УДК 004.056.5:004.89

І. А. Фіглярський
Л.М. Куперштейн

АВТОМАТИЗАЦІЯ АНАЛІЗУ БЕЗПЕКИ КОДУ НА ОСНОВІ ГЕНЕРАТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ

Вінницький національний технічний університет;

Анотація

Досліджено можливості застосування генеративного штучного інтелекту, зокрема великих мовних моделей, для автоматизації аналізу безпеки програмного коду. Проведено порівняльний аналіз традиційних засобів статичного аналізу та підходів на основі LLM. Визначено переваги та обмеження генеративних моделей у задачах ідентифікації вразливостей, класифікації типів загроз та генерування рекомендацій щодо виправлення небезпечних фрагментів коду. Запропоновано гібридний підхід, що поєднує SAST-інструменти з аналізом на основі LLM із використанням технології RAG.

Ключові слова: аналіз безпеки коду, генеративний штучний інтелект, великі мовні моделі, статичний аналіз, вразливості програмного забезпечення, SAST, DevSecOps.

Abstract

The paper investigates the application of generative artificial intelligence, specifically large language models, for automating source code security analysis. A comparative analysis of traditional static analysis tools and LLM-based approaches is conducted. The advantages and limitations of generative models in vulnerability identification, threat classification, and remediation recommendation generation are identified. A hybrid approach combining SAST tools with LLM-based analysis using RAG technology is proposed.

Keywords: code security analysis, generative artificial intelligence, large language models, static analysis, software vulnerabilities, SAST, DevSecOps.

Вступ

Забезпечення безпеки програмного забезпечення (ПЗ) є одним із ключових викликів сучасної інженерії програмних систем, що вимагає системного підходу до побудови захищених програмних продуктів [1]. Традиційні інструменти статичного аналізу безпеки (SAST) здатні виявляти типові патерни небезпечного коду відповідно до відомих класифікацій вразливостей [2], однак характеризуються значною кількістю хибнопозитивних спрацьовувань та обмеженим розумінням семантики програми. Розвиток великих мовних моделей (LLM) створює передумови для якісно нового підходу до аналізу безпеки, зокрема для автоматизованого виправлення вразливостей на основі генеративних моделей [3], а також їх застосування у задачах кібербезпеки [4].

Метою роботи є дослідження ефективності застосування генеративного ШІ для автоматизованого аналізу безпеки програмного коду, порівняння його можливостей із традиційними засобами SAST та розробка гібридного підходу, що поєднує переваги обох методів.

Аналіз традиційних підходів до статичного аналізу безпеки

Традиційні інструменти SAST, такі як SonarQube, Semgrep, Checkmarx та Fortify, функціонують на основі кількох базових механізмів: пошук за регулярними виразами, аналіз потоку даних, аналіз потоку керування та відповідність шаблонам [1]. Ці механізми забезпечують високу швидкість аналізу та можливість інтеграції в конвеєри CI/CD, однак мають суттєві обмеження при роботі зі складними програмними конструкціями. За різними оцінками, рівень хибнопозитивних спрацьовувань SAST-інструментів становить від 30% до 70% [2], що призводить до так званої «втоми від попереджень» (alert fatigue), коли розробники починають ігнорувати результати аналізу.

Аналіз безпеки ПЗ потребує підходу, що враховує не лише синтаксичні, але й семантичні характеристики коду [5]. Основними класами вразливостей, які ефективно виявляються SAST-інструментами, є: SQL-ін'єкції, для запобігання яким розроблено спеціалізовані системи [6]; міжсайтовий скриптинг (XSS); використання небезпечних функцій; жорстко закодовані облікові дані

[2]. Водночас складні вразливості, такі як порушення бізнес-логіки авторизації, небезпечна послідовність операцій та помилки в реалізації криптографічних протоколів, залишаються поза межами можливостей класичного SAST [3]. Методологічні підходи до безпечної розробки програмного забезпечення повинні бути адаптивними та охоплювати різні рівні аналізу [7].

Підхід до аналізу безпеки коду на основі LLM

Підхід на основі великих мовних моделей передбачає подання фрагментів вихідного коду до генеративної моделі (GPT-5.2 Codex, Claude Code, CodeLlama, StarCoder) разом із спеціально сформованим промптом, що описує завдання аналізу безпеки [3]. Промпт містить опис ролі моделі як експерта з безпеки, перелік класів вразливостей для пошуку відповідно до класифікації CWE [2], вимоги до формату відповіді та контекстну інформацію про технологічний стек проекту. Модель виконує ідентифікацію потенційних вразливостей із класифікацією за типами CWE, оцінює рівень критичності за шкалою CVSS та генерує конкретні рекомендації щодо виправлення, включаючи приклади безпечного коду. Можливості використання генеративних моделей у сфері кібербезпеки підтверджуються сучасними дослідженнями [4, 8].

Ключовою перевагою LLM є здатність аналізувати контекст міжфункціональних залежностей та виявляти вразливості, пов'язані з бізнес-логікою, що є недосяжним для класичних SAST-інструментів [3]. Наприклад, LLM може визначити, що функція авторизації не перевіряє рівень привілеїв користувача перед виконанням критичної операції, навіть якщо з точки зору синтаксису код є коректним. Крім того, LLM здатна генерувати людиночитабельні пояснення виявлених проблем, що значно спрощує процес їх усунення розробниками.

Для підвищення точності аналізу доцільно застосовувати технологію Retrieval-Augmented Generation (RAG), яка доповнює контекст LLM актуальною базою даних про відомі вразливості (CVE/NVD), специфічні патерни небезпечного коду для конкретної мови програмування та рекомендації з безпечного кодування (OWASP Cheat Sheets, CWE descriptions) [2, 5]. RAG-підхід дозволяє моделі посилатися на конкретні записи CVE при ідентифікації схожих патернів у коді, що підвищує довіру розробників до результатів аналізу.

Експериментальне дослідження та результати

Експериментальне дослідження проводилось на наборі з 15 тестових проектів мовами Python, JavaScript та Java із попередньо відомими вразливостями класів OWASP Top 10 [2]. Загальний обсяг тестової бази становив 847 файлів, що містили 124 підтверджені вразливості різних типів: SQL-ін'єкції (23 випадки) [6], XSS (19 випадків), небезпечна десеріалізація (12 випадків), порушення контролю доступу (18 випадків), використання компонентів з відомими вразливостями (15 випадків) та інші типи (37 випадків).

Порівняння проводилось між трьома підходами: класичний SAST-інструмент (Semgrep з набором правил p/security-audit), LLM-підхід (GPT-4 з оптимізованим промптом для аналізу безпеки) та запропонований гібридний підхід (Semgrep + LLM з RAG). Класичний SAST виявив 88 із 124 вразливостей (71,0%) із 46 хибнопозитивними спрацьовуваннями (рівень хибнопозитивних 34,3%) [1]. LLM-підхід виявив 108 вразливостей (87,1%) із 15 хибнопозитивними (12,2%) [3]. Гібридний підхід продемонстрував найкращі результати: 115 виявлених вразливостей (92,7%) із 9 хибнопозитивними (7,3%). Варто зазначити, що хибнонегативні спрацьовування (пропущені вразливості) є більш критичними, ніж хибнопозитивні, оскільки невиявлена вразливість може бути використана зловмисником. Рівень хибнонегативних становив: для SAST — 29,0% (36 пропущених вразливостей), для LLM — 12,9% (16 пропущених), для гібридного підходу — 7,3% (9 пропущених).

Суттєвою перевагою LLM-підходу виявилась здатність виявляти вразливості контролю доступу та бізнес-логіки: LLM ідентифікував 15 з 18 таких вразливостей, тоді як SAST — лише 4. Основним обмеженням LLM є значний час обробки (у середньому 12–18 секунд на файл порівняно з 0,1–0,3 секунди для SAST), залежність від якості промпту та обсягу аналізованого файлу й кодової бази. Також LLM інколи демонструє нестабільність результатів при повторному аналізі того самого коду, що потребує додаткових механізмів верифікації [9].

Гібридний підхід реалізовано за двоетапною архітектурою: на першому етапі SAST виконує швидке повне сканування кодової бази та формує перелік підозрілих фрагментів; на другому етапі LLM з RAG проводить поглиблений контекстний аналіз відмічених фрагментів та їх оточення, а

також додатково аналізує ділянки коду, пов'язані з бізнес-логікою та контролем доступу, які SAST не здатен охопити [5]. Саме завдяки цьому додатковому аналізу гібридний підхід досягає вищого рівня виявлення (92,7%), ніж кожен із методів окремо. Це також дозволяє зменшити загальний час аналізу порівняно із застосуванням лише LLM для всієї кодової бази, оскільки LLM обробляє не весь код повністю, а зосереджується на підозрілих фрагментах та критичних ділянках. Додатково LLM використовується для фільтрації хибнопозитивних результатів SAST, що дозволило зменшити їх кількість на 80,4%.

Висновки

Проведено порівняльний аналіз традиційних засобів статичного аналізу безпеки коду та підходу на основі великих мовних моделей. Встановлено, що генеративний ШІ забезпечує вищу точність виявлення вразливостей завдяки здатності враховувати семантичний контекст програми, проте потребує значних обчислювальних ресурсів. Запропоновано гібридний підхід, що поєднує переваги обох методів та є перспективним для практичного впровадження в процеси безпечної розробки ПЗ.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. McGraw G. Software Security: Building Security In / G. McGraw. – Boston : Addison-Wesley, 2006. – 448 p.

2. OWASP Foundation. OWASP Top Ten URL: https://owasp.org/www-project-top-ten.

3. Pearce H. Examining Zero-Shot Vulnerability Repair with Large Language Models / H. Pearce, B. Tan, B. Ahmad et al. // IEEE Symposium on Security and Privacy. – 2023. – P. 2339–2356.

4. Куперштейн Л. М. Про використання ChatGPT в кібербезпеці / Л. М. Куперштейн, Б. С. Примаков // Матеріали LII Науково-технічної конференції підрозділів ВНТУ, Вінниця. – 2023.

5. Zhou X. Large Language Model for Vulnerability Detection and Repair: Literature Review and the Road Ahead / X. Zhou, S. Cao, X. Sun, D. Lo // ACM Transactions on Software Engineering and Methodology. – 2024. – Vol. 34. – P. 1–31.

6. Войтович О. П. Система запобігання SQL-ін'єкціям / О. П. Войтович, О. Ю. Юрковецький, Л. М. Куперштейн // Міжнародна конференція Radio Electronics & Info Communications (REIC). – 2016.

7. Howard M. The Security Development Lifecycle / M. Howard, S. Lipner. – Redmond : Microsoft Press, 2006. – 320 p.

8. Sheng Z. LLMs in Software Security: A Survey of Vulnerability Detection Techniques and Insights / Z. Sheng, Z. Chen, S. Gu et al. // ACM Computing Surveys. – 2025. – Vol. 57, No. 7. – P. 1–33.

9. Ullah S. LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and Benchmarks / S. Ullah, M. Amin, A. Crego et al. // IEEE Symposium on Security and Privacy. – 2024.

Фіглярський Ілля Андрійович — студент групи 1БС-22б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: ilyafigliarskyi@gmail.com

Куперштейн Леонід Михайлович — канд. техн. наук, доцент кафедри захисту інформації, Вінницький національний технічний університет, м. Вінниця. e-mail: kupershtein@vntu.edu.ua

Fihliarskyi Illia — Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: ilyafigliarskyi@gmail.com

Kupershtein Leonid — Cand. Sc. (Eng.), Associate Professor of the Department of Information Protection, Vinnytsia National Technical University, Vinnytsia. E-mail: kupershtein@vntu.edu.ua