в якому продемонстровано вже іншу проблему SQL Ін’єкції а саме Blind (time-based) SQL-ін’єкція.

```
1  import pymysql
2  conn = pymysql.connect(host='localhost', user='root', password='', db='app')
3  cursor = conn.cursor()
4
5  user = input("User: ")
6  pwd  = input("Pass: ")
7  query = f"SELECT * FROM users WHERE username = '{user}' AND password =
       '{pwd}'"
8  cursor.execute(query)
9  if cursor.fetchone():
10     print("Logged in")
11
```

Рис. 3.5. Приклад коду з SQL-ін’єкцією на мові Python

Окрім вже знайомої проблеми абсолютно нового синтаксису на який треба дописувати нові алгоритми розпізнавання ще й продемонстровано нову проблему. Хакер підстановкою в поле паролю команди pwd=’wrong’ OR IF(SUBSTR(password,1,1)='a', SLEEP(5), 0)— може затримати відповідь сервера. Таким чином злодій може через затримку відповіді сервера перебирати данні паролю буква за буквою.

З вище описаних прикладів можна винести наступні проблеми написання класичного пошуку помилок через алгоритми:

1) Алгоритми повинні бути достатньо складними щоб працювати з унікальним кодом з унікальним стилем кожного програміста.
2) Під кожен вид синтаксису кожної мови треба писати окремий алгоритм розпізнавання
3) Додатково до попереднього пункту треба ще для кожного підвиду вразливості треба писати новий алгоритм розпізнавання та класифікування

Вище перелічені проблеми не просто додаються між собою а саме множаться і тим самим чим більше ми будемо мати мов програмування з унікальним синтаксисом та чим більше ми будемо мати класифікацій вразливостей тим більше алгоритмів розпізнавання нам треба буде писати і таким чином час на реалізацію програмного забезпечення буде стрімко йти до нескінченості.

З вище описаних проблем йде висновок що потрібна система яка буде встановлювати проблему не по чітко заданим алгоритмам а розпізнавати по «плаваючим» та «нечітким» признакам і в ідеалі мати можливість працювати з кейсами які не були до цього запрограмовані або навіть виявлені людиною. Цій проблемі є вирішення. Саме з метою вирішення проблем складних до автоматизації був створений штучний інтелект. Штучний інтелект має можливості розпізнавання суті тексту та генерації нового контенту на основі вже наявної бази. Тобто це каже що ШІ зможе як і виявляти проблеми коду в різних кодових базах які написані в різних стилях різними програмістами, працювати з різними синтаксичними видами мов та найголовніше виявляти ті кейси які ще не були знайдені та запрограмовані людиною. Тобто ШІ має змогу «думати та аналізувати» матеріал який буде наданий моделі [2, 17].

3.4. Вибір моделі ШІ

Як було описано з вище: ШІ зможе вирішити проблему розпізнавання вразливостей в тексту написаному в різних стилях і різними методами. Але яку модель ШІ обрати? Для цього треба роздивитись наявні на зараз види моделей штучного інтелекту [18]:

1) «Вузький» ШІ. Це вид ШІ здатен до розпізнавання тексту, класифікації об’єктів але не здатен запам’ятовувати та навчатись за межами тренувального процесу. Створений для розв’язання конкретної задачі
    a) Генеративний ШІ підвид «Вузького» ШІ який здатен аналізувати послідовність та взаємозв’язки та використовувати їх в генерації тексту, зображення або іншого контенту
2) Реакційні машини. ШІ які реагують лише на стимули, вона не здатні запам’ятовувати набуті знання але здатні адаптуватись
3) ШІ з обмеженою пам’яттю. ШІ які здатні запам’ятовувати данні та пере використовувати їх, тобто мати щось схоже на людськи спогади

Для вирішення задач потрібна модель яка поєднує в собі властивості всіх вище описаних типів. Вміти вирішувати поставлену задачу з розпізнавання вразливостей, запам’ятовувати результати та код, перенавчатись в ході використання «на льоту» та генерувати текстовий контент у вигляді звіту. Саме з цією задачею може справити мовна модель бо вона об’єднує в собі всі вище перелічені параметри всіх видів нейронної мережі. Велика мовна модель (ВММ) - це тип моделі машинного навчання, розроблений для задач обробки природної мови, таких як генерація мови. ВММ - це мовні моделі з багатьма параметрами, які навчаються за допомогою самонавчання на великому обсязі тексту.

3.5. Вибір мовної моделі

Для якісного розпізнавання та класифікації вразливостей треба обрати розумну мовну модель але при цьому вона повинна бути відносно дешевою в використанні щоби відповідати умовам та масштабам малого та середнього бізнесів. Тобто умови прийняття моделі повинні бути наступними:

- Модель повинна розпізнати та класифікувати вразливість в наданій кодовій базі
- Модель повинна витратити якнайменше грошового ресурсу на аналіз коду та генерації звіту

Для порівняння мовних моделей різних компаній було створено тестовий код, в коді навмисно було допущено вразливість, яку треба буде класифікувати ШІ. Для цього початково треба написати якісний промпт за яким буде працювати ШІ. Простими словами, промпт – це інструкція, яка дає ШІ зрозуміти, що від нього очікують. Він може бути коротким і чітким, або ж детальним і описовим, залежно від поставленого завдання [15]. Чим чіткіше та інформативніше сформульований промпт, тим кращим буде результат роботи ШІ. Існують наступні правила написання промпту [19]:

– Визначити цілі. Чітко сформульовані запитання: розуміння того, чого саме треба досягти, допоможе налаштувати спілкування з ChatGPT. Це може бути отримання інформації, генерація ідей, написання тексту або коду.
– Використання контексту. Надання достатньої інформації — чим більше контексту у промпті, тим точнішою і релевантною буде відповідь. Наприклад, якщо цікавить написання статті, треба зазначити тему, стиль, кількість знаків, ключові слова, структуру тексту і цільову авдиторію. Можна навіть задати роль штучному інтелекту, наприклад, «напиши текст як досвідчений архітектор».

- Використання прикметників. Додавання більше описовості — штучний інтелект краще зрозуміє сформований запит, якщо надати вичерпну характеристику бажаного тексту. Наприклад, замість «створи шаблон рекламної розсилки», краще конкретизувати промпт: «створи адаптивний шаблон для рекламної розсилки про нові шопери — зі згадкою ціни та описом матеріалів».
- Правильне формулювання питання. Поставлені відкриті запитання допоможуть отримати детальніші відповіді. Наприклад, замість загального формулювання «як графдизайнеру використовувати ШІ?», краще запитати «як графдизайнеру перевести растр у вектор в Adobe Illustrator за допомогою ШІ?».
- Писати промпт структуровано. Якщо складний запит, треба спробувати розбити його на кілька частин. Наприклад: напиши вступ до статті про фалафель → виділи чотири факти з історії фалафеля → розпиши окремо кожен з цих фактів → напиши висновок про історію фалафеля.
- Тестувати різні формати взаємодії. Можна запитувати про списки, поради, пояснення або навіть просити створити діалоги. Зміна формату може дати нові, неочікувані кращі результати.
- Значна деталізація запиту. Якщо відповідь не задовольняє, треба спробувати переформулювати запит. Можливо, варто уточнити певні аспекти або запитати про деталі. Недостатньо детальний промпт: «Запропонуй ідеї для обкладинки книги в жанрі фентезі». Детальний промпт: «Запропонуй ідеї для обкладинки книги в жанрі фентезі під назвою “Серце дракона”. Книга розповідає про пригоди молодого воїна, який шукає міфічного дракона в чарівному світі. Обкладинка повинна передавати настрій магії та пригод, і містити конкретні елементи: драконів, стародавні руїни,

зелений туман. Запропонуй різні варіанти дизайну, зокрема кольорові палітри, шрифти й можливі композиції».

– Запит повинен бути ввічливим. Навіть штучний інтелект хоче, щоби до нього ставилися добре. На реддіті користувачі ділилися досвідом взаємодії з ChatGPT — і дійшли висновку, що ШІ видає кращі результати, якщо поводитися з ним люб’язно, відписувати «будь ласка» або «дякую за роботу».

На основі вище перелічених правил було сформовано наступний промпт для використання.

Тепер треба порівняти різні відповіді різних мовних моделей і оцінити їх ресурсозатратність на аналіз коду. Для цього було обрано декілька популярних мовних моделей які вже встигли зарекомендувати себе на глобальному ринку технологій. Серед них були обрані такі великі і популярні моделі як моделі ChatGPT – перша популярна мовна модель. Також на розгляд було взято тіку не дуже популярні аналоги як Cloude, який славится серед програмістів своєю роботою з програмним кодом. Було взято, нещодавно вишедшую, китайську модель Deep Seek яка є найдешевшою серед всіх, чим і завоювала увагу прихильників. А також було взято до розгляду модель від компанії Google під назвою Gemini та свіжу модель від Ілона Маска Grok. Всі ці моделі будуть порівнюватись за наступними факторами:

- Кількість використаних токенів на аналіз
- Ціна на 1 міліон токенів
- Ціна за використані токени
- Якість відповіді яка буде обумовлена виявленю загроз та якістю формату відповіді

Будь ласка, виступи в ролі досвідченого аудиту безпеки ПЗ. Нижче наведено перелік типових вразливостей за стандартом CWE (код – найкоротша назва):

- CWE-79 «XSS»
- CWE-787 «Out-of-bounds Write»
- CWE-89 «SQL Injection»
- CWE-352 «CSRF»
- CWE-22 «Path Traversal»
- CWE-125 «Out-of-bounds Read»
- CWE-78 «OS Command Injection»
- CWE-416 «Use After Free»
- CWE-862 «Missing Authorization»
- CWE-434 «Unrestricted File Upload»
- CWE-94 «Code Injection»
- CWE-20 «Improper Input Validation»
- CWE-77 «Command Injection»
- CWE-287 «Improper Authentication»
- CWE-269 «Improper Privilege Management»
- CWE-502 «Insecure Deserialization»
- CWE-200 «Information Exposure»
- CWE-918 «SSRF»
- CWE-119 «Buffer Boundary Error»
- CWE-476 «NULL Pointer Dereference»
- CWE-798 «Hardcoded Credentials»
- CWE-190 «Integer Overflow»
- CWE-400 «Uncontrolled Resource Consumption»
- CWE-306 «Missing Authentication»
- CWE-863 «Improper Authorization»

Завдання:
1. Проаналізуй довільний фрагмент коду (будь-якої мови) – я підставлю його між потрійними ```.
2. Виявитай усі вразливості з наведеної вище добірки.
3. Сформуй звіт у вигляді Markdown-таблиці з такими стовпцями:
   - **№**
   - **Рядок** (де знайдено проблему)
   - **Опис** (дуже коротко, до 5 слів, з посиланням на CWE-код)

Формат відповіді:
```markdown
| № | Рядок | Опис (CWE-XX) |
|---|-------|---------------|
| 1 |  42   | SQL Injection (CWE-89) |
| 2 |  87   | XSS (CWE-79) |
...

Рис. 3.6. Промпт для мовної моделі
```

Таблиця 3.1

Порівняння вартості запиту на різних мовних моделях

Модель	Використано токенів на обробку запита	Ціна за 1 міліон токенів	Ціна за використанні токени	Результат аналізу
Chat GPT	156	$10	$0,00156	+ Надав коректну відповідь. + Вказав на вразливості
Gemini	181	$1,25	$0,0002263	+ Вказав вразливості. - Багато надлишкового тексту
Grok	187	$3	$0,000561	- Невірно визначив вразливості
Cloude	193	$3	$0,000579	- Не зміг надати конкретну відповідь. - Багато надлишкового тексту. - Не вказав на вразливості
Deep seek	155	$0,07	$0,00001085	+ Вказав на вразливості. - Багато надлишкового тексту

Джерело: розробка автора

З таблиці видно що Deep Seek є найдешевшим за ціною на міліон токенів, при тому йому ще потрібно найменше токенів для аналізу та скануванню коду. Cloude в свою чергу показав себе з найгіршої сторони, бо не зміг не виявити вразливості не вказати чіткі данні та ще й наповнив відповідь великою

кількістю непотрібного «водного» тексту, та ще й потребував найбільшу кількість токенів на реалізацію сканування та виявлення помилок.

```
using System;
using System.Data.SqlClient;

class Program
{
    static void Main()
    {
        Console.Write("Enter username: ");
        string username = Console.ReadLine();

        string connectionString = "Data Source=localhost;Initial Catalog=MyDB;Integrated
            Security=True";
        string query = "SELECT * FROM Users WHERE Username = '" + username + "'";

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();

            SqlDataReader reader = command.ExecuteReader();
            if (reader.HasRows)
            {
                Console.WriteLine("User found!");
            }
            else
            {
                Console.WriteLine("User not found.");
            }
        }
    }
}
```

Рис. 3.7. Тестовий код для тестування мовних моделей

Chat GPT хоч і є найдорожчим з списку але єдиний хто зміг надати конкретну і правильну інформацію та ще й в чіткому форматі. З цього слідує що мовна модель Chat GPT найліпше підходить для основи системи з аналізу коду на вразливості.

3.6. Архітектура застосунку

Після глибокого аналізу та вибору нейромережі для основи системи слід подумати про побудову самого програмного забезпечення та реалізації коду. Перед тим як почати написання коду треба визначити які задачі повинен виконувати програмний продукт та які цілі він переслідує. Для визначення цих моментів було використано схему User Case-ів в якій визначено який функціонал повинен бути в програмі з точки зору користувача.

В лиці основного юзера для продукту є розробник бо концепція застосунку йде як інструмент для розробки а саме від «розробників для розробників» тому і весь функціонал розглядається з точки зору саме розробника опираючись на його досвід та профорієнтацію.

Перше з чим повинен зустрітись користувач це вхід в систему з свого аккаунта, система аккаунтів в застосунку потрібна для роботи з моделлю підписки яка буде реалізована в застосунку в майбутньому. Основний функціонал створюваної системи є продукування запитів на аналіз коду та отримання звіту з вказаними помилками. Цій функціонал системи буде використовуватись якнайчастіше для отримання звітів про стан та цілісність написаного коду. Додатком є формування рекомендацій щодо звіту який було надано. Таким чином навіть недосвідчений програміст зможе чітко і правильно виправити вразливість в написаному коді та обробляти помилки з яким він не стикався у минулому.

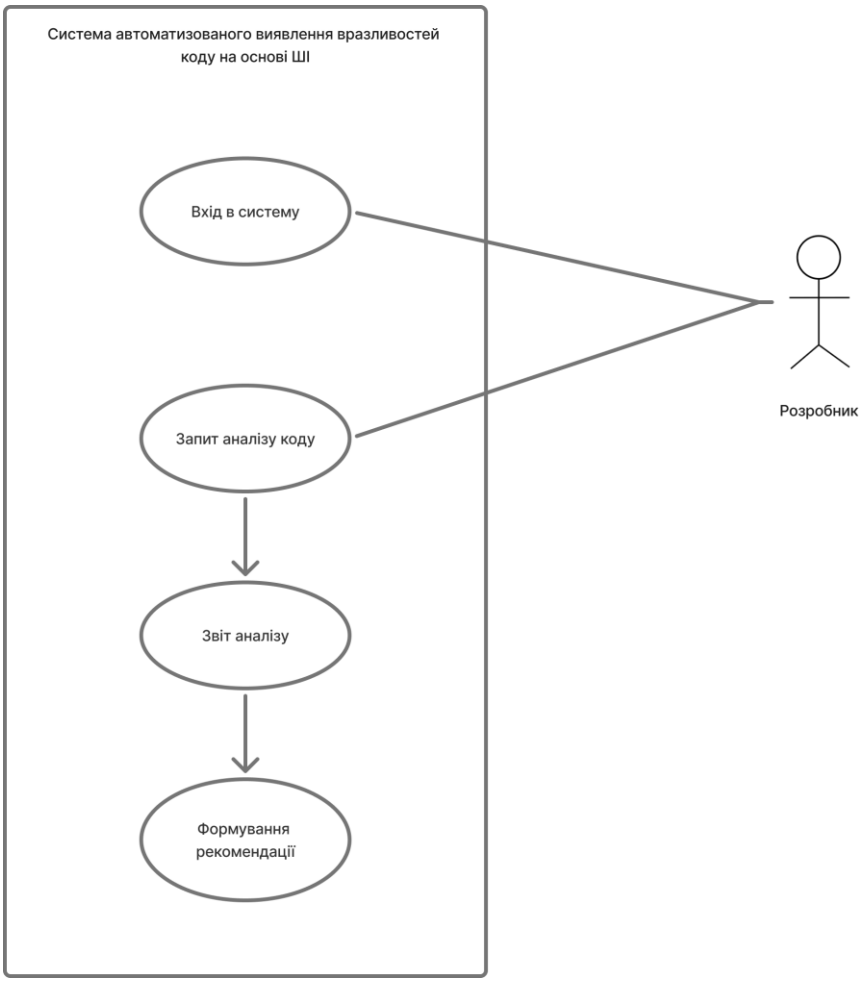


Рис. 3.8. Діаграма User-case

Вже маючи на руках готові схеми користувацьких випадків та списку базового функціоналу можна приступати до планування архітектури та побудови схеми для реалізації коду. Згідно з загальними правилами чистої архітектури програма буде розділена на два глобальні слої: деталі та бізнес логіка. До бізнес логіки буде віднесений сам механізм формування звіту, вибору файлу для аналізу та формування промпту для штучного інтелекту. Ці правила є стабільними та нечасто будуть змінюватись з часом тому їх буде проложено в основу, тобто в модуль який не залежить від інших модулей. До деталей буде віднесено користувацький інтерфейс бо інтерфейс як правило дуже часто піддається змінам та не є стабільною частиною програми, тим паче в майбутньому планується можливість переходу з інтерфейсу командної

строки до повноцінного графічного інтерфейсу або на змішаний варіант. Також до деталей буде віднесено сам модуль ШІ з причин некетрольованих змін.

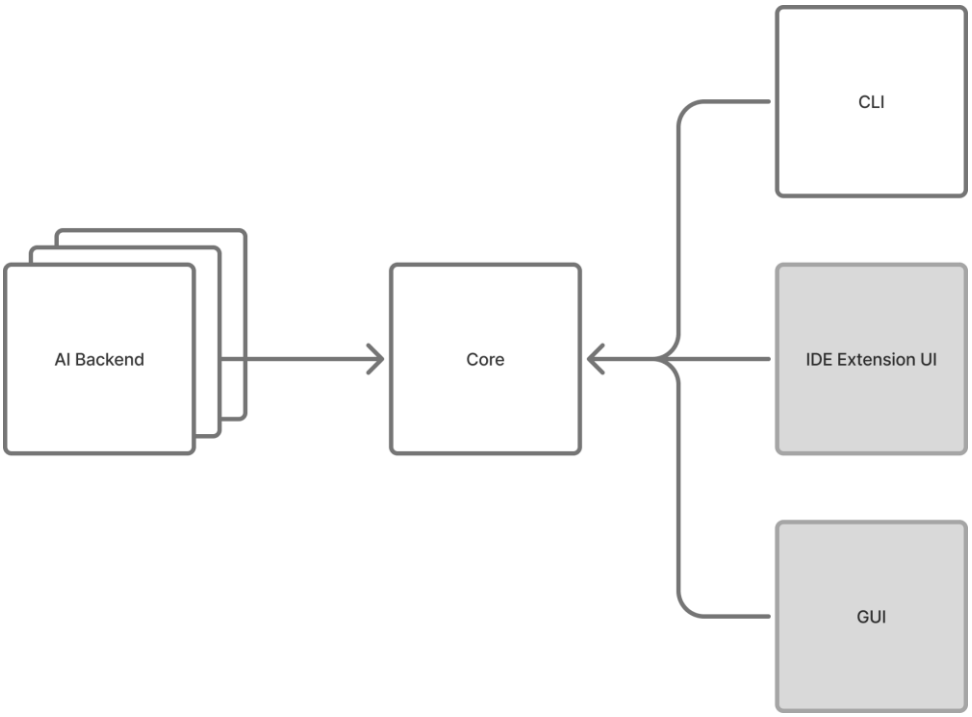


Рис. 3.9. Схематичне зображення архітектури ПЗ

Тобто сам ШІ сервіс може оновлюватись без контролю команди розробки продукту роботи. Додатково увесь час існування програми буде дуже відкрите питання зміни мовної моделі на інший підвид або на інший сервіс. І буде відкрите питання розгортання локальної моделі та її підключення. З цих причин буде ірраціонально робити бізнес логіку залежною від ШІ на рівні коду [20].

З цього всього матимемо таку схему. Модуль бізнес логіки (Core) буде незалежним і стабільним модулем. Модуль відповідальний за роботу з штучним інтелектом (AI Backend) матиме залежність від модулю бізнес логіки через розрахунки різноманітності ШІ. Також будуть окремі модулі під різні види користувацького інтерфейсу. На перших етапах буде реалізований лише інтерфейс командного рядка (CLI).

І вже на етапі з розпланованою архітектурою та встановленими користувацькими випадками вже можна приступити до вибору стеку технологій. Як основну мову програмування було вибрано мову C#. Мова C# є найпопулярнішою мовою для платформи .NET - вільного, крос-платформного, відкритого середовища розробки. Програми на C# можуть працювати на багатьох різних пристроях, від пристроїв Інтернету речей (IoT) до хмарних сервісів і скрізь, де тільки можна [8]. Можна писати програми для телефонів, настільних і портативних комп'ютерів та серверів. C# поєднує в собі: велику кількість бібліотек для роботи з різними технологіями, зручність та високий рівень програмування та крос платформу. Для взаємодії з сервісами штучного інтелекту як надійний протокол мережевого з’єднання було обрано HTTPS. HTTPS (абр. від англ. HyperText Transfer Protocol Secure; інші назви: HTTP over TLS, HTTP over SSL, і HTTP Secure) — схема URI, що синтаксично ідентична http: схемі, яка зазвичай використовується для доступу до ресурсів Інтернет. Використання https: URL вказує, що протокол HTTP має використовуватися, але з іншим портом за замовчуванням (443) і додатковим шаром шифрування/автентифікації між HTTP і TCP. Ця схема була винайдена у компанії Netscape Communications Corporation для забезпечення автентифікації та шифрування комунікацій і широко використовується в Інтернеті у програмному забезпеченні, в якому важлива безпека комунікацій, наприклад, у платіжних системах та корпоративних логінах [16]. HTTPS має забезпечувати безпеку передаваємого через мережу корпоративного коду та запобігати витоку даних через мережеве зчитування даних а також дає гарантію що під час передачі даних ніхто не зможе підмінити та підставити інші данні та результати сканування.

Як середовище для розробки було вибрано Visual Studio 2022 як зручне для використання та найбільш адаптоване до мови програмування C# рішення. На момент написання роботи було використано безкоштовну некомерційну ліцензію Visual Studio 2022 Personal, функціонал котрої є достатнім для

реалізації проекту. Інші середовища розробки, такі як від JetBrains, не були обрані через слабку адаптованість до роботи в середовищі Windows з екосистемою .Net та мовою C#, яка була обрана як найбільш підходяща реалізації проекту.

3.7. Огляд реалізованого рішення

По результатам розробки маємо реалізовану програму під назвою Varta. Для завершення інсталяції треба запустити виконавчий файл varta.exe від імені адміністратора. Цим кроком ми завершуємо початкове налаштування застосунку. В консолі буде виведено повідомлення про завершення налаштування застосунку.

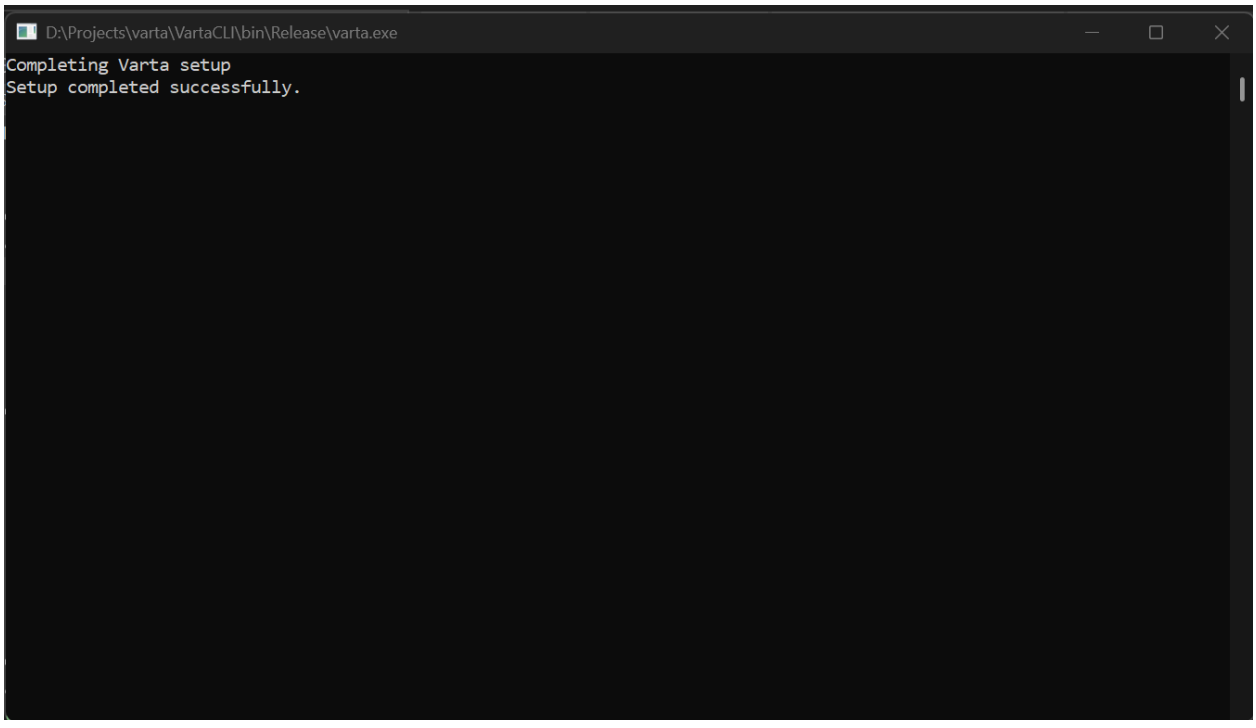


Рис. 3.10. Повідомлення про завершення конфігурації

Наступним кроком треба підготувати проект до сканування. Для цього треба запустити консоль в кореневій папці проекту і прописати команду varta init. Застосунок створить файл конфігурації і відкриє його через системний текстовий редактор. В файл конфігурації треба вписати розширення файлів, які є носіями коду та які повинні піддатися скануванню. Проект, на якому було продемонстровано застосунок, написано на мові C#, тому буде вказано розширення файлів *.cs. Файл треба зберегти, а редактор закрити.

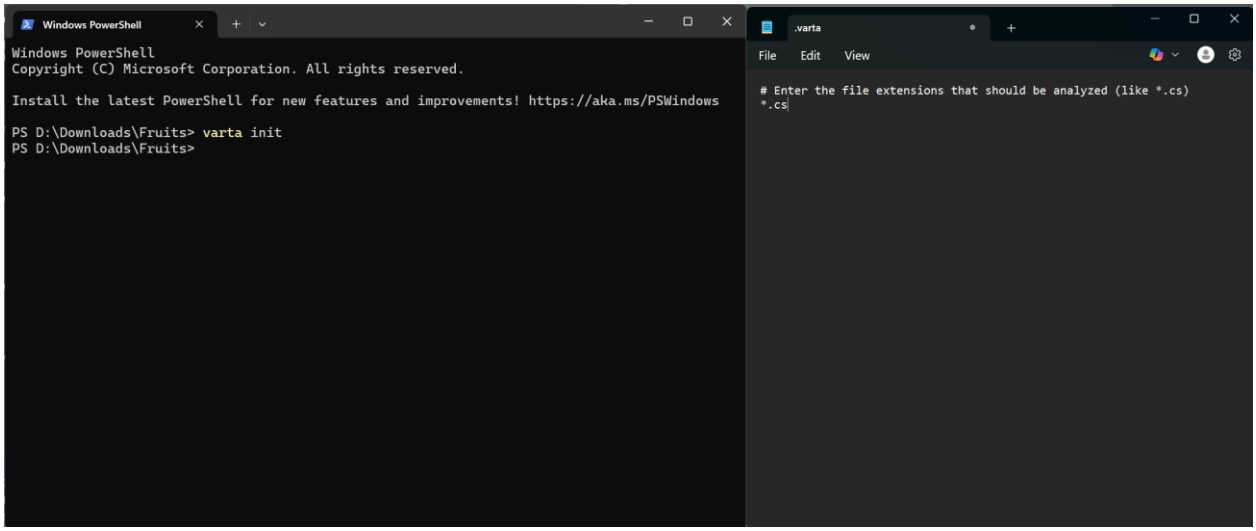


Рис. 3.11. Файл конфігурації проекту для роботи з Varta

Наступним кроком буде запуск операції сканування. Для цього треба прописати в консолі команду varta scan. Тим самим програма почне операцію сканування та перевірку кожного окремого файла на наявність кібервразливостей. Також буде автоматично класифіковано вразливості згудно з стандартом CWE.

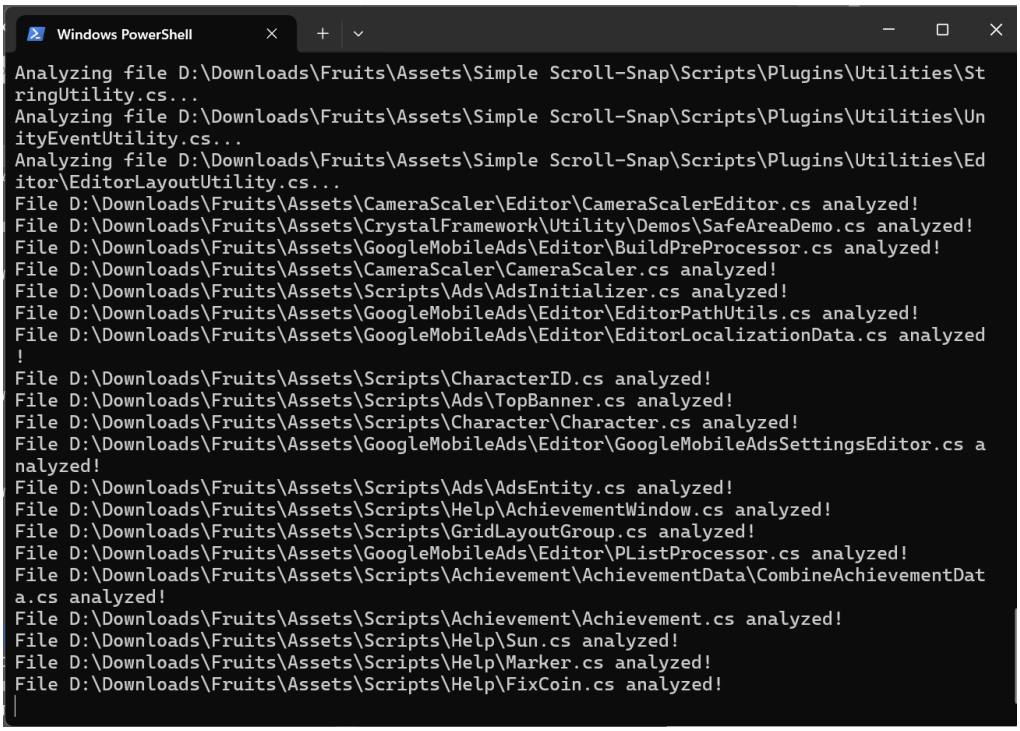


Рис. 3.12. Повідомлення в консолі під час сканування

Під час сканування на екран будуть виводитись назви файлів які вже пройшли перевірку. Після недовго процесу аналізу буде виведено табличку де вказано в яких файлах на яких рядках була знайдена вразливість.

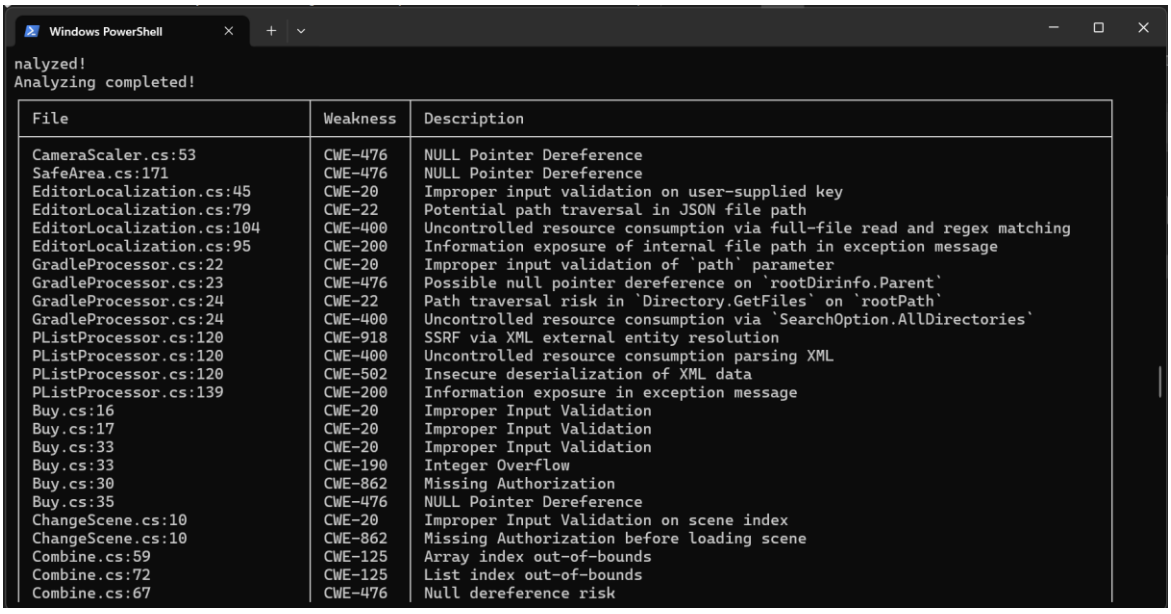


Рис. 3.13. Таблиця з результатами сканування

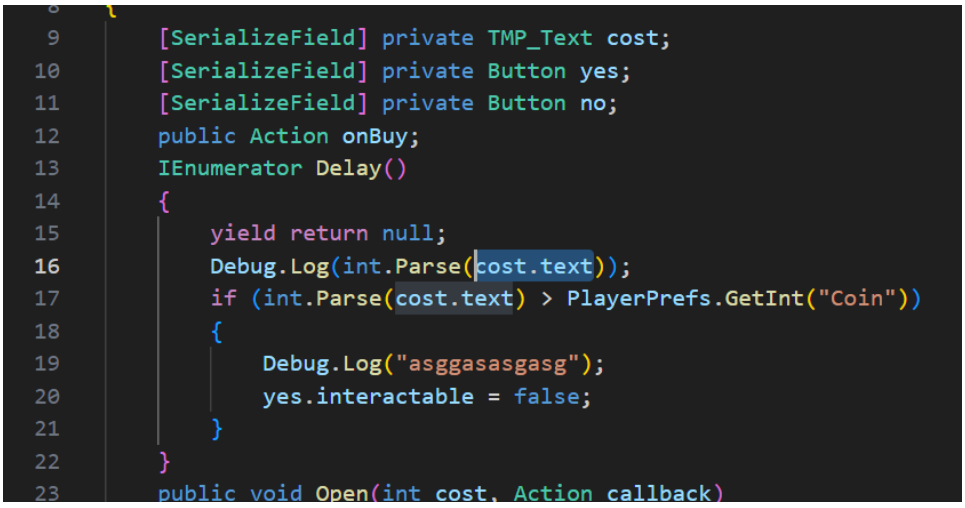


Рис. 3.13. Частина кодової бази де була знайдена вразливість. Buy.cs

На прикладі файлу Buy.cs підтверджується працездатність застосунку. В програмному коді на рядках 16-17 йде перетворення тексту в число без попередньої перевірки що відповідає вразливості CWE-20 Improper Input Validation.

ВИСНОВКИ

У сучасних умовах, коли геополітичні зміни відбуваються також у цифровому просторі, кіберзагрози набувають особливої гостроти. Зловмисники використовують автоматизовані методи для проведення масштабних атак, а обсяг програмного коду постійно зростає, що ускладнює ручний аналіз і підвищує ризик пропуску вразливостей. Саме тому впровадження систем на основі машинного навчання та штучного інтелекту стає необхідним для оперативного й надійного виявлення проблем безпеки в коді .

Метою цієї роботи було розробити систему автоматичного сканування кодової бази на наявність поточних вразливостей та прогнозування майбутніх на основі існуючих патернів. Для досягнення поставленої мети були визначені такі ключові завдання:

- аналіз існуючих методів аналізу коду;
- формулювання вимог до системи;
- розробка концепції та архітектури програмного продукту;
- вибір і впровадження компонентів штучного інтелекту;
- перевірка працездатності системи на реальних прикладах коду .

В результаті виконаної роботи запропоновано та реалізовано модульну архітектуру, що розділяє бізнес-логіку та деталі інтерфейсу, а також інтегровано мовну модель для аналізу й класифікації вразливостей. Запроваджений підхід дозволяє досягти значного прискорення процесу тестування на проникнення, мінімізувати людський фактор та забезпечити масштабованість рішення при роботі з великими кодовими базами.

Перспективою подальшого розвитку є розширення підтримуваних мов програмування, оптимізація витрат ресурсів при аналізі коду та інтеграція локальних моделей штучного інтелекту для роботи в умовах обмежених ресурсів. Це сприятиме підвищенню загальної безпеки програмних продуктів та забезпечить гнучкість системи в умовах постійних змін у ландшафті кіберзагроз.

ПЕРЕЛІК ПОСИЛАНЬ

1. Best vulnerability assessment & scanning tools [Електронний ресурс] // Unite.AI. — Режим доступу: https://www.unite.ai/uk/best-vulnerability-assessment-scanning-tools/ — Дата звернення: 14.05.2025.

2. 121.pdf [Електронний ресурс] // Електронна бібліотека ЗНТУ. — Режим доступу: https://eztuir.ztu.edu.ua/bitstream/handle/123456789/8762/121.pdf?sequence=1&isAllowed=y — Дата звернення: 14.05.2025.

3. CWE Top 25 Most Dangerous Software Errors — 2024 [Електронний ресурс] // MITRE. — Режим доступу: https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html — Дата звернення: 14.05.2025.

4. Ukraine Cybersec Market Review [Електронний ресурс] // IT Ukraine Association. — Режим доступу: https://itukraine.org.ua/files/Ukraine-Cybersec-Market-Review.pdf — Дата звернення: 14.05.2025.

5. Роль кібербезпеки: трохи [Електронний ресурс] // LB.ua. — Режим доступу: https://lb.ua/tech/2019/11/29/443542_rol_kiberbezopasnosti_nemnogo.html — Дата звернення: 14.05.2025.

6. Криптобіржа Bybit заявила, що її II… [Електронний ресурс] // LB.ua. — Режим доступу: https://lb.ua/economics/2025/02/21/661949_kriptobirzha_bybit_zayavila_shcho_ii.html — Дата звернення: 14.05.2025.

7. Salary – фахівець з кібербезпеки [Електронний ресурс] // Work.ua. — Режим доступу: https://www.work.ua/salary-фахівець+з+кібербезпеки/ — Дата звернення: 14.05.2025.

8. Overview of C# Tour [Електронний ресурс] // Microsoft Docs. — Режим доступу: https://learn.microsoft.com/uk-ua/dotnet/csharp/tour-of-csharp/overview — Дата звернення: 14.05.2025.

9. How much does GPT-4 cost? [Електронний ресурс] // OpenAI Help Center. — Режим доступу: https://help.openai.com/en/articles/7127956-how-much-does-gpt-4-cost — Дата звернення: 14.05.2025.

10. Tokenizer [Електронний ресурс] // OpenAI Platform. — Режим доступу: https://platform.openai.com/tokenizer — Дата звернення: 14.05.2025.

11. Gemini Tokenizer [Електронний ресурс] // gemini-tokenizer.site. — Режим доступу: https://www.gemini-tokenizer.site — Дата звернення: 14.05.2025.

12. Grok Tokenizer [Електронний ресурс] // Lunary.ai. — Режим доступу: https://lunary.ai/grok-tokenizer — Дата звернення: 14.05.2025.

13. Claude Tokenizer [Електронний ресурс] // Vercel. — Режим доступу: https://claude-tokenizer.vercel.app — Дата звернення: 14.05.2025.

14. Deepseek Tokenizer [Електронний ресурс] // Lunary.ai. — Режим доступу: https://lunary.ai/deepseek-tokenizer — Дата звернення: 14.05.2025.

15. Як ефективно використовувати нейромережу [Електронний ресурс] // Outsourcing.team. — Режим доступу: https://outsourcing.team/ua/blog/design/yak-efektivno-vikoristovuvati-nejromerezhu/#:~:text=Що%20таке%20промт,на%20ваші%20запитання%20інформативним%20способом. — Дата звернення: 14.05.2025.

16. HTTPS [Електронний ресурс] // Вікіпедія. — Режим доступу: https://uk.wikipedia.org/wiki/HTTPS — Дата звернення: 14.05.2025.

17. Як працює ІІ: принципи роботи сучасного AI [Електронний ресурс] // ItProger. — Режим доступу: https://itproger.com/ua/news/kak-rabotaet-ii-printsipi-raboti-sovremennogo-ai — Дата звернення: 14.05.2025.

18. Generative AI vs Other AI [Електронний ресурс] // Adobe. — Режим доступу: https://www.adobe.com/ua/products/firefly/discover/generative-ai-vs-other-ai.html — Дата звернення: 14.05.2025.

19. How to Write an AI-Readable Prompt [Електронний ресурс] // Skvot.io. — Режим доступу: https://skvot.io/uk/blog/how-to-write-an-ai-readable-prompt — Дата звернення: 14.05.2025.

20. The Clean Architecture [Електронний ресурс] // Uncle Bob's Blog. — Режим доступу: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html — Дата звернення: 14.05.2025.

21. Історія кібербезпеки: від зародження ідеї до наших днів [Електронний ресурс] // Education.ua. — Режим доступу: https://www.education.ua/blog/48055/ — Дата звернення: 14.05.2025.

22. Creeper: перший комп'ютерний вірус у світі [Електронний ресурс] // Binance. — Режим доступу: https://www.binance.com/uk-UA/square/post/786060 — Дата звернення: 14.05.2025.

23. 10 найбільш вражаючих кібератак в історії [Електронний ресурс] // ITTA.info. — Режим доступу: https://itta.info/10-najbilsh-vrazhayuchix-kiberatak-v-istorii/ — Дата звернення: 14.05.2025.

24. SonarQube [Електронний ресурс] // SonarSource. — Режим доступу: https://www.sonarsource.com/products/sonarqube/ — Дата звернення: 14.05.2025.

25. Checkmarx SAST Overview [Електронний ресурс] // Checkmarx. — Режим доступу: https://docs.checkmarx.com/en/34965-46311-checkmarx-sast-overview.html — Дата звернення: 14.05.2025.

26. Static Application Security Testing [Електронний ресурс] // OpenText. — Режим доступу: https://www.opentext.com/products/static-application-security-testing — Дата звернення: 14.05.2025.

27. OWASP Benchmark [Електронний ресурс] // OWASP. — Режим доступу: https://owasp.org/www-project-benchmark/ — Дата звернення: 14.05.2025.

28. Complete OWASP ZAP Guide [Електронний ресурс] // Medium. — Режим доступу: https://medium.com/@redfanatic7/complete-owasp-zap-guide-384a080ff502 — Дата звернення: 14.05.2025.

29. BurpSuite Pro Product Overview by E-SPIN [Електронний ресурс] // E-SPIN Corp. — Режим доступу: https://www.e-spincorp.com/burpsuite-pro-product-overview-by-e-spin/ — Дата звернення: 14.05.2025.

30. Assess [Електронний ресурс] // Contrast Security. — Режим доступу: https://docs.contrastsecurity.com/en/assess.html — Дата звернення: 14.05.2025.

31. hdiv/hdiv [Електронний ресурс] // GitHub. — Режим доступу: https://github.com/hdiv/hdiv — Дата звернення: 14.05.2025.

32. American Fuzzy Lop (software) [Електронний ресурс] // Wikipedia. — Режим доступу: https://en.wikipedia.org/wiki/American_Fuzzy_Lop_(software) — Дата звернення: 14.05.2025.

33. LibFuzzer [Електронний ресурс] // LLVM. — Режим доступу: https://llvm.org/docs/LibFuzzer.html — Дата звернення: 14.05.2025.

ДОДАТКИ

Додаток А

Лістинг: модуль VartaCLI скрипт Program.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using VartaCore;
using VartaGPT;

namespace VartaCLI
{
    0 references
    internal class Program
    {
        7 references
        private class SubCommand
        {
            3 references
            public string Name { get; }
            2 references
            public string Description { get; }
            2 references
            public Action<string[]> Action { get; }
            3 references
            public SubCommand(string name, string description, Action<string[]> action)
            {
                Name = name;
                Description = description;
                Action = action;
            }
        }
```

```
        static private ProjectAnalyzer analyzer;

        static private List<SubCommand> subCommands = new List<SubCommand>
        {
            new SubCommand("init", "Initialize Varta options", Init),
            new SubCommand("scan", "Analyze the project", Scan),
            new SubCommand("help", "Show help", Help)
        };
```

```
        0 references
        static void Main(string[] args)
        {
            try
            {
                AppInitializer.TryInitialize();
                analyzer = new ProjectAnalyzer(new GPTProvider(), Directory.GetCurrentDirectory());
                if (args.Length == 0)
                {
                    Help(null);
                    return;
                }
                string command = args[0];
                SubCommand subCommand = subCommands.FirstOrDefault(c => c.Name == command);
                if (subCommand == null)
                {
                    Console.WriteLine($"Unknown command: {command}");
                    Help(null);
                    return;
                }
                else
                {
                    subCommand.Action(args.Skip(1).ToArray());
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                return;
            }
        }
```

```
        3 references
        static private void Help(string[] args)
        {
            Console.WriteLine("Varta CLI - Command Line Interface");
            Console.WriteLine("Available commands:");
            Console.WriteLine("");
            foreach (var cmd in subCommands)
            {
                Console.WriteLine($"\t{cmd.Name}\t{cmd.Description}");
            }
            Console.WriteLine("");
        }

        1 reference
        static private void Init(string[] args)
        {
            analyzer.InitializeOptions();
        }

        1 reference
        static private async void Scan(string[] args)
        {
            try
            {
                List<Weakness> weaknesses = await analyzer.AnalyzeProject(Console.WriteLine);
                TablePrinter table = new TablePrinter("File", "Weakness", "Description");
                foreach (var weakness in weaknesses)
                {
                    table.AddRow($"{weakness.file}:{weakness.line}", weakness.code, weakness.description);
                }
                table.Print();
            }
            catch (ConfigDontFoundException ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("Please run 'varta init' to create a config file.");
                return;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                return;
            }
        }
    }
}
```

Додаток Б

Лістинг: модуль VartaCLI скрипт AppInitializator.cs

```
using System;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Security.Principal;

namespace VartaCLI
{
    1 reference
    internal class AppInitializer
    {
        3 references
        private static string ExeDirectory => Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location) ?? AppContext.BaseDirectory;

        1 reference
        public static void TryInitialize()
        {
            if (!IsPathInMachinePath())
            {
                if (IsAdministrator())
                {
                    AddCurrentExeDirectoryToMachinePath();
                }
                else
                {
                    throw new Exception("Administrator rights required. Please restart the application with elevated privileges.");
                }
            }
        }

        1 reference
        private static bool IsPathInMachinePath()
        {
            const EnvironmentVariableTarget target = EnvironmentVariableTarget.Machine;
            string current = Environment.GetEnvironmentVariable("PATH", target) ?? "";
            var parts = current.Split(';');
            return parts.Any(p => p.Equals(ExeDirectory, StringComparison.OrdinalIgnoreCase));
        }

        1 reference
        private static bool IsAdministrator()
        {
            using (var identity = WindowsIdentity.GetCurrent())
            {
                var principal = new WindowsPrincipal(identity);
                return principal.IsInRole(WindowsBuiltInRole.Administrator);
            }
        }

        1 reference
        private static void AddCurrentExeDirectoryToMachinePath()
        {
            Console.WriteLine("Completing Varta setup");
            const EnvironmentVariableTarget target = EnvironmentVariableTarget.Machine;
            string current = Environment.GetEnvironmentVariable("PATH", target) ?? "";
            var parts = current.Split(';');

            if (parts.Any(p => p.Equals(ExeDirectory, StringComparison.OrdinalIgnoreCase)))
            {
                Console.WriteLine("Path already exists in system PATH.");
                return;
            }

            string updated = current + ";" + ExeDirectory;
            Environment.SetEnvironmentVariable("PATH", updated, target);
            Console.WriteLine("Setup completed successfully.");
        }
    }
}
```

Додаток В

Лістинг: модуль VartaCLI скрипт TablePrinter.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace VartaCLI
{
    3 references
    internal class TablePrinter
    {
        private readonly List<string[]> _rows = new List<string[]>();
        6 references
        public string[] Headers { get; }

        1 reference
        public TablePrinter(params string[] headers)
        {
            if (headers == null || headers.Length == 0)
                throw new ArgumentException("At least one header is required.");
            Headers = headers;
        }

        1 reference
        public void AddRow(params string[] columns)
        {
            if (columns == null || columns.Length != Headers.Length)
                throw new ArgumentException($"Row must have {Headers.Length} values.");
            _rows.Add(columns);
        }
```

```
        1 reference
        public void Print()
        {
            int cols = Headers.Length;
            int winWidth = Console.WindowWidth;

            int[] maxContentWidths = new int[cols];
            for (int i = 0; i < cols; i++)
                maxContentWidths[i] = Headers[i].Length;
            foreach (var row in _rows)
                for (int i = 0; i < cols; i++)
                    maxContentWidths[i] = Math.Max(maxContentWidths[i], row[i].Length);

            int[] origWidths = (int[])maxContentWidths.Clone();

            int borderChars = cols + 1;
            int paddingChars = cols * 2;
            int availableForContent = winWidth - borderChars - paddingChars;

            int[] colWidths = new int[cols];
            int totalOrigWidth = 0;
            foreach (var w in origWidths) totalOrigWidth += w;
```

```

            if (totalOrigWidth <= availableForContent)
            {
                colWidths = (int[])origWidths.Clone();
            }
            else
            {
                int used = 0;
                for (int i = 0; i < cols; i++)
                {
                    colWidths[i] = Math.Max(1, origWidths[i] * availableForContent / totalOrigWidth);
                    used += colWidths[i];
                }
                int remaining = availableForContent - used;
                var widthsForPriority = (int[])origWidths.Clone();
                while (remaining > 0)
                {
                    int idx = 0;
                    for (int i = 1; i < cols; i++)
                        if (widthsForPriority[i] > widthsForPriority[idx]) idx = i;
                    colWidths[idx]++;
                    widthsForPriority[idx] = 0;
                    remaining--;
                }
            }

```

```
            List<string> Wrap(string text, int width)
            {
                var lines = new List<string>();
                if (string.IsNullOrEmpty(text))
                {
                    lines.Add(string.Empty);
                    return lines;
                }
                int pos = 0;
                while (pos < text.Length)
                {
                    int len = Math.Min(width, text.Length - pos);
                    lines.Add(text.Substring(pos, len));
                    pos += len;
                }
                return lines;
            }

            var headerBlocks = new List<List<string>>();
            int headerHeight = 0;
            for (int i = 0; i < cols; i++)
            {
                var lines = Wrap(Headers[i], colWidths[i]);
                headerBlocks.Add(lines);
                headerHeight = Math.Max(headerHeight, lines.Count);
            }
```

```
101
102             var wrappedRows = new List<List<List<string>>>();
103             var rowHeights = new List<int>();
104             foreach (var row in _rows)
105             {
106                 var cellLines = new List<List<string>>();
107                 int maxHeight = 0;
108                 for (int i = 0; i < cols; i++)
109                 {
110                     var lines = Wrap(row[i], colWidths[i]);
111                     cellLines.Add(lines);
112                     maxHeight = Math.Max(maxHeight, lines.Count);
113                 }
114                 wrappedRows.Add(cellLines);
115                 rowHeights.Add(maxHeight);
116             }
117
```

```
118             void PrintBorder(char left, char mid, char right)
119             {
120                 Console.Write(left);
121                 for (int i = 0; i < cols; i++)
122                 {
123                     Console.Write(new string('─', colWidths[i] + 2));
124                     Console.Write(i < cols - 1 ? mid : right);
125                 }
126                 Console.WriteLine();
127             }
128
129             PrintBorder('┌', '┬', '┐');
130
131             for (int line = 0; line < headerHeight; line++)
132             {
133                 Console.Write('│');
134                 for (int i = 0; i < cols; i++)
135                 {
136                     string cell = line < headerBlocks[i].Count ? headerBlocks[i][line] : string.Empty;
137                     Console.Write(' ' + cell.PadRight(colWidths[i]) + ' ' + '│');
138                 }
139                 Console.WriteLine();
140             }
141
142             PrintBorder('├', '┼', '┤');
```

```
143
144             for (int r = 0; r < wrappedRows.Count; r++)
145             {
146                 for (int line = 0; line < rowHeights[r]; line++)
147                 {
148                     Console.Write('│');
149                     for (int i = 0; i < cols; i++)
150                     {
151                         var lines = wrappedRows[r][i];
152                         string cell = line < lines.Count ? lines[line] : string.Empty;
153                         Console.Write(' ' + cell.PadRight(colWidths[i]) + ' ' + '│');
154                     }
155                     Console.WriteLine();
156                 }
157             }
158
159             PrintBorder('└', '┴', '┘');
160         }
161     }
162
163 }
164
```

Додаток Г

Лістинг: модуль VartaCore скрипт ProjectAnalyzer.cs

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Diagnostics;

namespace VartaCore
{
    3 references
    public class ProjectAnalyzer
    {
        private AIProvider aiProvider;
        private string projectRoot;

        5 references
        private string configPath => Path.Combine(projectRoot, ".varta");

        1 reference
        public ProjectAnalyzer(AIProvider aiProvider, string projectRoot)
        {
            this.aiProvider = aiProvider;
            this.projectRoot = projectRoot;
        }

        1 reference
        public void InitializeOptions()
        {
            if (!File.Exists(configPath))
            {
                CreateOptions();
                OpenOptionsInEditor();
            }
        }

        1 reference
        private void CreateOptions()
        {
            File.WriteAllText(configPath, "#Enter the file extensions that should be analyzed (like *.cs)");
        }


        private void OpenOptionsInEditor()
        {
            new Process
            {
                StartInfo = new ProcessStartInfo(configPath)
                {
                    UseShellExecute = true
                }
            }.Start();
        }


        private string[] GetFiles()
        {
            string[] extensions = GetOptions();
            List<string> files = new List<string>();
            foreach (string extension in extensions)
            {
                string[] filesWithExtension = Directory.GetFiles(projectRoot, extension, SearchOption.AllDirectories);
                files.AddRange(filesWithExtension);
            }
            return files.ToArray();
        }

```

```
        1 reference
        private string[] GetOptions()
        {
            if (!File.Exists(configPath))
            {
                throw new ConfigDontFoundException("Varta config file not found. Please run Varta in the project root directory.");
            }
            return File.ReadAllLines(configPath).Where(s => !s.StartsWith("#")).ToArray();
        }

        1 reference
        public async Task<List<Weakness>> AnalyzeProject(Action<string> log = null)
        {
            log?.Invoke("Analyzing project...");
            string[] files = GetFiles();
            List<Weakness> weaknesses = new List<Weakness>();
            foreach (string file in files)
            {
                weaknesses.AddRange(await AnalyzeFile(file, log));
            }
            log?.Invoke("Analyzing completed!");
            return weaknesses;
        }
```

```
        1 reference
        public async Task<List<Weakness>> AnalyzeFile(string path, Action<string> log = null)
        {
            log?.Invoke($"Analyzing file {path}...");
            List<RawWeakness> rawWeaknesses = await aiProvider.Analyze(path);
            string fileName = Path.GetFileName(path);
            List<Weakness> weaknesses = rawWeaknesses.Select(w => RawWeakness2Weakness(w, fileName)).ToList();
            log?.Invoke($"File {path} analyzed!");
            return weaknesses;
        }

        1 reference
        private Weakness RawWeakness2Weakness(RawWeakness rawWeakness, string file)
        {
            return new Weakness
            {
                code = rawWeakness.code,
                description = rawWeakness.description,
                line = rawWeakness.line,
                file = file
            };
        }
    }
}
```

Додаток Ґ

Лістинг: модуль VartaCore скрипт RawWeakness.cs

```
namespace VartaCore
{
    [System.Serializable]
    6 references
    public class RawWeakness
    {
        public string code;
        public string description;
        public int line;
    }
}
```

Додаток Д

Лістинг: модуль VartaCore скрипт Weakness.cs

```
namespace VartaCore
{
    8 references
    public class Weakness
    {
        public string code;
        public string description;
        public int line;
        public string file;
    }
}
```

Додаток Е

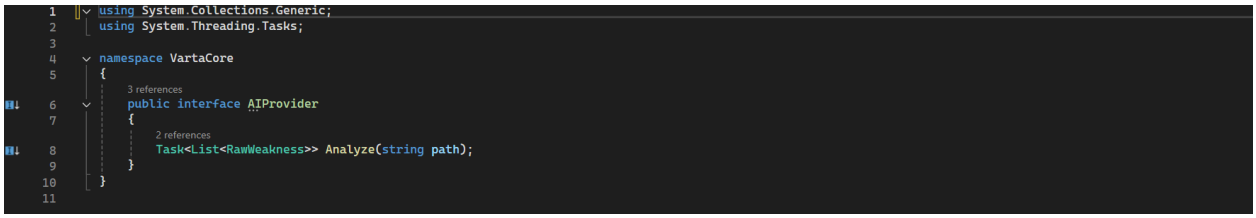
Лістинг: модуль VartaCore скрипт ConfigDontFoundException.cs

```
using System;

namespace VartaCore
{
    3 references
    public class ConfigDontFoundException : Exception
    {
        1 reference
        public ConfigDontFoundException(string message) : base(message)
        {
        }
    }
}
```

Додаток Є

Лістинг: модуль VartaCore скрипт AIProvider.cs



```
using System.Collections.Generic;
using System.Threading.Tasks;

namespace VartaCore
{
    3 references
    public interface AIProvider
    {
        2 references
        Task<List<RawWeakness>> Analyze(string path);
    }
}
```

Додаток Ж

Лістинг: модуль VartaGPT скрипт GPTProvider.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using VartaCore;
using OpenAI.Chat;
using Newtonsoft.Json;

namespace VartaGPT
{
    1 reference
    public class GPTProvider : AIProvider
    {
        private const string apiKey = "<apiKey KEY>";
        private const string prompt = @"prompt with {code}";

        2 references
        public async Task<List<RawWeakness>> Analyze(string path)
        {
            string prompt = GetPrompt(path);
            string answer = await GetAIAnswer(prompt);
            return GetWeaknessesFromAnswer(answer);
        }

        1 reference
        private string GetPrompt(string file)
        {
            string content = System.IO.File.ReadAllText(file);
            return prompt.Replace("{code}", content);
        }
```

```
        1 reference
        private async Task<string> GetAIAnswer(string prompt)
        {
            ChatClient client = new ChatClient(model: "o4-mini-2025-04-16", apiKey: apiKey);

            ChatCompletion completion = client.CompleteChat(prompt);

            return completion.Content[0].Text;
        }

        1 reference
        private List<RawWeakness> GetWeaknessesFromAnswer(string aiAnswer)
        {
            string formatedAnswer = aiAnswer;
            formatedAnswer = formatedAnswer.Replace("```json", "");
            formatedAnswer = formatedAnswer.Replace("```", "");
            return JsonConvert.DeserializeObject<List<RawWeakness>>(formatedAnswer);
        }

    }
}
```