9. Browne, S. H., Bernstein, M., & Bickler, P. E. (2025). Evaluation of Pulse Oximetry Accuracy in a Commercial Smartphone and Smartwatch Device During Human Hypoxia Laboratory Testing // Sensors, 25(5), 1286. https://doi.org/10.3390/s25051286.

10. Alzueta, E., Gombert-Labedens, M., Javitz, H., Yuksel, D., Perez-Amparan, E., Camacho, L., Kiss, O., Zambotti, M., Sattari, N., Alejandro-Pena, A., Zhang, J., Shuster, A., Morehouse, A., Simon, K., Mednick, S., & Baker, F. C. (2024). Menstrual Cycle Variations in Wearable-detected Finger Temperature and Heart Rate, but not in Sleep Metrics, in Young and Midlife Individuals // Journal of Biological Rhythms, 39(5), 395–412. https://doi.org/10.1177/07487304241265018.

11. Muratyan, A., Cheung, W., Dibbo, S. V., & Vhaduri, S. (2021). Opportunistic Multi-Modal User Authentication for Health-Tracking IoT Wearables // arXiv preprint arXiv:2109.13705. https://doi.org/10.48550/arXiv.2109.13705.

12. Sancho, J., Alesanco, Á., & García, J. (2018). Biometric Authentication Using the PPG: A Long-Term Feasibility Study // Sensors, 18(5), 1525. https://doi.org/10.3390/s18051525.

13. Davoodi, M., Soker, A., Behar, J., & Yaniv, Y. (2023). Using Beat-to-Beat Heart Signals for Age-Independent Biometric Verification // Scientific Reports, 13(1). https://doi.org/10.1038/s41598-023-42841-4.

14. Ashtiyani, M., Iavasani, S. N., Alvar, A. A., & Deevband, M. R. (2018). Heart Rate Variability Classification Using Support Vector Machine and Genetic Algorithm // Journal of Biomedical Physics and Engineering, 8(4). https://doi.org/10.31661/jbpe.v0i0.614.

15. Wittenberg, T., Koch, R., Pfeiffer, N., & Eskofier, B. (2020). Evaluation of HRV Estimation Algorithms from PPG Data Using Neural Networks // Current Directions in Biomedical Engineering, 6(3), 505–509. https://doi.org/10.1515/cdbme-2020-3130.

16. Ding, J., Liu, Q., Ling, B. W.-K., & Li, W. (2026). Heart Rate Estimation Based on Joint Convolutional Neural Network and Conditional Generative Adversarial Network via Heart Rate Variabilities and Other Features Extracted from Photoplethysmograms // Biomedical Signal Processing and Control, 112(C), 108831. https://doi.org/10.1016/j.bspc.2025.108831.

17. Ólafsdóttir G. B., Larsson J. Deep Learning Approach for Extracting Heart Rate Variability from a Photoplethysmographic Signal // Bachelor Thesis, Kristianstad University, Sweden. 2020. Режим доступу: https://researchportal.hkr.se/ws/portalfiles/portal/35216086/FULLTEXT01.pdf.

Надійшла 30.10.2025

УДК 004.056.53:004.45:004.8
DOI: 10.31673/2409-7292.2025.041212

Максимович М.В.

ВИКОРИСТАННЯ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ НУЛЬОВОГО ДНЯ

Стрімка цифровізація та широко поширене використання відкритого програмного забезпечення істотно підвищують ризики появи нових загроз у ланцюгах постачання ПЗ. Особливо небезпечними є вразливості нульового дня, які можуть непомітно потрапляти у проєкти через сторонні бібліотеки та залишатися невиявленими до моменту їх фактичної експлуатації. У таких умовах актуальним є пошук підходів, здатних забезпечити проактивний аналіз залежностей та своєчасне виявлення потенційно небезпечних компонентів. Метою статті є оцінка можливостей застосування методів штучного інтелекту для ідентифікації вразливих сторонніх бібліотек та аналіз ефективності такого підходу на основі моделювання реального інциденту компрометації сторонньої бібліотеки. У роботі описано характерні властивості вразливостей нульового дня, а також підкреслено обмеження традиційних інструментів забезпечення безпеки, які не завжди здатні оперативно реагувати на нові загрози або враховувати транзитивні залежності. Проведений експеримент передбачав аналіз десяти тестових проєктів із використанням агентного підходу на основі різних моделей штучного інтелекту, що дозволило оцінити їхню здатність виявляти скомпрометовані компоненти, формувати структуровані звіти та надавати рекомендації щодо мінімізації ризиків. Отримані результати підтвердили точність підходу та його потенціал для інтеграції в сучасні процеси розробки та супроводу програмного забезпечення. Водночас підкреслено недетермінованість роботи мовних моделей, що зумовлює необхідність додаткової перевірки отриманих результатів. Проведене дослідження окреслює подальші перспективи розвитку, зокрема вдосконалення механізмів проактивного моніторингу, а також розробку методів верифікації результатів, отриманих із використанням методів штучного інтелекту, що сприятиме підвищенню достовірності та надійності аналітичних висновків.

Ключові слова: штучний інтелект, вразливості нульового дня, безпека ПЗ, автоматизація, відкриті бібліотеки, програмне забезпечення.

Вступ

Зростаюча інтеграція технологій у всі сфери повсякденного життя суттєво підвищує значення кібербезпеки. Для забезпечення надійного та сталого функціонування технологічні екосистеми потребують ефективного захисту від кіберзагроз [1], саме тому кібербезпека є однією з ключових умов надійності та успішності будь-якого сучасного програмного продукту. Відкрите програмне забезпечення стало основою розробки сучасних програмних систем, забезпечуючи функціонування широкого спектра застосунків, інфраструктурних компонентів і сервісів. Його прозорість, модульна архітектура та модель спільної розробки сприяли швидким інноваціям і зниженню вартості розробки в індустрії. Водночас поширені проблеми безпеки у відкритому коді підсилюють занепокоєння щодо захищеності ланцюга постачання програмного забезпечення [2]. Особливу небезпеку становлять вразливості нульового дня, які залишаються невідомими до моменту їх експлуатації зловмисниками. Попри значний прогрес у сфері автоматизованого тестування безпеки, сучасні інструменти переважно орієнтовані на виявлення вже відомих вразливостей, зафіксованих у публічних базах даних або сигнатурних каталогах. Такий підхід обмежує можливість своєчасного реагування на нові або потенційні загрози, що можуть виникати внаслідок використання сторонніх бібліотек.

Відсутність гнучких механізмів раннього виявлення вразливих компонентів у програмних проєктах підвищує ризик інтеграції небезпечних залежностей, що ускладнює подальше усунення наслідків і збільшує витрати на забезпечення безпеки.

Аналіз літератури

Вразливості нульового дня є невідомими розробникам на момент їх експлуатації, що надає зловмисникам можливість здійснювати атаки до впровадження будь-яких заходів захисту. Такі вразливості становлять особливу небезпеку для електронної комерції, оскільки можуть призводити до фінансових втрат, порушення логістичних процесів та зниження довіри користувачів [3].

Як зазначається у статті [4], традиційні методи виявлення zero-day атак включають сигнатурний аналіз, статистичне та поведінкове виявлення, графові моделі та перевірку цілісності компонентів, проте автори у статті [5] вважають, що жоден із сучасних інструментів - статичних, сигнатурних чи поведінкових - не здатен повністю забезпечити безпеку ланцюга постачання програмного забезпечення. Навіть багаторівнева стратегія захисту має обмеження у виявленні нових загроз, тому подальші дослідження в контексті впровадження інноваційних методів на основі штучного інтелекту є надзвичайно актуальними.

Часто вразливості нульового дня можуть бути додані до проєкту через сторонні бібліотеки. Для виявлення таких компрометованих компонентів застосовується метод Software Composition Analysis (SCA), який дозволяє ефективно і автоматизовано перевіряти залежності, що використовуються у проєкті, на наявність відомих вразливостей, тим самим забезпечуючи захист від атак через ланцюг постачання ще до етапу збірки програмного забезпечення [6].

Проте в сучасних SCA рішеннях є багато недоліків, як наприклад у статті [7] автори підкреслюють, що більшість SCA-рішень ефективні лише за ідеальних умов, тоді як у реальних сценаріях вони часто виявляють значні прогалини у виявленні вразливостей і не здатні протистояти еволюційним загрозам. Крім того, у статті [8] зазначається, що ефективність інструментів SCA безпосередньо залежить від актуальності та повноти їхніх баз даних вразливостей.

Автори підкреслюють, що неточні або застарілі записи можуть призводити до пропусків відомих вразливостей, тому для підвищення надійності аналізу рекомендується використовувати кілька інструментів одночасно, щоб мінімізувати ризик хибних спрацювань. Також, інструменти SCA не завжди коректно обробляють складні, транзитивні або неявно оголошені залежності, що знижує точність виявлення [9].

© Максимович М.В. (2025) Використання методів штучного інтелекту для виявлення вразливостей нульового дня. Сучасний захист інформації, 4(64), 123–132.
https://doi.org/10.31673/2409-7292.2025.041212

Сучасні системи на основі методів штучного інтелекту можуть значно покращити, якщо не повністю замінити стандартні методи автоматизованого забезпечення безпеки. Як зазначається у [10], розвиток технологій штучного інтелекту та аналітики великих даних створює нові можливості для підвищення ефективності кіберзахисту, зокрема у прогнозуванні потенційних вразливостей, виявленні аномальної активності та автоматизованому реагуванні на загрози в режимі реального часу. Також, у статті [11] описується, що використання методів штучного інтелекту відкриває нові можливості для підвищення точності та повноти аналізу програмного забезпечення. Інтелектуальні моделі здатні виявляти приховані закономірності та встановлювати семантичні зв'язки між компонентами коду, що значно розширює можливості традиційних підходів і підвищує ефективність виявлення вразливостей та компрометованих бібліотек у складних програмних системах.

Таким чином, можна зробити висновок, що зі стрімким розвитком методів штучного інтелекту з'являються нові підходи до забезпечення безпеки програмного забезпечення, які здатні суттєво підвищити ефективність традиційних інструментів або навіть частково їх замінити. Це зумовлює актуальність подальших досліджень у сфері застосування технологій штучного інтелекту для підвищення рівня безпеки програмних систем та вдосконалення процесів виявлення і запобігання кіберзагрозам.

Метою даної статті є аналіз можливостей застосування методів на основі штучного інтелекту для виявлення вразливостей нульового дня, які можуть потрапляти до програмного проєкту через інтеграцію сторонніх бібліотек. Серед основних завдань дослідження - оцінка обмежень традиційних інструментів забезпечення безпеки та експериментальна перевірка ефективності ШІ-підходу для ідентифікації скомпрометованих компонентів і формування звітності та рекомендацій на основі аналізу.

Основна частина

На сьогоднішній день в світі існує безліч вразливостей, які в тій чи іншій формі несуть ризики цифровим продуктам. Найбільш поширені вразливості добре описані компанією OAWSP, в документі OWASP Top-10 [12], та крім них також постійно появляються нові вразливості. Дуже небезпечними є вразливості нульового дня (zero day vulnerabilities) - це дефекти безпеки, про існування яких розробник програмного забезпечення та спільнота ще не поінформовані. Такі вразливості часто експлуатуються до офіційного розкриття або випуску оновлення, що істотно ускладнює їх виявлення традиційними засобами моніторингу. Ризик їхнього впливу посилюється в умовах ланцюгів постачання ПЗ та широкого використання сторонніх бібліотек, де приховані зміни можуть швидко масштабуватися через транзитивні залежності.

Як приклад такої вразливості можна навести інцидент, який мав місце у вересні, 2025 року [13]. Тоді з'явилася інформація, що одна з найпопулярніших бібліотек на основі мови програмування Javascript - debug, яка щотижня завантажується понад 100 мільйонів разів і входить до складу тисяч інших проєктів, - була скомпрометована: версія 4.4.2 містила шкідливий код, здатний викрадати криптовалютні ключі та токени доступу.

Після публікації цього попередження спільнота швидко виявила, що зараження не обмежується лише бібліотекою debug: низка інших популярних бібліотек - зокрема chalk, ansi-styles, supports-color та їхні похідні - також стали вразливими через транзитивні залежності, тобто були скомпрометовані опосередковано, бо у своїх ланцюгах імпорту використовували вразливу версію debug [14]. Це показало, наскільки небезпечною може бути компрометація одного центрального пакета у відкритій екосистемі NPM, коли вразливість миттєво поширюється на тисячі застосунків через ланцюг постачання програмного забезпечення.

Одним із методів, який можна застосовувати для боротьби з вразливими сторонніми бібліотеками, є аналіз складу програмного забезпечення (Software Composition Analysis, SCA). SCA - це технологія, що дозволяє автоматично ідентифікувати всі зовнішні компоненти, які використовуються в програмному продукті, зокрема їхні версії, джерела походження, ліцензії

© Максимович М.В. (2025) Використання методів штучного інтелекту для виявлення вразливостей нульового дня. Сучасний захист інформації, 4(64), 123–132.
https://doi.org/10.31673/2409-7292.2025.041212

та відомі вразливості. Під час аналізу інструмент сканує файли конфігурацій (наприклад, package.json, pom.xml, requirements.txt або yarn.lock) і формує детальну карту залежностей, включно з транзитивними (вкладеними) бібліотеками.

Отриманий перелік звіряється з базами даних вразливостей, що дозволяє визначити, чи містить проєкт компоненти з відомими CVE-записами. Тим не менш, SCA може мати ряд недоліків, що проявилися під час розслідування інциденту з компрометацією NPM-бібліотек. По-перше, такі інструменти покладаються на офіційні бази CVE, тому не здатні оперативно виявити нові, ще не задокументовані zero-day вразливості. По-друге, вони часто працюють лише з основними залежностями, не аналізуючи транзитивні ланцюги, через що компрометація глибоко вкладених пакетів, як-от debug у складі chalk чи ansi-styles, може залишитися непоміченою. По-третє, результати SCA залежать від точності метаданих у репозиторіях, тоді як у цьому інциденті шкідливий код був опублікований під тим самим номером версії, що ускладнило автоматичне виявлення атаки.

Розвиток штучного інтелекту суттєво трансформує підходи до забезпечення кібербезпеки, роблячи процеси виявлення та реагування на загрози швидшими й динамічнішими. Алгоритми машинного навчання здатні аналізувати великі обсяги даних - від коду бібліотек до метаданих публікацій - і виявляти нетипові шаблони, що свідчать про потенційні атаки на ланцюг постачання програмного забезпечення. Особливу роль у цьому відіграють агентні системи ШІ, які можуть автономно взаємодіяти з різними джерелами даних, обмінюватися результатами аналізу, перевіряти гіпотези про вразливості та формувати узгоджені звіти про ризики в реальному часі. Такий підхід може забезпечувати проактивну оборону навіть проти ще невідомих zero-day загроз.

Сучасні AI-рішення можуть моніторити безпекові форуми та технічні спільноти, де інформація про вразливості з'являється раніше, ніж у базах CVE. До таких джерел належать Reddit (r/netsec, r/cybersecurity), Exploit Database, BleepingComputer Forums, HackerOne, BugTraq, 0day.today, а також обговорення на GitHub і Stack Overflow. Моделі обробки природної мови (NLP) здатні у реальному часі відстежувати повідомлення з цих ресурсів, виявляти згадки про нові бібліотеки або підозрілі пакети та автоматично порівнювати ці дані зі складом конкретного проєкту.

У контексті інциденту з компрометацією бібліотек NPM (наприклад, debug@4.4.2, chalk, ansi-styles) така система могла б використати дані з GitHub-обговорень, ще до появи офіційних повідомлень, сформувати звіт про використання вразливих компонентів і проактивно попередити ключових розробників про загрозу.

Подібний аналіз може бути реалізований як інфраструктурний елемент, що автоматично спрацьовує за тригером - появою нової публікації, згадки про пакет або за розкладом, - забезпечуючи безперервний моніторинг та виявлення потенційних ризиків (рис. 1).

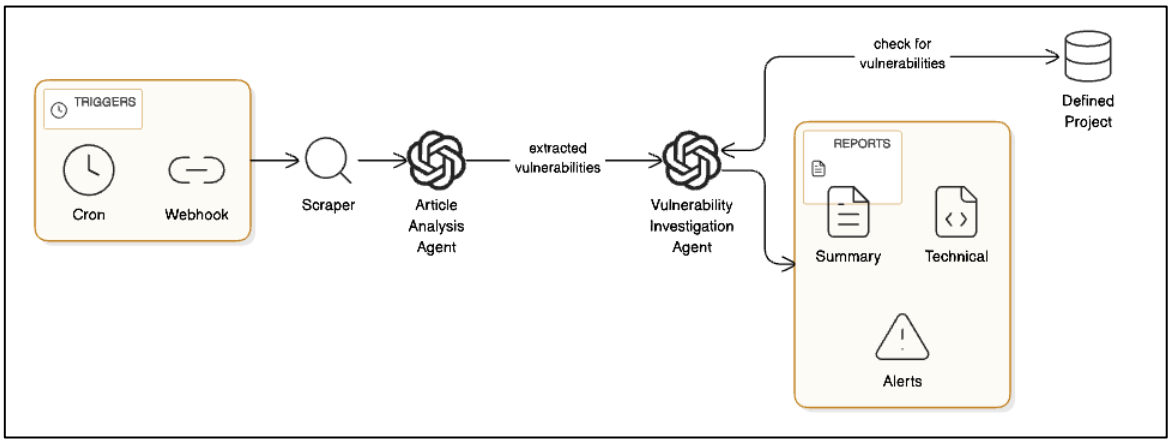

Рис. 1. Агентна система для аналізу та виявлення вразливих пакетів на основі даних з відкритих джерел

© Максимович М.В. (2025) Використання методів штучного інтелекту для виявлення вразливостей нульового дня. Сучасний захист інформації, 4(64), 123–132. https://doi.org/10.31673/2409-7292.2025.041212

Таким чином, враховуючи наведено вище інформацію, можна зробити висновок, шо сучасний розвиток методів штучного інтелекту відкриває новий горизонт потенційних вдосконалень методів автоматизованих виявлень вразливостей, а дослідження в цій галузі здобувають пріоритет та стають актуальним як ніколи.

Методика дослідження

Для аналізу ефективності підходу, описаного в попередньому розділі, було використано симуляцію описаного вище інциденту. В рамках експерименту налаштовано 10 різних проєктів на базі мови програмування Javascript, кожен з яких мав свої залежності від сторонніх бібліотек.

В кожен з проєктів було додано вразливі бібліотеки, які згадувались в інциденті, також були додані версії даних бібліотек без вразливостей.

Було проведено 3 досліди, для кожного з яких було змінено порядок включення вразливих бібліотек у проєкти. Кінцевий набір проєктів разом з їхніми вразливостями наведено в таблиці 1, де напівжирним шрифтом виділено бібліотеки з вразливостями, а нежирним – невразливі версії бібліотек.

Таблиця 1

Налаштування тестових проєктів з відомими вразливостями

Назва Проєкту	**Сценарій A**	**Сценарій B**	**Сценарій C**
api-gateway	**debug@4.4.2**	debug@4.4.1	–
user-dashboard	**chalk@5.6.1** **ansi-styles@6.2.2**	–	–
data-processor	**supports-color@10.2.1** **strip-ansi@7.1.1** **wrap-ansi@9.0.1**	–	supports-hyperlinks@4.1.0 has-ansi@6.0.0
notification-service	debug@4.4.1 ansi-regex@6.2.0	–	**debug@4.4.2** **ansi-regex@6.2.1** **strip-ansi@7.1.1**
analytics-engine	chalk@5.6.0 color-convert@3.1.0	–	–
auth-service	–	**debug@4.4.2** **color-name@2.0.1** **is-arrayish@0.3.3**	–
file-uploader	–	slice-ansi@7.1.0 color@5.0.0	**wrap-ansi@9.0.1** **chalk-template@1.1.1** **backslash@0.2.1**
payment-processor	–	**chalk@5.6.1** **ansi-styles@6.2.2**	–
logging-service	–	**supports-color@10.2.1** **color-string@2.1.1** **simple-swizzle@0.2.3**	ansi-styles@6.2.1 simple-swizzle@0.2.2
config-manager	–	–	**chalk@5.6.1** **color-convert@3.1.1**

В рамках кожного досліду було зроблено 3 спроби виявлення вразливостей у кожному з проєктів. В якості основних метрик аналізу було використано наступні.

T_{gen} (Time generation) (1) - час генерації звіту:

$$T_{gen} = T_{end} - T_{start}, \quad (1)$$

де T_{start} - початок виконання аналізу, а T_{end} - кінець виконання аналізу.

TPR (True positive rate) (2) - частка вразливостей, які були вірно ідентифіковані інструментом.

$$TPR = TP / (TP + FN), \quad (2)$$

де TP (true positive) – кількість правильно виявлених вразливостей, а FN (false negative) – кількість невиявлених вразливостей.

P (Precision) (3) - відношення кількості правильно ідентифікованих вразливостей до всіх виявлених.

$$P = TP / (TP + FP) , \quad (3)$$

де FP (false positives) – кількість помилкових тривог для вразливостей, яких насправді немає в коді.

У межах проведеного експерименту як інтелектуального агента було використано середовище Cursor, що забезпечує інтеграцію алгоритмів штучного інтелекту для аналізу та автоматизації процесів розробки. Для налаштування логіки поведінки агента було використано наступну вхідну інструкцію (prompt):

```
You are a Software Security Assessment Manager.
You are provided with a list of vulnerable libraries.
Your task is to assess the given software projects to determine whether they
include any of these vulnerable components.
For each detected vulnerability, you must:
Analyze the vulnerability details, including affected libraries, versions,
and potential impact.
Evaluate the risk level and provide recommendations for mitigation.
Generate a report summarizing all identified vulnerabilities and potential
risks.
```

У межах експерименту для кожного зі сценаріїв було застосовано окрему модель штучного інтелекту:

- Composer-1 від Cursor - для Сценарію A,
- Sonnet-4.5 (Claude) від Anthropic - для Сценарію B
- GPT-5 від OpenAI - для Сценарію C.

Такий підхід забезпечив можливість порівняльного аналізу ефективності різних мовних моделей у межах аналогічних завдань з виявлення вразливостей та формування звітів.

Результати дослідження

В результаті проведеного дослідження встановлено, що запропонований метод виявлення вразливих бібліотек на основі застосування методів штучного інтелекту має значний практичний потенціал. На основі аналізу, для кожного сценарію, беручи за основу реальне повідомлення про вразливі бібліотеки, був згенерований детальний звіт, який включав наступне:

- Контекст проблеми, з повним списком вразливих бібліотек (рис. 2).

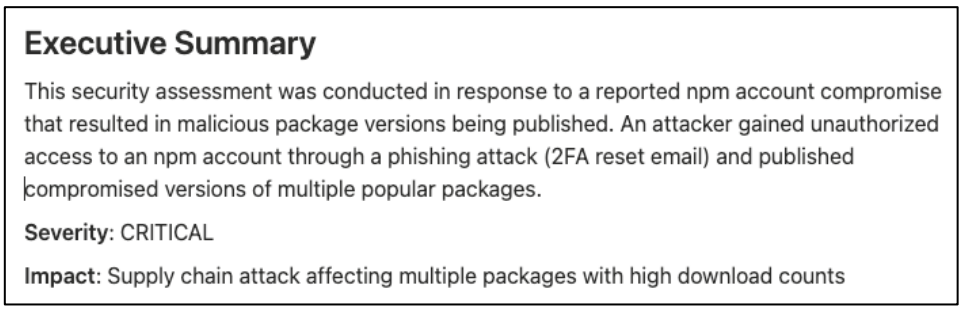

Рис. 2. Фрагмент зі звіту з коротким підсумком

- Детальний опис потенційної вразливості.
- Приклад звіту про наявність вразливих версій бібліотек в проєктах (рис. 3).

Summary Table

Project Name	Status	Vulnerable Packages	Action Required
analytics-engine	✅ SAFE	None	None
api-gateway	⚠️ VULNERABLE	debug@4.4.2	Immediate update required
auth-service	✅ SAFE	None	None
config-manager	✅ SAFE	None	None
data-processor	⚠️ VULNERABLE	strip-ansi@7.1.1, supports-color@10.2.1, wrap-ansi@9.0.1	Immediate update required
file-uploader	✅ SAFE	None	None
logging-service	✅ SAFE	None	None
notification-service	✅ SAFE	None	None
payment-processor	✅ SAFE	None	None
user-dashboard	⚠️ VULNERABLE	chalk@5.6.1, ansi-styles@6.2.2	Immediate update required

Рис. 3. Фрагмент зі звіту з описом вразливих проєктів

- Інформацію про наявність версій бібліотек, наближених до вразливих (рис. 4).

1. analytics-engine

- **Status:** ✅ SAFE
- **Analysis:** Uses `chalk@5.6.0` and `color-convert@3.1.0` - different versions from compromised ones
- **Note:** Consider updating to the latest versions for security best practices

Рис. 4. Фрагмент зі звіту про наявність бібліотек з версіями, наближеними до вразливих

- Оцінку ризиків та рекомендовані невідкладні заходи (рис. 5).

Immediate Actions (Critical - Within 24 hours)

1. **Update Vulnerable Packages**
   - **api-gateway**: Update `debug` package
   - **data-processor**: Update `strip-ansi`, `supports-color`, and `wrap-ansi` packages
   - **user-dashboard**: Update `chalk` and `ansi-styles` packages

Рис. 5. Фрагмент зі звіту про оцінку ризиків та рекомендаціями

Таблиця 2

Налаштування тестових проєктів з відомими вразливостями

Назва Проєкту	**Спроба 1**	**Спроба 2**	**Спроба 3**
api-gateway	debug@4.4.2	debug@4.4.2	debug@4.4.2
user-dashboard	chalk@5.6.1, ansi-styles@6.2.2	chalk@5.6.1, ansi-styles@6.2.2	chalk@5.6.1, ansi-styles@6.2.2
data-processor	supports-color@10.2.1, strip-ansi@7.1.1, wrap-ansi@9.0.1	supports-color@10.2.1, strip-ansi@7.1.1, wrap-ansi@9.0.1	supports-color@10.2.1, strip-ansi@7.1.1, wrap-ansi@9.0.1

В усіх експериментальних сценаріях запропонований метод продемонстрував 100% повноту виявлення (TPR=1,0) та 100% точність визначення вразливостей (Precision=1,0) щодо

© Максимович М.В. (2025) Використання методів штучного інтелекту для виявлення вразливостей нульового дня. Сучасний захист інформації, 4(64), 123–132. https://doi.org/10.31673/2409-7292.2025.041212

скомпрометованих бібліотек. Це свідчить про високу ефективність підходу та дає підстави розглядати його для подальшого впровадження в автоматизовані системи забезпечення безпеки програмного забезпечення. Результати проведених тестів наведені у таблицях 2,3,4.

Таблиця 3

Налаштування тестових проєктів з відомими вразливостями

Назва Проєкту	**Спроба 1**	**Спроба 2**	**Спроба 3**
auth-service	debug@4.4.2, color-name@2.0.1, is-arrayish@0.3.3	debug@4.4.2, color-name@2.0.1, is-arrayish@0.3.3	debug@4.4.2, color-name@2.0.1, is-arrayish@0.3.3
payment-processor	chalk@5.6.1, ansi-styles@6.2.2	chalk@5.6.1, ansi-styles@6.2.2	chalk@5.6.1, ansi-styles@6.2.2
logging-service	supports-color@10.2.1, color-string@2.1.1, simple-swizzle@0.2.3	supports-color@10.2.1, color-string@2.1.1, simple-swizzle@0.2.3	supports-color@10.2.1, color-string@2.1.1, simple-swizzle@0.2.3

Таблиця 4

Налаштування тестових проєктів з відомими вразливостями

Назва Проєкту	**Спроба 1**	**Спроба 2**	**Спроба 3**
notification-service	debug@4.4.2, ansi-regex@6.2.1, strip-ansi@7.1.1	debug@4.4.2, ansi-regex@6.2.1, strip-ansi@7.1.1	debug@4.4.2, ansi-regex@6.2.1, strip-ansi@7.1.1
file-uploader	wrap-ansi@9.0.1, chalk-template@1.1.1, backslash@0.2.1	wrap-ansi@9.0.1, chalk-template@1.1.1, backslash@0.2.1	wrap-ansi@9.0.1, chalk-template@1.1.1, backslash@0.2.1
config-manager	chalk@5.6.1, color-convert@3.1.1	chalk@5.6.1, color-convert@3.1.1	chalk@5.6.1, color-convert@3.1.1

У межах експериментального дослідження, середній час виконання усіх трьох сценаріїв становив **28,78 секунди**, що свідчить про стабільну продуктивність реалізованого методу аналізу вразливостей. Результат виконання кожного експерименту для кожної моделі зображений в таблиці 5.

Таблиця 5

Час виконання аналізу для кожного експерименту

Експеримент	**Модель**	**Спроба 1**	**Спроба 2**	**Спроба 3**
Експеримент А	Composer-1	12 секунд	13 секунд	10 секунд
Експеримент В	Claude-4.5	50 секунд	17 секунд	20 секунд
Експеримент С	GPT-5	50 секунд	45 секунд	42 секунди

- *Експеримент А (Composer-1, Cursor)* продемонстрував найвищу швидкодію та стабільність, із середнім часом виконання 11,67 с і незначними відхиленнями (10-13 с). Це вказує на ефективну інтеграцію моделі у процес аналізу.
- *Експеримент В (Sonnet-4.5, Anthropic)* мав середній час 29,00 с, із помірною варіативністю (17-50 с), що може бути пов'язано з адаптивною поведінкою моделі при обробці запитів різної складності.
- *Експеримент С (GPT-5, OpenAI)* показав найдовший середній час виконання - 45,67 с, однак із високою стабільністю результатів (42-50 с), що свідчить про більшу обчислювальну складність і ширшу контекстну обробку під час генерації звітів.

Такі результати свідчать, що вибір оптимальної LLM може істотно вплинути на швидкодію системи автоматизованого аналізу вразливостей. Незважаючи на те, що кожен експеримент продемонстрував 100% точність виявлення вразливостей, та протягом експерименту не вдалося отримати результату з хибними висновками, не можна гарантувати

© Максимович М.В. (2025) Використання методів штучного інтелекту для виявлення вразливостей нульового дня. Сучасний захист інформації, 4(64), 123–132.
https://doi.org/10.31673/2409-7292.2025.041212

аналогічну ефективність для всіх наступних аналізів. Це пов'язано з тим, що методи на основі штучного інтелекту є недетермінованими, тобто результати їх роботи можуть бути різними при кожній ітерації. Більше того, через явище галюцинацій моделей або інші технічні фактори можливе формування хибних висновків чи помилкових класифікацій, що потребує додаткової перевірки результатів. Таким чином, можна зробити висновок, що даний метод має великі перспективи для практичного застосування, проте залишаються необхідними подальші дослідження в області визначення підходів до верифікації результатів аналізу.

Висновки і рекомендації

Проведене дослідження підтвердило ефективність запропонованої методики виявлення вразливих сторонніх бібліотек із використанням методів штучного інтелекту. Розроблений підхід продемонстрував високу ефективність у всіх експериментальних сценаріях, що засвідчує здатність підходу коректно ідентифікувати вразливі компоненти. Отримані результати свідчать, що метод має значний потенціал як допоміжний інструмент до існуючих методів автоматизованого аналізу безпеки, зокрема систем SCA (Software Composition Analysis), які здійснюють аналіз залежностей та контроль використання сторонніх бібліотек.

Дослідження було зосереджене на виявленні вразливих бібліотек відповідно до відомих даних, однак запропонований підхід має потенціал для розширення у напрямі виявлення безпосередньо нових вразливостей чи векторів атак, а не лише ідентифікації відомих бібліотек із ризиками. Це відкриває можливість його застосування як доповнення до методів динамічного (DAST), статичного (SAST) та інтерактивного (IAST) тестування безпеки, що дозволить формувати проактивні системи виявлення та попередження загроз на основі аналітичних можливостей великих мовних моделей та агентних систем. Серед недоліків підходу залишається його недетермінований характер, який зумовлює можливі варіації результатів через ефект галюцинацій або зміну контексту генерації. Це підкреслює необхідність додаткових досліджень у сфері верифікації результатів та розроблення методів перевірки достовірності аналітичних висновків. Подальші дослідження також доцільно спрямувати на інтеграцію запропонованої методики у життєвий цикл розробки програмного забезпечення. Така інтеграція дозволить забезпечити безперервний моніторинг компонентів, раннє виявлення ризиків і автоматизоване реагування в межах процесів DevSecOps, що сприятиме підвищенню надійності, стійкості та проактивності систем безпеки сучасного програмного забезпечення.

Перелік посилань

1. Systematic Review of Current Approaches and Innovative Solutions for Combating Zero-Day Vulnerabilities and Zero-Day Attacks. (n.d.). Retrieved November 9, 2025, from https://ieeexplore.ieee.org/document/11028033.

2. Anasuri, S. (2023). Secure Software Supply Chains in Open-Source Ecosystems. International Journal of Emerging Trends in Computer Science and Information Technology, 4(1), 62–74. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I1P108.

3. Gunasekara, A. (2023). AI-Driven Big Data Analytics for Transforming Cybersecurity for Zero-Day Vulnerabilities in E-Commerce Supply Chains. Journal of Advances in Cybersecurity Science, Threat Intelligence, and Countermeasures, 7(12), 17–31.

4. Kumar, V., & Sinha, D. (2021). A robust intelligent zero-day cyber-attack detection technique. Complex & Intelligent Systems, 7, 2211–2234. https://doi.org/10.1007/s40747-021-00396-9.

5. (PDF) A Comprehensive Review of Open-Source Malware Scanners in the Software Supply Chain. (2025, March 24). ResearchGate. https://www.researchgate.net/publication/395471396_A_Comprehensive_Review_of_Open-Source_Malware_Scanners_in_the_Software_Supply_Chain.

6. Pre-Build OSS Compliance: Automated Dependency, License, and CVE Detection. (n.d.). Retrieved November 9, 2025, from https://ieeexplore.ieee.org/abstract/document/11136967.

7. Shu, C., Chen, W., Fan, G., Yu, H., Huang, Z., & Liang, Y. (2025). Tool or Toy: Are SCA tools ready for challenging scenarios? Computers & Security, 158, 104624. https://doi.org/10.1016/j.cose.2025.104624.

8. Imtiaz, N., Thorne, S., & Williams, L. (2021, August 27). A Comparative Study of Vulnerability Reporting by Software Composition Analysis Tools. arXiv.Org. https://doi.org/10.1145/3475716.3475769.

9. Software Composition Analysis for Vulnerability Detection: An Empirical Study on Java Projects | Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (world). (n.d.). ACM Conferences. https://doi.org/10.1145/3611643.3616299.

© Максимович М.В. (2025) Використання методів штучного інтелекту для виявлення вразливостей нульового дня. Сучасний захист інформації, 4(64), 123–132. https://doi.org/10.31673/2409-7292.2025.041212

10. Gunasekara, A. (2023). AI-Driven Big Data Analytics for Transforming Cybersecurity for Zero-Day Vulnerabilities in E-Commerce Supply Chains. Journal of Advances in Cybersecurity Science, Threat Intelligence, and Countermeasures, 7(12), 17–31.

11. Jiang, L., An, J., Huang, H., Tang, Q., Nie, S., Wu, S., & Zhang, Y. (2024, January 20). BinaryAI: Binary Software Composition Analysis via Intelligent Binary Source Code Matching. arXiv.Org. https://arxiv.org/abs/2401.11161v3.

12. Sane, P. (2020). Is the OWASP Top 10 List Comprehensive Enough for Writing Secure Code? 58–61. Scopus. https://doi.org/10.1145/3437075.3437089.

13. GHSA-8mgj-vmr8-frr6—GitHub Advisory Database. (n.d.). GitHub. Retrieved November 17, 2025, from https://github.com/advisories/GHSA-8mgj-vmr8-frr6.

14. debug-js. (n.d.). (RESOLVED) Version 4.4.2 published to npm is compromised · Issue #1005 · debug-js/debug. GitHub. Retrieved November 17, 2025, from https://github.com/debug-js/debug/issues/1005.

Надійшла 05.11.2025

УДК 004.056:004.4’2
DOI: 10.31673/2409-7292.2025.041213

Мичуда Л.З., Расяк В.І.

ОГЛЯД СУЧАСНИХ ПІДХОДІВ ДО БЕЗПЕЧНОГО КОНФІГУРУВАННЯ DOCKER COMPOSE

У статті проведено аналітичний огляд сучасних підходів до забезпечення безпеки контейнерних конфігурацій, зокрема у контексті використання Docker Compose як інструменту для декларативного опису багатоконтейнерних систем. Визначено ключові проблеми, пов’язані з конфігураційними помилками у файлах docker-compose.yml, які можуть призводити до порушення конфіденційності, цілісності та доступності контейнеризованих сервісів. Проаналізовано можливості та обмеження існуючих інструментів перевірки безпеки: Docker Bench for Security, OpenSCAP, Trivy, Checkov, KICS та інших, з точки зору їхньої ефективності щодо аналізу конфігураційного рівня. Особливу увагу приділено питанням відповідності конфігурацій міжнародним стандартам та рекомендаціям безпеки, зокрема CIS Docker Benchmark і практикам DevSecOps.

У результаті дослідження визначено основні типи ризиків, характерні для середовищ Docker Compose, та запропоновано концептуальну класифікацію таких ризиків за категоріями конфіденційності, цілісності, доступності, ізоляції та відповідності стандартам. Обґрунтовано необхідність створення уніфікованих політик безпечного конфігурування Docker Compose і розроблення автоматизованих інструментів перевірки, інтегрованих у процеси CI/CD. Отримані результати формують теоретичну основу для подальших досліджень у сфері забезпечення безпеки контейнерних конфігурацій та розроблення практичних засобів їх аудиту.

Ключові слова: контейнеризація, Docker Compose, безпека конфігурацій, DevSecOps, Docker Bench for Security, OpenSCAP, контейнерні середовища.

Вступ

У сучасних інформаційних системах контейнеризація є ключовою технологією, що забезпечує ефективність, портативність і масштабованість розгортання програмних застосунків. Серед існуючих інструментів екосистеми контейнеризації платформа Docker посідає провідне місце, а її компонент Docker Compose відіграє важливу роль у спрощенні процесу конфігурації багатоконтейнерних систем. Формат опису у вигляді файлу docker-compose.yml дозволяє декларативно визначати взаємозв’язки між сервісами, параметри мережі, обмеження ресурсів і змінні середовища, що істотно полегшує процес розгортання та супроводу застосунків у середовищах DevOps [10, 17].

Водночас із розширенням масштабів використання контейнеризації спостерігається зростання кількості інцидентів, пов’язаних із конфігураційними помилками. Неправильне визначення привілеїв контейнерів, некоректне налаштування мережевих портів, зберігання секретів у відкритому вигляді чи відсутність обмежень ресурсів можуть призвести до серйозних загроз інформаційній безпеці. Такі недоліки не завжди є результатом навмисних дій розробників, а часто зумовлені відсутністю чітких методичних рекомендацій або автоматизованих засобів перевірки конфігурацій на рівні Docker Compose.