Дослідження впливу генеративного штучного інтелекту на аналіз результатів статичного сканування коду за допомогою статичного аналізатора Bearer CLI

Study of the Influence of Generative Artificial Intelligence on the Analysis of Static Code Scanning Results Using the Bearer CLI Static Analyzer

Роман Жишко [A]
студент кафедри Захист інформації, e-mail: roman.zhyshko.kb.2021@lpnu.ua, ORCID: 0009-0001-9188-3931

Ігор Кос [A]
студент кафедри Захист інформації, e-mail: ihor.kos.kb.2021@lpnu.ua, ORCID: 0009-0001-4558-5036

Єлизавета Черемних [A]
студент кафедри Захист інформації, e-mail: yelyzaveta.cheremnykh.kb.2021@lpnu.ua, ORCID: 0009-0005-8901-7101

Даниїл Журавчак [A]
доктор філософії, асистент кафедри Захисту інформації, e-mail: danyil.y.zhuravchak@lpnu.ua, ORCID: 0000-0003-4989-0203

Віталій Сусукайло [A]
Corresponding author: доктор філософії, асистент кафедри Захисту інформації, e-mail: vitalii.a.susukailo@lpnu.ua, ORCID: 0000-0003-4431-9964

Roman Zhyshko [A]
Student of the Department Information Protection, e-mail: roman.zhyshko.kb.2021@lpnu.ua, ORCID: 0009-0001-9188-3931

Ihor Kos [A]
Student of the Department Information Protection, e-mail: ihor.kos.kb.2021@lpnu.ua, ORCID: 0009-0001-4558-5036

Yelyzaveta Cheremnykh [A]
Student of the Department Information Protection, e-mail: yelyzaveta.cheremnykh.kb.2021@lpnu.ua, ORCID: 0009-0005-8901-7101

Danyil Zhuravchak [A]
Doctor of Philosophy, Assistant, e-mail: danyil.y.zhuravchak@lpnu.ua, ORCID: 0000-0003-4989-0203

Vitalii Susukailo [A]
Corresponding author: Doctor of Philosophy, Assistant, e-mail: vitalii.a.susukailo@lpnu.ua, ORCID: 0000-0003-4431-9964

[A] Національний університет "Львівська політехніка", Львів, Україна

[A] Lviv Polytechnic National University, Lviv, Ukraine

Received: April 8, 2025 | Revised: April 28, 2025 | Accepted: April 30, 2025

DOI: 10.33445/sds.2025.15.2.17

Мета роботи: дослідити можливості генеративного штучного інтелекту для покращення інтерпретації результатів статичного сканування коду за допомогою інструмента Bearer CLI.

Метод дослідження: кількісні та експериментальні методи, зокрема застосування великих мовних моделей для обробки результатів SAST-аналізу, а також експертного оцінювання авторами їхньої точності, корисності, узгодженості та обґрунтованості оцінки наслідків ризиків за допомогою статистичного аналізу на основі 5-бальної шкали.

Результати дослідження: проведено оцінку п'яти популярних великих мовних моделей (GPT-4o, GPT-4.5, GPT-o3-mini-high, Gemini 2.5 Pro, Claude 3.7 Sonnet) щодо їхньої здатності покращувати аналіз результатів статичного сканування Bearer CLI для трьох типів вразливостей. Порівняльний аналіз за критеріями точності, корисності, узгодженості та оцінки бізнес-впливу показав, що інтеграція генеративного ШІ суттєво покращує інтерпретацію стандартного виводу інструмента. Найвищу ефективність за сукупністю критеріїв продемонструвала модель Gemini 2.5 Pro, що підтверджує значний потенціал ШІ для якісного та доступного аналізу SAST-звітів.

Теоретична цінність дослідження: дослідження поглиблює розуміння можливостей генеративного ШІ для ефективнішого аналізу результатів SAST-аналізаторів, демонструючи їх потенціал у наданні точніших пояснень та виправлень вразливостей, що є теоретичною основою для майбутніх досліджень в автоматизації безпеки коду.

Практична цінність дослідження: отримані результати можуть стати базою для поліпшення інструментів статичного аналізу, таких як Bearer CLI. Це допоможе розробникам швидше

Purpose: to investigate how generative artificial intelligence can improve static code analysis results interpretation using the Bearer CLI tool.

Method: quantitative and experimental methods, including using large language models for processing the results of SAST analysis, and expert evaluation by the authors of their accuracy, usefulness, consistency, and validity of assessing the consequences of risks using statistical analysis based on a 5-point scale.

Findings: five popular large language models (GPT-4o, GPT-4.5, GPT-o3-mini-high, Gemini 2.5 Pro, Claude 3.7 Sonnet) were evaluated for their ability to enhance the analysis of Bearer CLI static scanning results for three types of vulnerabilities. Comparative analysis based on criteria of accuracy, usefulness, consistency, and business impact assessment showed that integrating generative AI significantly improves the interpretation of the tool's standard output. The Gemini 2.5 Pro model demonstrated the best overall performance, confirming the significant potential of AI for improving the quality and accessibility of SAST report analysis.

Theoretical implications: The study deepens understanding of the capabilities of generative AI to more effectively analyze the results of SAST analyzers, demonstrating their potential in providing more accurate explanations and fixes for vulnerabilities, which is a theoretical basis for future research in code security automation.

Practical implications: the obtained results can serve as a basis for improving static analysis tools such as Bearer CLI. This will

орієнтуватися в результатах сканування і оперативно реагувати на потенційні загрози.

Цінність дослідження: дослідження демонструє, як застосування ШІ для аналізу результатів статичного сканування дозволяє знизити кількість хибнопозитивних спрацювань, зробити звіти більш зрозумілими та ефективно виявляти реальні вразливості, що сприяє підвищенню безпеки коду на ранніх етапах розробки.

Тип статті: емпіричне дослідження.

Ключові слова: штучний інтелект, статичний аналіз, мовна модель, безпека коду, вразливість.

help developers navigate scan results more efficiently and quickly respond to potential threats.

Value: The study demonstrates how using AI to analyze the results of static scanning can reduce the number of false positives, make reports more understandable and effectively identify real vulnerabilities, which contributes to improving code security in the early stages of development.

Paper type: empirical study.

Key words: artificial intelligence, static analysis, language model, code security, vulnerability.

Вступ

В умовах стрімкого розвитку технологій питання безпечної розробки програмного забезпечення є актуальним та критично важливим. Виявлення слабких місць коду за допомогою статичного аналізатора коду (SAST) дає змогу мінімізувати ризики ще на етапі розробки, дозволяючи виправляти вразливості, допоки витрати на усунення залишаються мінімальними, та навчати розробників створювати безпечний код [1]. При цьому інструменти SAST часто генерують хибні спрацювання, а звіти про аналіз є складними для розуміння, тому їх використання вимагає від розробників додаткових знань у сфері кібербезпеки. Навчання та здобуття кваліфікації співробітниками у цій галузі може бути часо- та фінансовоємним. Враховуючи зростаючу складність програмних систем та кількість кіберзагроз, пошук ефективних методів аналізу результатів SAST є вкрай важливим.

Для вирішення наведеної вище проблеми пропонується використання генеративного штучного інтелекту (ШІ). Великі мовні моделі (LLM) вже були неодноразово успішно протестовані на можливість виявлення вразливостей коду. Великі обсяги даних, на яких навчаються такі моделі, містять історичну інформацію про вразливості програм, що дає змогу знаходити аналогічні проблеми в нових зразках коду [2]. Генеративний ШІ може спростити аналіз результатів, пояснювати знайдені вразливості та надавати рекомендації щодо їх усунення, використовуючи термінологію, зрозумілу розробникам. Метою цього дослідження є оцінка впливу генеративного штучного інтелекту на процес аналізу результатів статичного сканування коду за допомогою Bearer CLI.

Одним із прикладів SAST-аналізатора є Bearer CLI – інструмент для статичного аналізу коду з відкритим вихідним кодом [3], який підтримує мови програмування Go, Java, JavaScript, PHP, Python та Ruby. Сканер використовує вбудований список правил для пошуку вразливостей та безпекових ризиків, базуючись на OWASP Top 10 та MITRE CWE Top 25, що містять систематизовану інформацію про найпоширеніші помилки безпеки програмного забезпечення. Очікується, що результати дослідження покажуть, як генеративний ШІ може підвищити ефективність аналізу результатів роботи Bearer CLI, зменшити кількість хибних спрацювань і покращити розуміння розробниками знайдених вразливостей.

Теоретичні основи дослідження

Дослідження впливу генеративного штучного інтелекту на аналіз результатів статичного сканування коду ґрунтується на сучасних теоретичних засадах статичного аналізу програмного забезпечення [4], засобах обробки природної мови та великих мовних моделей для генерації тексту. Статичне тестування безпеки додатків (англ. SAST) дозволяє виявляти потенційні вразливості та недоліки у коді програми без виконання самого коду [5, 6]. Результати сканування інструментів статичного тестування завжди містять технічну безпекову інформацію, яка є складною у розумінні для більшості розробників. У завданні інтерпретації результатів тестування можуть допомогти мовні моделі генеративного штучного інтелекту. Великі мовні моделі (LLM) – це дуже великі моделі глибокого навчання , які попередньо навчаються на величезних обсягах даних [7]. Найбільшими та найпотужнішими LLM є ті, які

працюють на основі трансформенної архітектури. Трансформенна архітектура – це нейронна мережа, яка вивчає контекст послідовних даних і генерує з нього нові дані. Простіше кажучи, трансформер – це тип моделі штучного інтелекту, який вчиться розуміти природну мову та генерувати її у людському стилі, аналізуючи шаблони у великих обсягах текстових даних [8]. Мовні моделі описують загальним терміном “штучний інтелект (ШІ)”. Завдяки навчанню на великих обсягах текстових даних моделі можуть перетворювати складні технічні виводи у доступні рекомендації щодо виправлення виявлених вразливостей. Найпотужнішими та найсучаснішими представниками генеративних моделей штучного інтелекту є моделі міркування (reasoning models) – моделі, навчені за допомогою навчання з підкріпленням для виконання складних міркувань [9]. Моделі міркування думають, перш ніж відповісти, створюючи довгий внутрішній ланцюжок думок, перш ніж відповісти користувачеві. Моделі міркування чудово підходять для вирішення складних проблем, кодування, наукового міркування та багатоетапного планування для агентських робочих процесів [10].

Bearer CLI — це статичний аналізатор коду (SAST) з відкритим кодом, який можна використовувати для сканування вихідного коду на наявність потенційних уразливостей. Аналіз відбувається за допомогою вбудованих правил безпеки та конфіденційності. Ці правила охоплюють типові вразливості, описані в OWASP Top 10 та CWE Top 25. Після завершення сканування генерується звіт з переліком знайдених вразливостей, їх детальним описом та рекомендаціями щодо виправлення. Також включається посилання на файли та рядки коду, де виявлені проблеми. Bearer CLI дозволяє сортувати виявлені проблеми за критичністю та типом, що спрощує їх подальшу обробку та виправлення [3].

Bearer CLI може виводити результати у різних форматах. Стандартний вивід передбачає виведення інформації безпосередньо в термінал із зазначенням виявлених проблем, їх описом та рекомендаціями. Збереження також можливе у JSON-формат, який дозволяє автоматизувати обробку результатів.

Інтеграція генеративного штучного інтелекту сприяє зниженню обсягів рутинної роботи та підвищує ефективність праці [11]. Таким чином, використання генеративного ШІ для вербального представлення результатів SAST відкриває нові можливості для покращення доступності технічної інформації та оптимізації процесів забезпечення безпеки програмного забезпечення.

Постановка проблеми

Основною проблемою, розглянутою в цій статті, є складність аналізу результатів статичного сканування коду через велику кількість специфічної технічної інформації та хибних спрацювань [12]. Це ускладнює виявлення реальних вразливостей, їх розуміння та, відповідно, методів їх усунення, а також вимагає від розробників додаткових знань у сфері кібербезпеки.

Інструменти SAST, як і Bearer CLI, часто створюють звіти, які важко інтерпретувати без глибокого досвіду у безпеці кіберпростору. У результаті розробники можуть ігнорувати деякі результати або витрачати значні ресурси на те, щоб розібратися, що конкретно їм потрібно виправити для усунення недоліку. Одним із найбільш перспективних рішень цієї проблеми є використання генеративного штучного інтелекту для автоматизації пояснення виявлених вразливостей та зменшення кількості помилок [13].

Незважаючи на потенціал генеративного ШІ у галузі статичного тестування безпеки додатків, його ефективність у цьому контексті ще недостатньо вивчена. Важливо дослідити та оцінити, як мовні моделі можуть покращити процес обробки результатів сканування SAST, щоб підвищити точність виявлення загроз, швидкість їх виправлення та зменшити навантаження на розробників програмного забезпечення, на що і націлене дане дослідження.

Методологія дослідження

Для оцінки впливу генеративного штучного інтелекту на результати роботи статичного аналізатора Bearer CLI було протестовано п’ять LLM моделей: GPT-4o, GPT-4.5, GPT o3-mini-high, Gemini 2.5 Pro, 3.7 Sonnet. Кожна з них обробляла однаковий набір вразливостей, представлених у стандартизованому форматі.

Щоб мінімізувати вплив можливих помилок токенізатора – засобу перетворення текстової інформації у вектори, зрозумілий формат для мовних моделей, – вхідні дані були перекладені англійською мовою. Використовувався наступний шаблон запиту (Рис. 1), що містить ключову інформацію про вразливість, включаючи її ідентифікатор, критичність, контекст у коді, а також запити на пояснення, оцінку можливих хибнопозитивних спрацьовувань, пропозицію виправлення та аналіз бізнес-наслідків.

```
Проаналізуй знахідку SAST.
Вхідні дані:
SAST Правило: [ID правила або Title]
CWE: [CWE ID, наприклад, 798]
Серйозність: [Рівень серйозності, наприклад, critical]
Файл: [Повний шлях до файлу]
Номер рядка: [Номер рядка]
Рядок: [Код в конкретному рядку]
Фрагмент коду:
[Вставте сюди ~3 рядки коду навколо вказаного рядка]
Завдання:
1. Поясни простими словами, що означає CWE [Вставте CWE ID].
2. Аналіз коду:
   а. Оціни ймовірність False Positive. Що потрібно перевірити
      в коді чи логіці застосунку, щоб це підтвердити або спростувати?
   б. Поясни, чому саме цей фрагмент коду є проблемою в даному контексті.
   в. Надай виправлений фрагмент коду конкретно для цього місця.
3. Бізнес-вплив: Оціни потенційні бізнес-наслідки (наприклад, витік даних,
      фінансові чи репутаційні втрати), якщо цю вразливість проексплуатують,
      враховуючи серйозність та контекст.
```

Рисунок 1 – Шаблон запиту для генеративного ШІ

Для тестування було обрано три унікальні вразливості різних категорій, виявлені в проєкті OWASP Juice Shop при статичному тестуванні. Вивід, покращений мовними моделями, було порівняно між собою, а також з оригінальною версією результатів інструмента SAST. Оцінювання здійснювалося вручну авторами даного матеріалу за критеріями **точності пояснень** (правильність та повнота), **корисності виправлень** (ефективність та безпечність), **узгодженості відповідей** (відповідність структурі запиту) та **релевантності оцінки бізнес-впливу** (обґрунтованість наслідків). Для оцінки виводу кожної з моделей використовувалась 5-бальна шкала, де 1 – повна невідповідність критерію або незадовільна якість, а 5 – відмінна відповідь, що повністю задовольняє критерій.

Отримані результати дозволили визначити, яка з протестованих LLM показала найкращі результати за визначеними критеріями та оцінити потенціал використання генеративного ШІ для покращення аналізу результатів статичного аналізу коду.

Результати

Для порівняння аналізу моделями результатів сканування було обрано три вразливості різних рівнів.

Розглянемо оригінальний вивід SAST-аналізатора Bearer CLI для **першої вразливості javascript_express_hardcoded_secret**, форматований у JSON. Він містить певну корисну інформацію, однак пояснення неповні – не розкрито всі суттєві аспекти ризику. Рекомендації

з виправлення містять правильні поради (наприклад, застосовувати змінні середовища), але вони подані поверхнево та немає конкретного коду безпечної реалізації. Щодо узгодженості відповідей – структура повністю розбалансована, а аналіз бізнес-наслідків абсолютно відсутній. Таким чином, оригінальний вивід недостатньо інформативний і не відповідає вимогам якісного аналізу вразливостей. Оскільки для наступних вразливостей застосовано такий ж принцип опису, їх ми окремо розглядати не будемо.


```
{"critical":[{"cwe_ids":["798"],"id":"javascript_express_hardcoded_secret",
"title":"Usage of hard-coded secret","description":"## Description\n\nStoring
secrets directly in code compromises security. It's safer to use environment
variables or a secret management system.\n\n## Remediations\n\n- Do not store
plaintext secrets in your code. This makes your application vulnerable to
unauthorized access if the codebase is exposed.\n  ```javascript\n  app.use(
\n    session({\n      secret: \"shh-my-secret\",\n      name: \
"my-custom-session-name\",\n    })\n  )\n  ```\n- Do use environment variables
to store secrets. This method keeps sensitive information out of your codebase.
\n  ```javascript\n  app.use(\n    session({\n      secret: process.env.SECRET,
\n      name: \"my-custom-session-name\",\n    })\n  )\n  ```\n- Do use a
secret management system or a key management service (KMS) with encryption for
enhanced security. These services provide secure storage and management of
secrets, reducing the risk of exposure.\n\n## References\n\n- [OWASP hardcoded
passwords](https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password)
\n- [Google Cloud Key Management Service](https://cloud.google.com/kms/docs)
\n- [AWS Key Management Service](https://aws.amazon.com/kms/)\n","documentation_url":
"https://docs.bearer.com/reference/rules/javascript_express_hardcoded_secret",
"line_number":56,"full_filename":"juice-shop/lib/insecurity.ts",
"filename":"lib/insecurity.ts","category_groups":["PII","Personal Data"],
"source":{"start":56,"end":56,"column":{"start":41,"end":108}},"sink":{
"start":56,"end":56,"column":{"start":41,"end":108},"content":""},
"parent_line_number":56,"fingerprint":"c095a5cc1948bae67c38b5c5d85eb4f4_0",
"old_fingerprint":"6841f0355c0a7e03b3d87c5c53f54493_0","code_extract":"export
const authorize = (user = {}) =\u003e jwt.sign(user, privateKey, { expiresIn:
'6h', algorithm: 'RS256' })"},...]}
```


Рисунок 2 – Оригінальний вивід аналізатора Bearer CLI у форматі JSON

Для аналізу вразливості було сформульовано запит, який відповідає наступним критеріям: точність пояснень, корисність виправлень, узгодженість відповідей та релевантність оцінки бізнес-впливу.

```
Analyze the SAST output.
Input:
1. SAST Rule: javascript_express_hardcoded_secret
2. CWE: 798
3. Severity: critical
4. File: juice-shop/lib/insecurity.ts
5. Line number: 56
6. Line: export const authorize = (user = {}) => jwt.sign(
   user, privateKey, { expiresIn: '6h', algorithm: 'RS256' })
7. Code Snippet: export const isAuthorized = () => expressJwt(
     ({ secret: publicKey }) as any)
   export const denyAll = () => expressJwt(
     { secret: '' + Math.random() } as any)
   export const authorize = (user = {}) => jwt.sign(user, privateKey,
     { expiresIn: '6h', algorithm: 'RS256' })
   export const verify = (token: string) => token ? (jws.verify as (
     (token: string, secret: string) => boolean))(token, publicKey) : false
   export const decode = (token: string) => { return jws.decode(token)?.payload }
Tasks:
1. Explain in simple terms what CWE 798 means.
2. Code Analysis:
   a. Estimate the probability of a False Positive. What do you need to check in
      the code or logic of the application to confirm or deny this?
   b. Explain why this particular code snippet is a problem in this context.
   c. Provide a corrected code snippet for this particular location.
3. Business Impact: Estimate the potential business impact (e.g., data breach,
      financial or reputational loss) if this vulnerability is exploited, taking
      into account the severity and context.
```

Рисунок 3 – Запит для вразливості javascript_express_hardcoded_secret

Було детально розглянуто результати моделей по першій вразливості – javascript_express_hardcoded_secret. CWE-798: “Use of Hard-coded Credentials” означає, що облікові дані, наприклад паролі, секретні ключі, ключі API або інші облікові дані зберігаються безпосередньо у вихідному коді програми. Це робить систему дуже вразливою, оскільки будь-хто, хто має доступ до коду (розробники та потенційні зловмисники), може легко знайти та використати ці секрети.

GPT-4o: модель впоралась добре. Її відповідь здатна допомогти у аналізі вразливості для фахового розробника. Точність пояснень – хороша, дає розуміння проблеми, її причини та наслідки. Корисність виправлень – середня, запропоновані виправлення допоможуть в усуненні вразливості, однак самі виправлення – поверхневі та недостатньо детальні. Оцінка бізнес-впливу – релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

GPT-4.5: модель впоралась відмінно. Її відповідь також цілком здатна допомогти у аналізі вразливості для фахового розробника. Точність пояснень – хороша, розробник розумітиме причини виникнення та наслідки . Корисність виправлень – дуже висока, запропоновані виправлення допоможуть в усуненні вразливості, модель також додала додаткову обробку помилок у разі відсутності секретних ключів у змінних середовища. Оцінка бізнес-впливу – повністю релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

GPT-o3-mini-high: модель впоралась добре. Її відповідь також цілком здатна допомогти у аналізі вразливості для фахового розробника. Точність пояснень – задовільна, розробник поверхнево розумітиме проблеми, пов’язані з вразливістю. Корисність виправлень – дуже висока, запропоновані виправлення допоможуть в усуненні вразливості, модель також додала додаткову обробку помилок у разі відсутності секретних ключів у змінних середовища. Оцінка бізнес-впливу – повністю релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

Gemini 2.5 Pro: модель впоралась винятково, її результат – найкращий у порівнянні з іншими моделями. Її відповідь здатна повністю допомогти у аналізі вразливості для фахового розробника. Точність пояснень – відмінна, розробник матиме детальне але водночас просте пояснення, повне розуміння причин виникнення та наслідків експлуатації вразливості. Корисність виправлень – дуже висока, запропоновані виправлення допоможуть в усуненні вразливості, модель також додала додаткову обробку помилок у разі відсутності секретних ключів у змінних середовища. Оцінка бізнес-впливу – повністю релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

Claude 3.7 Sonnet: модель впоралась добре. Її відповідь також цілком здатна допомогти у аналізі вразливості для фахового розробника. Точність пояснень – хороша, розробник розумітиме причини виникнення та наслідки. Корисність виправлень – дуже висока, запропоновані виправлення допоможуть в усуненні вразливості, модель також додала додаткову обробку помилок у разі відсутності секретних ключів у змінних середовища. Оцінка бізнес-впливу – достатньо релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

Перейдемо до **другої вразливості — javascript_express_external_file_upload**. Аналогічно було сформульовано запит та проаналізовано результати п’яти моделей.

```
Analyze the SAST output.
Input:
1. SAST Rule: javascript_express_external_file_upload
2. CWE: 73
3. Severity: high
4. File: juice-shop/routes/quarantineServer.ts
5. Line number: 14
6. Line: res.sendFile(path.resolve('ftp/quarantine/', file))
7. Code Snippet: if (!file.includes('/')) {
      res.sendFile(path.resolve('ftp/quarantine/', file))
    } else {
      res.status(403)
      next(new Error('File names cannot contain forward slashes!'))
    }
Tasks:
1. Explain in simple terms what CWE [Insert CWE ID] means.
2. Code Analysis:
   a. Estimate the probability of a False Positive. What do you need
      to check in the code or logic of the application to confirm or deny this?
   b. Explain why this particular code snippet is a problem in this context.
   c. Provide a corrected code snippet for this particular location.
3. Business Impact: Estimate the potential business impact (e.g., data breach,
      financial or reputational loss) if this vulnerability is exploited, taking
      into account the severity and context.
```

Рисунок 4 – Запит для вразливості javascript_express_external_file_upload

CWE-73: “External Control of File Name or Path” означає, що вихідний код містить небезпечні логічні помилки, які дозволяють досягати файлів за вказанням шляху без його перевірки, завдяки чому зловмисник може вийти за межі призначеної для нього директорії та отримати доступ до інших файлів або каталогів на сервері, до яких він не повинен мати доступу.

GPT-4o: модель впоралась відмінно. Її відповідь здатна повністю допомогти у аналізі вразливості для фахового розробника. Точність пояснень – відмінна та дає повне розуміння проблеми, її причини та наслідки. Корисність виправлень – також відмінна, запропоновані виправлення допоможуть в усуненні вразливості. Оцінка бізнес-впливу – релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

GPT-4.5: модель впоралась задовільно. Її відповідь здатна допомогти у аналізі вразливості для фахового розробника. Точність пояснень – хороша, розробник розумітиме причини виникнення та наслідки. Корисність виправлень – висока, запропоновані виправлення допоможуть в усуненні вразливості. Оцінка бізнес-впливу – релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

GPT-o3-mini-high: модель впоралась відмінно. Її відповідь також цілком здатна допомогти у аналізі вразливості для фахового розробника. Точність пояснень – відмінна та дає повне розуміння проблеми, її причини та наслідки. Корисність виправлень – висока, запропоновані виправлення допоможуть в усуненні вразливості. Оцінка бізнес-впливу – повністю релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

Gemini 2.5 Pro: модель впоралась винятково, для цієї вразливості її результат – теж найкращий у порівнянні з іншими моделями. Її відповідь здатна повністю допомогти у аналізі вразливості для фахового розробника. Точність пояснень – відмінна, розробник матиме детальне але водночас просте пояснення, повне розуміння причин виникнення та наслідків експлуатації вразливості. Корисність виправлень – дуже висока, запропоновані виправлення допоможуть в усуненні вразливості, модель дуже детально написала код який запобігає вразливості. Оцінка бізнес-впливу – повністю релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

Claude 3.7 Sonnet: модель впоралась задовільно. Її відповідь може допомогти у аналізі вразливості для фахового розробника. Точність пояснень – задовільна, розробник розумітиме причини виникнення та наслідки, але пояснення короткі та поверхневі. Корисність виправлень – висока, запропоновані виправлення допоможуть в усуненні вразливості. Оцінка бізнес-впливу – достатньо релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

Перейдемо до **третьої вразливості — javascript_express_open_redirect**. Як і в попередніх випадках, було сформульовано відповідний запит та проаналізовано результати п’яти мовних моделей.

```
Analyze the SAST output.
Input:
1. SAST Rule: javascript_express_open_redirect
2. CWE: 601
3. Severity: medium
4. File: juice-shop/routes/redirect.ts
5. Line number: 19
6. Line: res.redirect(toUrl)
7. Code Snippet: if (security.isRedirectAllowed(toUrl)) {
      challengeUtils.solveIf(challenges.redirectCryptoCurrencyChallenge, () => {
        return toUrl === 'https://explorer.dash.org/address/Xr556RzuwX6hg5EGpkybbv5RanJoZN17kW'
        || toUrl === 'https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm'
        || toUrl === 'https://etherscan.io/address/0x0f933ab9fcaaa782d0279c300d73750e1311eae6' })
      challengeUtils.solveIf(challenges.redirectChallenge,
        () => { return isUnintendedRedirect(toUrl) })
      res.redirect(toUrl)
    } else {
      res.status(406)
      next(new Error('Unrecognized target URL for redirect: ' + toUrl))
    }

Tasks:
1. Explain in simple terms what CWE [Insert CWE ID] means.
2. Code Analysis:
   a. Estimate the probability of a False Positive. What do you need
      to check in the code or logic of the application to confirm or deny this?
   b. Explain why this particular code snippet is a problem in this context.
   c. Provide a corrected code snippet for this particular location.
3. Business Impact: Estimate the potential business impact (e.g., data breach,
      financial or reputational loss) if this vulnerability is exploited, taking
      into account the severity and context.
```

Рисунок 5 – Запит для вразливості javascript_express_open_redirect

CWE-601: “URL Redirection to Untrusted Site ('Open Redirect')” означає, що вразливість виникає коли веб-додаток перенаправляє користувача на іншу URL-адресу, яка вказана у вхідному параметрі запиту, але не перевіряє належним чином, чи є ця цільова URL-адреса безпечною та довіреною. Зловмисник може створити посилання, яке виглядає так, ніби веде на легітимний сайт, але насправді містить параметр, що перенаправляє користувача на фішинговий або небезпечний сайт. Користувач, довіряючи початковому посиланню, може перейти за ним і стати жертвою атаки.

GPT-4o: модель впоралась добре. Її відповідь здатна допомогти у аналізі вразливості для фахового розробника. Точність пояснень – відмінна, лаконічно пояснює проблему, її причини та можливі наслідки. Корисність виправлень – задовільна, запропоновані виправлення – поверхневі та недостатньо детальні, вони повноцінно не усувають проблему. Оцінка бізнес-впливу – релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

GPT-4.5: модель впоралась задовільно. Її відповідь загалом допоможе з усуненням уразливості, але не розкриває причини і наслідки проблеми. Точність пояснень – задовільна, фокусується лише на одній загрозі, пов’язаній з уразливістю. Корисність виправлень – хороша, виправлення містить додаткову обробку помилок. Оцінка бізнес-впливу – неточна, упущено декілька критичних ризиків.

GPT-o3-mini-high: модель впоралась добре. Її відповідь дозволить розробнику зрозуміти суть уразливості й усунути її. Точність пояснень – середня, надає розуміння уразливості та способи визначення хибнопозитивності спрацювання, але на відміну від інших

моделей, сам цього не оцінює. Корисність виправлень – хороша, виправлений код також містить додаткову обробку помилок. Оцінка бізнес-впливу – цілком релевантна, надає повне розуміння дає розуміння, які можуть бути наслідки експлуатації вразливості.

Gemini 2.5 Pro: модель впоралась винятково, для цієї вразливості її результат – теж найкращий у порівнянні з іншими моделями. Її відповідь здатна повністю допомогти у аналізі вразливості для фахового розробника. Точність пояснень – відмінна, розробник матиме детальне але водночас просте пояснення, повне розуміння причин виникнення та наслідків експлуатації вразливості. Корисність виправлень – дуже висока, запропоновані виправлення допоможуть в усуненні вразливості, модель дуже детально написала код який запобігає вразливості. Оцінка бізнес-впливу – повністю релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

Claude 3.7 Sonnet: модель впоралась задовільно. Її відповідь може допомогти у аналізі вразливості для фахового розробника. Точність пояснень – задовільна, розробник розумітиме причини виникнення та наслідки, але пояснення короткі та поверхневі. Корисність виправлень – висока, запропоновані виправлення допоможуть в усуненні вразливості. Оцінка бізнес-впливу – достатньо релевантна та дає розуміння, які можуть бути наслідки експлуатації вразливості.

Після аналізу усіх результатів роботи генеративних моделей ШІ, було проведено оцінювання згідно з чотирма критеріями – точність пояснень, корисність виправлень, узгодженість відповідей і релевантність оцінки бізнес-впливу за п’ятибальною шкалою, та винесено у окрему таблицю (табл. 1). Варто відзначити, що всі протестовані моделі продемонстрували високий рівень узгодженості відповідей, чітко дотримуючись заданої структури запиту. Найкращі результати за більшістю критеріїв продемонструвала Gemini 2.5 Pro, що отримала найвищий загальний бал. Інші моделі також показали конкурентоспроможні результати, проте мали певні недоліки, зокрема в точності пояснень або оцінці бізнес-впливу.

Таблиця 1 – Результати оцінювання роботи генеративних моделей ШІ за критеріями

	Критерії	GPT-4o	GPT-4.5	GPT-o3-mini-high	Gemini 2.5 Pro	Claude 3.7
Вразливість 1 (CWE-798)	точність пояснень	4	4	3	5	4
	корисність виправлень	3	5	5	5	5
	узгодженість відповідей	5	5	5	5	5
	релевантність оцінки бізнес-впливу	4	5	5	5	4
Вразливість 2 (CWE-73)	точність пояснень	5	4	5	5	3
	корисність виправлень	5	4	4	5	4
	узгодженість відповідей	5	5	5	5	5

	Критерії	GPT-4o	GPT-4.5	GPT-o3-mini-high	Gemini 2.5 Pro	Claude 3.7
	релевантність оцінки бізнес-впливу	4	4	5	5	4
Вразливість 3 (CWE-601)	точність пояснень	5	4	4	5	3
	корисність виправлень	3	4	4	5	4
	узгодженість відповідей	5	5	5	5	5
	релевантність оцінки бізнес-впливу	4	2	5	5	4
Фінальний бал:		**52**	**51**	**55**	**60**	**50**

Висновки

У результаті проведеного дослідження було проаналізовано вплив генеративного штучного інтелекту на аналіз результатів роботи SAST інструмента Bearer CLI. З метою порівняльної оцінки можливостей різних мовних моделей у контексті досліджуваної проблематики було розглянуто GPT-4o, GPT-4.5, GPT-o3-mini-high, Gemini 2.5 Pro та Claude 3.7.

Порівняльний аналіз показав, що найкращий результат щодо точності, зрозумілості та повноти пояснень було досягнуто за використання моделі Gemini 2.5 Pro, яка забезпечила найкращі пояснення з якісними рекомендаціями щодо виправлення вразливостей. Якісні результати показала також модель GPT-o3-mini-high. Дві попередньо названі моделі показують тенденцію, що моделі міркування демонструють найбільш точні та якісні результати. Решта моделей також показали хороший результат, який покращує розуміння результатів статичного аналізу людьми, непов'язаними з кібербезпекою.

Вище описані результати свідчать про потенціал використання генеративного ШІ для аналізу результатів SAST, що актуально для людей, які не мають достатньо глибоких знань у галузі кібербезпеки. Також дослідження підтвердило доцільність застосування генеративних моделей для надання зрозумілих рекомендацій з виправлення вразливостей.

Запропоновані рекомендації для покращення ефективності використання генеративного ШІ включають: випробування більш спеціалізованих моделей або адаптація вже доступних під конкретні вимоги інструментів статичного сканування, що може включати розробку автоматизованих агентів ШІ для зручності використання. Хорошим рішенням буде інтеграція генеративних моделей у процеси CI/CD для автоматизованого формування пояснень в реальному часі.

Фінансування

Це дослідження не отримало конкретної фінансової підтримки.

Конкуруючі інтереси

Автори заявляють, що у них немає конкуруючих інтересів.

Список використаних джерел

1. OpenText. What is SAST? [Електронний ресурс]. Available from: https://www.opentext.com/what-is/sast. Accessed: April 1, 2025.
2. Ahmad, B., Tan, B., Karri, R., & Pearce, H. (2023). FLAG: Finding line anomalies (in code) with generative AI. arXiv. https://doi.org/10.48550/arXiv.2306.12643
3. Bearer. Bearer CLI documentation [Електронний ресурс]. Available from: https://docs.bearer.com. Accessed: April 1, 2025.
4. Markson, Reed & Samson, Mike & Owen, Antony. (2023). Automated Code Review: Leveraging Generative AI for Vulnerability Detection in Software Development. ResearchGate. https://www.researchgate.net/publication/390108092_Automated_Code_Review_Leveraging_Generative_AI_for_Vulnerability_Detection_in_Software_Development
5. Check Point. What is Static Application Security Testing (SAST)? [Електронний ресурс]. Available from: https://www.checkpoint.com/cyber-hub/cloud-security/what-is-static-application-security-testing-sast. Accessed: April 1, 2025.
6. Sharma, R. An Analysis of Generative AI Capabilities in Security Testing. Evaluating Static Code Analysis Performance. Digitala Vetenskapliga Arkivet. https://www.diva-portal.org/smash/get/diva2:1941555/FULLTEXT01.pdf
7. Amazon Web Services. What is a large language model? [Електронний ресурс]. Available from: https://aws.amazon.com/what-is/large-language-model. Accessed: April 1, 2025.
8. DataCamp. How transformers work [Електронний ресурс]. Available from: https://www.datacamp.com/tutorial/how-transformers-work. Accessed: April 1, 2025.
9. Yuanjiang Cao, Quan Z. Sheng, Julian McAuley & Lina Yao. (2021). Reinforcement Learning for Generative AI: A Survey. Journal Of Latex Class Files, 14(8). https://doi.org/10.48550/arXiv.2308.14328
10. OpenAI. Guides: Reasoning [Електронний ресурс]. Available from: https://platform.openai.com/docs/guides/reasoning?api-mode=responses. Accessed: April 1, 2025.
11. Ding, A. et al., "Generative AI for Software Security Analysis: Fundamentals, Applications, and Challenges" in IEEE Software, vol. 41, no. 06, pp. 46-54, Nov.-Dec. 2024 .https://doi.ieeecomputersociety.org/10.1109/MS.2024.3416036
12. Hicken, A. (2023, 12 травня). False Positives in Static Code Analysis. Parasoft. https://www.parasoft.com/blog/false-positives-in-static-code-analysis/
13. Fan, Gang, Xie, Xiaoheng, Zheng, Xunjin, Liang, Yinan, and Di, Peng. (2023). Static Code Analysis in the AI Era: An In-depth Exploration of the Concept, Function, and Potential of Intelligent Code Analysis Agents. arXiv. https://doi.org/10.48550/arXiv.2310.08837

References

1. OpenText. What is SAST? [Електронний ресурс]. Available from: https://www.opentext.com/what-is/sast. Accessed: April 1, 2025.
2. Ahmad, B., Tan, B., Karri, R., & Pearce, H. (2023). FLAG: Finding line anomalies (in code) with generative AI. arXiv. https://doi.org/10.48550/arXiv.2306.12643
3. Bearer. Bearer CLI documentation [Електронний ресурс]. Available from: https://docs.bearer.com. Accessed: April 1, 2025.
4. Markson, Reed & Samson, Mike & Owen, Antony. (2023). Automated Code Review: Leveraging Generative AI for Vulnerability Detection in Software Development. ResearchGate. https://www.researchgate.net/publication/390108092_Automated_Code_Review_Leveraging_Generative_AI_for_Vulnerability_Detection_in_Software_Development

5. Check Point. What is Static Application Security Testing (SAST)? [Електронний ресурс]. Available from: https://www.checkpoint.com/cyber-hub/cloud-security/what-is-static-application-security-testing-sast. Accessed: April 1, 2025.
6. Sharma, R. An Analysis of Generative AI Capabilities in Security Testing. Evaluating Static Code Analysis Performance. Digitala Vetenskapliga Arkivet. https://www.diva-portal.org/smash/get/diva2:1941555/FULLTEXT01.pdf
7. Amazon Web Services. What is a large language model? [Електронний ресурс]. Available from: https://aws.amazon.com/what-is/large-language-model. Accessed: April 1, 2025.
8. DataCamp. How transformers work [Електронний ресурс]. Available from: https://www.datacamp.com/tutorial/how-transformers-work. Accessed: April 1, 2025.
9. Yuanjiang Cao, Quan Z. Sheng, Julian McAuley & Lina Yao. (2021). Reinforcement Learning for Generative AI: A Survey. Journal Of Latex Class Files, 14(8). https://doi.org/10.48550/arXiv.2308.14328
10. OpenAI. Guides: Reasoning [Електронний ресурс]. Available from: https://platform.openai.com/docs/guides/reasoning?api-mode=responses. Accessed: April 1, 2025.
11. Ding, A. et al., "Generative AI for Software Security Analysis: Fundamentals, Applications, and Challenges" in IEEE Software, vol. 41, no. 06, pp. 46-54, Nov.-Dec. 2024 .https://doi.ieeecomputersociety.org/10.1109/MS.2024.3416036
12. Hicken, A. (2023, 12 травня). False Positives in Static Code Analysis. Parasoft. https://www.parasoft.com/blog/false-positives-in-static-code-analysis/
13. Fan, Gang, Xie, Xiaoheng, Zheng, Xunjin, Liang, Yinan, and Di, Peng. (2023). Static Code Analysis in the AI Era: An In-depth Exploration of the Concept, Function, and Potential of Intelligent Code Analysis Agents. arXiv. https://doi.org/10.48550/arXiv.2310.08837