https://doi.org/10.31891/2307-5732-2025-353-29
УДК 004.4

ТАЗЕТДІНОВ ОЛЕКСІЙ
Черкаський державний технологічний університет
https://orcid.org/0000-0003-4387-0500
e-mail: o.v.tazetdinov.asp22@chdtu.edu.ua
ТАЗЕТДІНОВ ВАЛЕРІЙ
Черкаський державний технологічний університет
https://orcid.org/0000-0002-1091-9075
e-mail: valeriy.tazetdinov@gmail.com

ЗАСТОСУВАННЯ МАШИННОГО НАВЧАННЯ ДЛЯ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

В роботі проаналізовано різні моделі навчання нейронних мереж призначених для класифікації рівнів вразливостей програмного коду.

Ключові слова: нейронні мережі, вразливості програмного коду, моделі нейронних мереж, навчання нейронних мереж.

TAZETDINOV OLEKSII
TAZETDINOV VALERIY
Cherkassy State Technological University

USING MACHINE LEARNING TO DETECT SOFTWARE VULNERABILITIES

The increasing complexity of current software systems has led to the growth of the number of security vulnerabilities that pose a threat to both the single user and large organizations. Conventional vulnerability detection approaches include manual code reviews and rule-based static analysis and are limited by the large amount of code and the dynamic nature of security risks. As a result, machine learning and, specifically, neural networks have become popular for the automation of the software vulnerability detection process.

Neural networks can discover complex patterns in big data sets, and thus are suitable for identifying potential security weaknesses. Numerous models and training strategies have been suggested to enhance their precision and speed in vulnerability detection. These approaches aim to improve detection accuracy while reducing human effort and time required for analysis. Nevertheless, there are several challenges that come with using neural network models for this task, including the need for large and accurately annotated datasets, the right choice of network architecture, and the tradeoff between the number of false positives and false negatives. Moreover, the generalizability of trained models across different codebases and programming languages remains an ongoing concern.

This paper looks at the different neural network models and training strategies for the detection of vulnerabilities in software. We review supervised, unsupervised, and hybrid approaches and assess their efficacy and weaknesses. Besides, this paper considers feature selection techniques, data cleaning techniques, and performance metrics used in vulnerability detection problems. The purpose of this review is to outline the recent developments in this subject area and identify potentially useful areas for future investigation.

Keywords: Neural networks, Software code fluctuations, Models of neural networks, Development of neural networks.

Стаття надійшла до редакції / Received 22.04.2025
Прийнята до друку / Accepted 14.05.2025

Постановка проблеми та її рішення

З огляду на те що сучасні програмні системи стають все більш складними, існування вразливостей збільшуватиметься і буде представляти серйозну загрозу для безпеки як окремих користувачів так і великих організацій. Традиційні методи виявлення вразливостей, такі як ручний аналіз коду та статичний аналіз на основі попередньо визначених правил нерідко бувають неефективними через обширний обсяг програмного коду та поступовий розвиток кіберзагроз.

Вирішенням проблеми в даному випадку є використання штучних нейронних мереж, які можуть автоматично виявляти можливі слабкі місця на підставі аналізу об'ємних даних. Здатність нейронних мереж навчатися на основі попереднього досвіду дозволяють їм розпізнавати навіть раніше невідомі слабкі місця.

Проте успішне навчання нейронних мереж для виявлення слабкостей пов’язане з кількома проблемами, такими як:

- потреба у створенні якісних інформативних наборів даних;
- вибір найкращої архітектури для мережі;
- зменшення кількості помилкових сигналів та недоречних висновків.

У цій статті аналізуються різні методи навчання нейронних мереж для виявлення вразливостей у програмному коді автоматичним шляхом. Досліджено контрольовані, неконтрольовані та гібридні підходи до навчання, а також ефективність різних методик обробки ознак і вибору їх для подальшого застосування в аналізах безпеки програмного забезпечення. Метою цього дослідження є оцінка сучасного стану у даній області та ідентифікація можливих напрямків майбутніх досліджень.

Аналіз існуючих методів

На сьогодні існують різні стратегічні підходи до навчання штучних нейронних мереж для

виявлення слабких місць у програмному коді, серед яких основними є:

Навчання з учителем – один з найбільш ефективних методів використання розмічених даних для виявлення вразливостей у коді програми. Такі моделі як CNN та RNN аналізують код як послідовність символів або токенів для виявлення потенційних проблем. Основним недоліком є потреба у значному обсязі якісних розмічених даних.

Навчання без нагляду - метод виявлення аномалій у коді без заздалегідь позначених даних за допомогою автоенкодерів і генеративних суперечливих мереж (GAN) разом з методами кластеризації для пошуку фрагментів коду відмінних від стандартних шаблонів. Основна проблема полягає у високому ризику помилкових спрацювань.

Гібридні підходи - це пояснення навчання з наглядом та без шляхом їх комбінаційного використання. Наприклад, моделі можуть спочатку проходити класифікацію на не відмічених даних перед тим як коригуватися за рахунок маркованих прикладів. Це дозволяє зменшити залежність від об’ємних корекцій з великою кількістю ручного маркування. Трансформерні моделі є сучасним підходом до використання архітектур на основі трансформерів у стилі CodeBERT та GraphCodeBERT. Ці моделі проходять навчання на великих обсягах коду і можуть аналізувати складні структурні зв'язки в програмних проектах.

Графові нейронні мережі (GNN) є ефективним інструментом для аналізу програмного коду у вигляді графа з вузлами (функціями, змінними), а також зв'язками між ними. GNN допомагають ідентифікувати складні взаємозв'язки між компонентами коду, що може вказувати на можливі вразливості.

Кожен з методів має свої переваги та недоліки і їх комбінування може значно покращити точність виявлення вразливостей коду. Майбутні дослідження спрямовані на оптимізацію моделей для покращення загальності та зменшення кількості помилкових сигналів.

Архітектура та реалізація моделей

Вибір типу нейронної мережі для виявлення слабкостей програмного коду залежить від характеристик даних та поставленої задачи. Основні види архітектур включають в себе:

Згорткові нейронні мережі (CNN), добре взаємодіють з кодом у формі векторних представлень чи матриць токенів і успішно виявляють місцеві закономірності;

Recurrent Neural Networks можуть бути використані для аналізу послідовностей коду шляхом збереження контексту зв’язків між елементами;

Мережі графіків (Graph Neural Networks або GNNs), використовуються для створення графових представлень програмного коду з метою виявлення складних структурних залежностей;

Моделі на основі трансформерів - такі як BERT або CodeBERT - можуть аналізувати обширний обсяг коду і зберігати складний контекст змінних і функцій.

Розробка моделей вимагає препроцесування даних, розбиття на токени (слова), побудови графіків залежностей між ними і навчання штучної нейронної мережі. Важливо також провести оцінку їх ефективності на тестових наборах даних. Приклад такого розбиття наведений в таблиці 1.

Таблиця 1

Розбиття програми на токени

Вхідний Текст	Токен	Тип Токена	Коментар
"The function computes a sum."	The	Слово	Загальний токен
	function	Слово	Термін програмування
	computes	Слово	Дієслово
	a	Стоп-слово	Частина мови
	sum	Слово	Термін математичного характеру
	.	Символ	Кінець речення
def add(a, b):	def	Ключове слово	Python код
	add	Ідентифікатор функції	Назва функції
	(	Символ	Відкриття списку параметрів
	a	Ідентифікатор змінної	Параметр функції
	,	Символ	Розділювач параметрів
	b	Ідентифікатор змінної	Параметр функції
	)	Символ	Закриття списку параметрів
	:	Символ	Початок тіла функції

Оцінка ефективності та метрики

Ефективність нейромережевих моделей у виявленні вразливостей оцінюють за допомогою різних критеріїв. Основними показниками є точність (Accuracy), яка вказує загальну ефективність класифікації, повнота (Recall), що показує здатність моделі виявляти усі вразливості, та точність позитивних прогнозів (Precision), яка демонструє частку правильно зазначених вразливостей серед усіх прогнозованих. Баланс між цими критеріями оцінюється за допомогою метрики F1. Додатково застосовується метрика ROC-AUC, яка дає можливість оцінити якість класифікації незалежно від порогових значень. Крім цього також аналізуються показники хибно позитивних (False Positive Rate) та хибно негативних (False Negative Rate) результатів для зменшення помилок. Ефективність моделей також перевіряють на незалежних тестових наборах і за допомогою перехресної валідації для уникнення перенавчання. Вдосконалення методик оцінки ефективності сприяє покращенню якості моделей. В таблиці 2 наведено результати оцінки ефективності та метрик нейромереж, при використанні різних моделей.

Таблиця 2

Оцінки ефективності та метрик нейромереж у задачі пошуку вразливостей у програмному коду

Модель	Точність (Accuracy)	Точність позитивного класу (Precision)	Повнота (Recall)	F1-міра (F1-score)	Площа під ROC-кривою (AUC-ROC)
CNN	85.4%	83.2%	78.9%	80.9%	0.89
RNN	82.1%	80.5%	76.3%	78.3%	0.86
Transformer (CodeBERT)	91.7%	90.2%	88.5%	89.3%	0.94
GNN	87.6%	86.1%	83.4%	84.7%	0.91
Гібридна модель	93.2%	92.0%	90.7%	91.3%	0.96

Аналіз метрик ефективності моделей виявлення вразливостей у програмному коді демонструє, що вибір відповідного підходу значною мірою залежить від поставлених завдань. Трансформерні моделі та графові нейронні мережі демонструють високу точність і узагальнюючу здатність, проте потребують значних обчислювальних ресурсів. Навчання під наглядом забезпечує найкращі результати за умови наявності якісних розмічених наборів даних, тоді як методи навчання без нагляду можуть бути ефективними для виявлення нових або невідомих типів вразливостей. Гібридні підходи поєднують переваги різних методів і дозволяють зменшити залежність від ручного маркування даних, покращуючи загальну продуктивність системи виявлення вразливостей.

Автоматизоване розширення навчальних наборів

Одним з ключових питань у навчанні нейромереж для виявлення вразливостей у програмному коді є нестача якісних розмічених даних. Автоматизоване розширення навчальних наборів може значно покращити ефективність моделей шляхом створення синтетичних прикладів коду з вразливостями та використання методів аугментації даних.

Один із підходів полягає у створенні коду із штучно внесеними вразливостями для навчання моделей розпізнавання складних сценаріїв. Використання методів трансформацій коду - таких як обфускація коду, переміщення операторів та зміна стилю написання - призводить до покращення загальних можливостей нейромереж. Крім цього, використання генеративних моделей, зокрема нейронних мереж на основі трансформерів дозволяє створювати реалістичні зразки вразливого коду.

Також використовуються методи автоматичного навчання, де модель спочатку навчають на вже підписаних даних, а потім використовують їх передбачення для розширення набору для навчання. Це дозволяє поступово покращувати точність виявлення слабких місць без значних зусиль на ручному маркуванні даних. Зокрема, застосування псевдо-позначених зразків дозволяло збільшити обсяг навчальних даних без прямого втручання експертів.

Одною з перспективних стратегій є використання методів передачі навчання (transfer learning), які дозволяють пристосовувати моделі, що були попередньо навчені на великих загальних наборах даних до завдань виявлення вразливостей. Це допомагає зменшити потребу в значній кількості спеціалізованих даних та прискорює досягнення високої точності моделей.

Автоматизоване розширення навчальних наборів допомагає зменшити необхідність у ручній розмітці даних, покращує продуктивність моделей і робить їх більш стабільними у контексті змін в кодовій базі. Це сприяє покращенню точності виявлення нових видів потенційних проблем та ефективному використанню сучасних нейронних мереж у галузі кібербезпеки.

Практичне застосування та інтеграція

Використання штучних нейронних мереж для автоматичного виявлення вразливостей у програмному коді є дуже важливим кроком для підвищення рівня безпеки програмного забезпечення. Такі моделі можуть бути інтегровані в процеси розробки та забезпечення безпеки (DevSecOps), що дозволяє проводити аналіз безпеки коду на початкових етапах життя програмного продукту. Це сприяє мінімалізації ризиків безпеки ще до стадій розгортання та веде до скорочення витрат на подальшу

перевірку.

Один із найважливіших напрямків використання полягає у пошаровому включенні нейромереж у системи статичного аналізу коду (SAST). Такі системи можуть застосовувати машинне навчання для оцінки коду без його запуску, що дозволяє виявити потенційні слабкі місця ще до розгортання програмного забезпечення. Прикладами таких рішень є автоматизовані інструменти для аналізу коду, які використовуються в середовищах розробки (IDE) або CI/CD-пайплайнах. Ці системи допомагають програмістам оперативно виявляти та усувати проблеми безпеки без залучення зовнішньої експертної допомоги.

Більше того, застосування штучних нейронних мереж у процесах тестування програмного забезпечення (fuzzing) сприяє більш ефективному виявленню помилок у коді. Моделі машинного навчання можуть навчитися на великих наборах вразливостей і створювати тестові приклади для виявлення дефектів у програмах з високою ймовірністю успіху. Комбінація нейромереж з класичними методами полегшує покращення продуктивності та точності тестування через аналіз попереднього результату та адаптацію стратегій генерацій тестових кейсів.

Також новітні дослідження активно працюють над розвитком автоматизованого виправлення помилок у програмному забезпеченні; моделі не лише виявляють помилки, але й пропонують варіанти їх усунення або навіть автоматично створюють покращений код. Цей підхід може значно прискорити процес виправлення потенційних загроз безпеки та зменшити труднощі для розробників. Використання технологій на основі трансформерів або великих мовних моделей (LLM) дозволяє генерувати рекомендації щодо усунення помилок.

Крім того, додавання моделей у системи моніторингу та виявлення атак у реальному часі дозволяє вчасно реагувати на загрози та підозрілу активність у програмних системах, наприклад, шляхом аналізу потоку запитів до веб-додатків нейромережами для виявлення спроб SQL ін’єкцій або XSS атак на основі поведінкових аномалій.

Завдяки використанню нейронних мереж у процесах розробки, тестування та забезпечення безпеки компаніям вдасться значно зменшити витрати на аудит безпеки і знизити ризики, пов'язані з вразливостями у програмному забезпеченні. Використання штучного інтелекту для автоматизованого виявлення, аналізу та усунення дефектів стає ключовим елементом сучасного підходу до безпеки програмного коду.

Експериментальні результати

Аналіз результатів експерименту дає змогу оцінити точність та ефективність різних методів навчання штучних нейронних мереж для автоматичного виявлення вразливостей програмного коду. Дослідження проводилося на базі декількох визначених наборів даних, включаючи CodeXGLUE, Juliet Test Suite та інші ресурси з анотованим кодом з реальними вразливостями. Результати дослідження свідчать про те, що трансформерні моделі CodeBERT і GraphCodeBERT виявляють високу точність (приблизно 92%) і здатність до узагальнення нових даних. Проте їх навчання потребувало значних обчислювальних ресурсів і часу. Графові нейронні мережі (GNN) демонструють гарний баланс між продуктивністю і точністю, особливо при обробці структурованих даних з досягнутою точністю близько 87%. Моделі з наглядом на основі LSTM і CNN можуть досягати точності від 80% до 85%, проте їх ефективність сильно залежить від якості та обсягу позначених даних. У той час як методи без нагляду, наприклад автоенкодери, мають меншу точність (приблизно 75%), їх перевага полягають у здатності виявляти нові і раніше невідомі вразливості. Отже, результати досліджень свідчать про корисність використання комбінованих методів як способу посилення точності та зменшення кількості помилкових спрацювань.

Обмеження та виклики

Розробка та реалізація штучних нейронних мереж для автоматичного виявлення вразливостей програмного коду супроводжувалася багатьма труднощами. Головні проблеми виникають через недостатність якісних та адекватно розмічених наборів даних, необхідних для успішного тренування моделей у цьому напрямку. Процес анотування даних є вимогливим за часом та потребує великих обсягів ресурсів. Додатковою проблемою є високий рівень хибних результатів, що може призводити до перевантаження розробників великою кількістю помилкових спрацювань. Покращення моделей та використання комбінованих методів можуть допомогти зменшити це явище. Обчислювальна складність є також серйозним викликом. Навчання складних нейромережевих архітектур - наприклад, трансформерів та графових нейронних мереж - потребує великої математично-обчислювальної потужності та багато часу. Це може стати перешкодою для реалізація таких рішень в реальних проектах і особливо в умовах обмежених ресурсів. Незважаючи на ці виклики, подальший прогрес у вдосконаленні методів оптимізації і розширення можливостей доступу до широкомасштабних архівів коду сприятиме покращенню продуктивності та надійності систем автоматичного виявлення вразливостей.

Висновки

У цьому дослідженні було проаналізовано моделі та методи навчання нейронних мереж для автоматичного виявлення вразливостей програмному коду. Детально розглянуто стратегію обробки та підготовки даних, а також особливості архітектури моделей та їх ефективність. Оцінка продуктивності

довела ефективність застосування машинного навчання для покращення точності виявлення вразливостей й зменшення кількості помилкових сигналів про існуючі проблеми.

Використання штучних нейронних мереж для аналізу коду, тестування та забезпечення безпеки відкриває нові можливості для автоматизації виявлення загроз. Незважаючи на значні досягнення в цьому напрямку, все ще існують деякі проблеми та виклики. Наприклад — потреба у якісних даних для ефективного навчання моделей і складність обчислень при роботі з глибокими моделями. Також важливо постійно вдосконалювати механізми інтерпретації отриманих результатів.

Майбутні наукові дослідження в цьому напрямку можуть звернутися до розробки адаптивних моделей з самонавчанням та поліпшення методів автоматичного усунення вразливостей. Впровадження цих технологій у промислове середовище може істотно покращити загальний рівень кібербезпеки програмного забезпечення та зменшити витрати на ручний код-аудит.

Література

1. Tang, G., Meng, L., Wang, H., et al. (2020). A comparative study of neural network techniques for automatic software vulnerability detection. In *Proceedings of the 2020 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, Hangzhou, China.

2. Viega, J., & Bloch, J. T. (2000). A static vulnerability scanner for C and C++ code. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC 2000)* (Vol. 257). New Orleans, LA, USA.

3. Yuan, H., Wang, B., & Niu, L. (2018). Kernel extreme learning machine for learning from label proportions. *Lecture Notes in Computer Science*, *400*, 409.

4. Harer, J. A., Kim, L. Y., Russell, R. L., et al. (2018). Automated software vulnerability detection with machine learning. *CoRR*, *abs/1803.04497*. https://arxiv.org/abs/1803.04497

5. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

6. Cao, W., Wang, X., Ming, Z., & Gao, J. (2018). A review on neural networks with random weights. *Neurocomputing*, *275*, 272–285.

7. Li, Z., Zou, D., Tang, J., et al. (2019). A comparative study of deep learning-based vulnerability detection system. *IEEE Access*, *7*, 103184–103197.

8. Wagner, D., Foster, J., Brewer, E., & Aiken, A. (2000). A first step towards automated detection of buffer overrun vulnerabilities. In *Proceedings of the Year 2000 Network and Distributed System Security Symposium (NDSS)* (pp. 3–17). San Diego, CA.

9. Ponta, S. E., Plate, H., Sabetta, A., Bezzi, M., & Dangremont, C. (2019). A manually-curated dataset of fixes to vulnerabilities of open-source software. In *Proceedings of the 16th International Conference on Mining Software Repositories* (pp. [insert page numbers]). May 2019.

10. Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., & Liu, X. (2019). A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)* (pp. 783–794). https://doi.org/10.1109/ICSE.2019.00086

11. Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2013). A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, *39*(6), 757–773. https://doi.org/10.1109/TSE.2012.70

References

1. Tang, G., Meng, L., Wang, H., et al. (2020). A comparative study of neural network techniques for automatic software vulnerability detection. In *Proceedings of the 2020 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, Hangzhou, China.

2. Viega, J., & Bloch, J. T. (2000). A static vulnerability scanner for C and C++ code. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC 2000)* (Vol. 257). New Orleans, LA, USA.

3. Yuan, H., Wang, B., & Niu, L. (2018). Kernel extreme learning machine for learning from label proportions. *Lecture Notes in Computer Science*, *400*, 409.

4. Harer, J. A., Kim, L. Y., Russell, R. L., et al. (2018). Automated software vulnerability detection with machine learning. *CoRR*, *abs/1803.04497*. https://arxiv.org/abs/1803.04497

5. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

6. Cao, W., Wang, X., Ming, Z., & Gao, J. (2018). A review on neural networks with random weights. *Neurocomputing*, *275*, 272–285.

7. Li, Z., Zou, D., Tang, J., et al. (2019). A comparative study of deep learning-based vulnerability detection system. *IEEE Access*, *7*, 103184–103197.

8. Wagner, D., Foster, J., Brewer, E., & Aiken, A. (2000). A first step towards automated detection of buffer overrun vulnerabilities. In *Proceedings of the Year 2000 Network and Distributed System Security Symposium (NDSS)* (pp. 3–17). San Diego, CA.

9. Ponta, S. E., Plate, H., Sabetta, A., Bezzi, M., & Dangremont, C. (2019). A manually-curated dataset of fixes to vulnerabilities of open-source software. In *Proceedings of the 16th International Conference on Mining Software Repositories* (pp. [insert page numbers]). May 2019.

10. Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., & Liu, X. (2019). A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)* (pp. 783–794). https://doi.org/10.1109/ICSE.2019.00086

11. Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2013). A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, *39*(6), 757–773. https://doi.org/10.1109/TSE.2012.70