https://doi.org/10.31891/2219-9365-2025-84-19
УДК 004

ПЕТЛЯК Наталія
Хмельницький національний університет
https://orcid.org/0000-0001-5971-4428
e-mail: npetlyak@khmnu.edu.ua
ГОРДЄЄВ Богдан
Хмельницький національний університет
https://orcid.org/0009-0003-7260-8429
e-mail: hordieievbv@khmnu.edu.ua
ПЕЛЕХАТА Анастасія
Хмельницький національний університет
https://orcid.org/0009-0003-8027-9039
e-mail: pelekhataa@khmnu.edu.ua
НАГОРНЯК Андрій
Хмельницький національний університет
https://orcid.org/0009-0002-7426-1361
e-mail: nahorniakam@khmnu.edu.ua

ІНТЕЛЕКТУАЛЬНИЙ МЕТОД ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ТИПУ ПЕРЕПОВНЕННЯ БУФЕРУ В ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

У сучасних умовах цифрової трансформації програмне забезпечення стало ключовим інструментом функціонування суспільства, економіки та держави. Зростання його складності супроводжується виникненням нових ризиків інформаційної безпеки, серед яких особливу небезпеку становлять переповнення буферу. Такі вразливості здатні призвести до витоку даних, порушення цілісності інформаційних систем або навіть до виконання шкідливого коду зловмисниками. Традиційні методи ручного аудиту та класичного тестування коду демонструють обмежену ефективність, адже потребують значних людських і часових ресурсів та не забезпечують належного рівня достовірності у великих масштабах. Це зумовлює актуальність пошуку нових підходів, що базуються на автоматизації та інтеграції інтелектуальних алгоритмів у процес аналізу програм. У даній роботі запропоновано метод виявлення вразливостей типу переповнення буферу, який поєднує математичне моделювання, символьне виконання програмного коду та алгоритм Q-навчання як інструмент адаптивного симулятивного покриття. Метод забезпечує підвищення точності та достовірності аналізу, а також сприяє зменшенню кількості хибнопозитивних результатів. Проведене тестування довело ефективність розробленого підходу у практичних умовах, що дозволяє розглядати його як перспективну складову сучасних систем кіберзахисту.

Ключові слова: інформаційна безпека, навчання з підкріпленням, Q-навчання, переповнення буферу, виявлення вразливостей, кібербезпека.

PETLIAK Natalia, GORDIEIEV Bohdan,
PELEKHATA Anastasia, NAGORNYAK Andriy
Khmelnytskyi National University

INTELLIGENT METHOD FOR DETECTING BUFFER OVERFLOW VULNERABILITIES IN SOFTWARE

The rapid growth of information technologies and the complexity of software systems has intensified the risks of security vulnerabilities, among which buffer overflow remains one of the most critical. Traditional methods such as manual code auditing, static and dynamic analysis, and conventional testing are limited by scalability, accuracy, and high false-positive rates. This research proposes an intelligent method for detecting buffer overflow vulnerabilities that integrates mathematical modelling, symbolic execution, and reinforcement learning through Q-learning. The approach models buffer overflow conditions formally, employs symbolic variables to explore execution paths systematically, and applies Q-learning to mitigate the path explosion problem by prioritising the most promising branches. Constraint-based test input generation ensures realistic scenarios that activate potential vulnerabilities while reducing false positives. The method was implemented using LLVM-based symbolic execution (KLEE), dynamic instrumentation (Valgrind), and Python-based Q-learning with TensorFlow and OpenAI Gym. Experimental validation on synthetic benchmarks and real-world vulnerable code demonstrated improvements of over 30% in execution path coverage and a 20% reduction in false positives compared to classical symbolic execution. The study highlights the potential of combining formal models, symbolic analysis, and reinforcement learning to improve both precision and reliability in software vulnerability detection. The developed approach shows promise for integration into modern cybersecurity tools to enhance early identification of critical defects. Future work will focus on optimising computational efficiency, extending applicability to other vulnerability classes, and validating performance in large-scale industrial systems.

Keywords: information security, reinforcement learning, Q-learning, buffer overflow, vulnerability detection, cybersecurity.

Стаття надійшла до редакції / Received 22.09.2025
Прийнята до друку / Accepted 22.11.2025

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

Інтенсивне поширення інформаційних технологій у бізнесі, державному управлінні, науці та повсякденному житті призвело до безпрецедентного зростання залежності суспільства від надійності програмного забезпечення [1]. Будь-які вразливості коду здатні становити загрозу не лише окремим користувачам, а й критичним інфраструктурним системам. Одним із найпоширеніших та водночас найбільш небезпечних типів вразливостей залишаються переповнення буферу [2-3]. Вони виникають у результаті неконтрольованого запису даних у пам'ять за межами відведеного буфера, що відкриває можливості для некоректної роботи програм, порушення їхньої логіки або несанкціонованого доступу зловмисників [4]. Особливу небезпеку становить потенціал таких помилок призводити до віддаленого виконання коду, що робить їх одним із найпопулярніших векторів атак.

Традиційні підходи до виявлення вразливостей, зокрема ручний аудит коду чи класичне тестування, залишаються важливими у практиці забезпечення кібербезпеки, проте вони не здатні задовольнити вимоги масштабних і складних проєктів. Статичний аналіз надає можливість досліджувати вихідний код без його виконання, але часто супроводжується високим відсотком хибнопозитивних результатів. Динамічний аналіз дозволяє виявляти помилки під час фактичного виконання програм, проте вимагає значних обчислювальних ресурсів і часу. Символьне виконання поєднує переваги обох підходів, адже оперує символьними змінними замість конкретних вхідних даних, що дає змогу аналізувати велику кількість можливих шляхів виконання програми. Проте цей метод стикається із серйозним викликом — явищем «вибуху шляхів», коли кількість варіантів виконання зростає експоненційно зі збільшенням складності коду. Подолання цієї проблеми потребує впровадження інтелектуальних механізмів, здатних оптимізувати процес вибору найперспективніших гілок аналізу.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

У сучасній науковій літературі спостерігається стале зростання інтересу до автоматизованих методів виявлення вразливостей у програмному забезпеченні, особливо в контексті проблеми переповнення буфера та похідних від неї загроз. Значна частина досліджень орієнтована на інтеграцію статичного й динамічного аналізу з методами машинного навчання, що відкриває нові можливості для точнішої ідентифікації складних вразливостей.

У методі GRACE [5] для виявлення вразливостей використовується поєднання великої мовної моделі та структурної інформації у вигляді графів, що дозволяє враховувати як локальний, так і глобальний контекст програми. Його архітектура включає три взаємопов'язані модулі: вибір демонстрацій, генерацію графових структур і розширене виявлення вразливостей. Такий підхід забезпечує інтеграцію синтаксичних, семантичних і лексичних характеристик коду, що підвищує точність класифікації та знижує кількість хибнопозитивних результатів. Попри очевидні переваги, метод потребує значних ресурсів і залежить від якості навчальних даних, що обмежує масштабованість у промислових середовищах.

У роботі над VulBERTa [6] акцент зроблено на використанні попередньо навченої моделі RoBERTa, адаптованої до програмного коду мов C/C++, що дає змогу відтворювати як синтаксичні, так і семантичні характеристики. Метод дозволяє здійснювати як бінарну, так і багатокласову класифікацію вразливостей, проте вимагає значних обчислювальних ресурсів та якісних даних для навчання. У методі LineVD [7] пропонується аналіз коду на рівні окремих операторів із використанням поєднання графових нейронних мереж і трансформерів, що забезпечує точність навіть у складних випадках, хоча й призводить до високих обчислювальних витрат і ризику хибнопозитивних спрацювань. У свою чергу, метод VulCNN [8] інтерпретує вихідний код у вигляді зображень для подальшої обробки згортковими нейронними мережами, що дає змогу поєднати масштабованість і збереження семантики, проте створює додаткові витрати на попередню обробку та знижує інтерпретованість.

Інший напрямок досліджень зосереджується на інтеграції інструментів безпосередньо у процес розробки. Так, AIBugHunter [9] забезпечує виявлення та навіть часткове усунення вразливостей під час написання коду в середовищі Visual Studio Code, поєднуючи багатофункціональний аналіз і трансформерні моделі. VulANalyzeR [10] орієнтований на аналіз двійкового коду, використовуючи рекурентні архітектури, графові згортки та механізм уваги для не лише виявлення, а й класифікації вразливостей. Обидва підходи значно розширюють інструментарій практичної кібербезпеки, але залишаються ресурсоємними й чутливими до якості навчальних даних.

Класичні дослідження також підтверджують важливість проблеми. У роботі [11] на основі аналізу реальних проєктів Linux, Mozilla та Xen показано низьку результативність поширених статичних аналізаторів і відсутність кореляції між програмними метриками та наявністю переповнень буфера. У статті [12] запропоновано оптимізацію символьного виконання через таргетований аналіз «тестових блоків», що дає змогу зменшити вибух шляхів, хоча метод залишається залежним від якості специфікацій вразливостей. У роботі [13] презентовано модифікацію графа властивостей коду ContextCPG, що інтегрує імена змінних та типи даних і підвищує точність виявлення на 8%, хоча метод обмежений кількома класами вразливостей.

Додаткові підходи охоплюють поєднання формальних методів і експертного досвіду. У статті [14] запропоновано мережу вразливостей на основі мережі Петрі, інтегровану з графами залежностей і керування, що покращує виявлення вразливостей типу «забруднення», хоча масштабні експерименти ще відсутні. У статті [15] описано гібридну систему для Java, яка поєднує графові та послідовнісні методи й досягає точності 99,2%, проте обмежується лише однією мовою. У роботі [16] представлено фреймворк на базі трансформаторних вбудовувань для аналізу скомпільованого коду TEDVIL, що досягає точності 92,5% і демонструє ефективність навіть в умовах обмежених ресурсів, але зосереджується лише на стекових переповненнях. Нарешті, стаття [17] присвячена вразливостям типу Use-After-Free, пропонуючи перший комплексний огляд методів їх виявлення та зменшення наслідків, де показано переваги й обмеження статичних і динамічних детекторів.

Загалом аналіз публікацій [5]–[17] демонструє досягнення у сфері автоматизованого виявлення вразливостей, проте водночас підкреслює обмеження на рівні масштабованості, універсальності та інтерпретованості сучасних методів. Це створює підґрунтя для подальших досліджень, спрямованих на інтеграцію різних підходів, оптимізацію обчислювальних витрат та розробку методів, здатних ефективно працювати з новими й рідкісними класами вразливостей.

ФОРМУЛЮВАННЯ ЦІЛЕЙ СТАТТІ

Метою дослідження є розробка методу виявлення вразливостей типу переповнення буферу у програмному забезпеченні, який забезпечує підвищення точності, достовірності та ефективності діагностики завдяки інтеграції математичного моделювання, символьного виконання та алгоритмів навчання з підкріпленням, зокрема Q-навчання. Запропонований підхід покликаний подолати обмеження традиційних методів, зокрема проблему вибуху шляхів у символьному виконанні, а також знизити рівень хибнопозитивних результатів при аналізі складних програмних систем.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

Інтелектуальний метод виявлення вразливостей типу переповнення буферу в програмному забезпеченні

Запропонований метод виявлення вразливостей типу переповнення буферу у програмному забезпеченні ґрунтується на інтеграції трьох взаємопов’язаних компонентів: математичного моделювання переповнення буферу, символьного виконання програмного коду та алгоритму Q-навчання як інструмента адаптивного симулятивного покриття. Поєднання цих підходів дозволяє не лише формалізувати умови виникнення критичних помилок, а й оптимізувати процес аналізу шляхів виконання програм, спрямовуючи обчислювальні ресурси на найперспективніші області.

Основою методу є математична модель переповнення буферу, що відображає співвідношення між вхідними даними, розміром виділеної пам’яті та фактичним записом інформації у буфер. У моделі використовується поняття граничних значень, при яких відбувається вихід за межі відведеної області. Такий підхід дозволяє формально описати ситуації, за яких виникає вразливість, і надалі використовувати ці умови для побудови системи обмежень. Ця система виступає базисом для подальшої генерації тестових вхідних даних, спрямованих на активацію потенційно небезпечних сценаріїв у роботі програм. Другим елементом методу є символьне виконання, яке забезпечує аналіз програм не з конкретними вхідними значеннями, а з абстрактними символьними змінними. Це дає можливість охоплювати одночасно широкий простір можливих станів програми. У процесі символьного виконання формується дерево виконання, де кожен вузол відповідає певному стану програми, а кожна гілка відображає можливу траєкторію її роботи залежно від вхідних даних. Для виявлення переповнень буферу особливу увагу приділено генерації таких вхідних даних, які можуть призвести до порушення умов безпеки. Проте ключовою проблемою цього підходу є експоненційне зростання кількості шляхів при збільшенні складності програмного коду. Явище так званого «вибуху шляхів» унеможливлює повне дослідження навіть для середніх за розміром програм, що знижує практичну цінність класичного символьного аналізу.

Щоб подолати зазначене обмеження, у метод інтегровано алгоритм Q-навчання, що належить до класу методів навчання з підкріпленням. Його використання дозволяє системі аналізу виконувати вибір траєкторій у дереві виконання таким чином, щоб пріоритет надавався шляхам, які є найбільш перспективними з точки зору виявлення вразливостей. У контексті цього завдання агентом виступає система символьного аналізу, а середовищем простір усіх можливих шляхів виконання програми. Дії агента полягають у виборі конкретної гілки дерева для подальшого дослідження, тоді як функція винагороди будується з урахуванням результатів цього вибору. Зокрема, вищі значення винагороди нараховуються у випадках, коли вибір шляху призводить до збільшення охоплення коду, активації нових сценаріїв виконання або виявлення потенційно небезпечних ситуацій. Завдяки цьому алгоритм поступово навчається розпізнавати найбільш продуктивні напрями аналізу й віддавати перевагу тим шляхам, які з більшою ймовірністю приховують вразливості.

Формалізація симулятивного покриття реалізується шляхом побудови дерева обмежень, яке поєднує логічні умови, що виникають під час символьного виконання, з математичною моделлю переповнення

буферу. Таким чином, кожна гілка дерева відображає не лише можливий сценарій роботи програми, а й відповідні умови, за яких може виникати небезпечна ситуація. Алгоритм Q-навчання використовує цю інформацію для прийняття рішень, які гілки аналізувати першочергово. У результаті досягається баланс між глибиною та широтою пошуку, що значно підвищує ефективність аналізу порівняно з класичними методами.

Важливою складовою методу є алгоритм генерації вхідних даних, побудований на основі розв'язання системи обмежень для конкретних шляхів і вузлів у дереві виконання. Такий підхід дозволяє створювати тестові набори, що максимально відповідають умовам, які можуть призвести до переповнення буферу. Використання цього алгоритму не лише підвищує точність діагностики, а й скорочує кількість хибнопозитивних результатів, адже тестування спрямовується на реально небезпечні ділянки коду.

Запропонований метод реалізовано у вигляді прототипу з використанням сучасних інструментів аналізу програмного забезпечення. Зокрема, застосовано фреймворки динамічного бінарного інструментування, які забезпечують можливість дослідження виконуваних файлів незалежно від мови програмування, та інструменти символьного виконання, що підтримують проміжні представлення коду. Це робить метод універсальним і придатним для широкого спектра практичних застосувань. Узагальнюючи, метод виявлення вразливостей типу переповнення буферу у програмному забезпеченні можна розглядати як багаторівневу систему, де математична модель забезпечує формальне описання проблеми, символьне виконання створює основу для систематичного аналізу коду, а Q-навчання оптимізує процес пошуку, спрямовуючи його в найбільш критичні області. Така інтеграція дозволяє подолати ключові обмеження традиційних підходів і досягти більш високих показників точності, надійності та практичної придатності у сфері кібербезпеки.

Експериментальна перевірка методу здійснювалася у середовищі Linux (Ubuntu 20.04) на апаратній платформі з процесором Intel Core i7, 16 ГБ оперативної пам'яті та підтримкою віртуалізації. Для реалізації символьного виконання використовувався інструментарій KLEE, інтегрований з фреймворком LLVM, що забезпечило можливість аналізу вихідного коду мов C та C++. Динамічне бінарне інструментування проводилося із застосуванням Valgrind, тоді як компонент Q-навчання було реалізовано у Python з використанням бібліотек TensorFlow та OpenAI Gym. Тестова база включала набір невеликих програм із навмисно закладеними переповненнями буферу, приклади з відкритих репозиторіїв вразливостей та середньорівневі бібліотеки з відомими дефектами управління пам'яттю. Така комбінація дозволила оцінити роботу методу як у контрольованих умовах, так і на реальних програмних прикладах, а середовище тестування забезпечило відтворюваність експериментів та можливість масштабування аналізу.

Тестування запропонованого методу здійснювалося на наборі програм із відомими вразливостями типу переповнення буферу, що дозволило порівняти ефективність класичного символьного виконання та символьного аналізу з інтеграцією Q-навчання. У процесі дослідження було зафіксовано, що стандартні алгоритми символьного виконання стикаються з ефектом вибуху шляхів, через що значна частина обчислювальних ресурсів витрачається на дослідження малоперспективних або повторюваних гілок виконання. Це призводило до збільшення часу аналізу та зниження точності виявлення критичних помилок. Використання Q-навчання дозволило оптимізувати процес вибору траєкторій виконання програми. Алгоритм поступово навчився надавати пріоритет шляхам, що ведуть до активації нових умов, розширення покриття коду або безпосереднього виявлення небезпечних ситуацій. У результаті середня кількість унікальних шляхів, опрацьованих під час аналізу, зросла на понад 30 % порівняно з базовим символьним виконанням. Це забезпечило більш повне охоплення програми та дозволило виявити вразливості, які залишалися непоміченими у класичній конфігурації. Ще одним важливим результатом стало зниження кількості хибнопозитивних спрацювань. Завдяки таргетованому дослідженню найбільш перспективних шляхів кількість некоректних діагнозів скоротилася приблизно на 20 %. Таким чином, підвищилася не лише продуктивність аналізу, а й його достовірність, що є важливим для практичного застосування у сфері кібербезпеки. Попри зростання ефективності, інтеграція Q-навчання супроводжувалася збільшенням часу обробки на етапі початкового навчання агента. Однак цей недолік компенсувався стабільним зростанням якості результатів після кількох циклів тренування. У підсумку метод продемонстрував здатність до адаптації, забезпечивши суттєве покращення показників точності та надійності аналізу у порівнянні з традиційними підходами.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Розроблений метод виявлення вразливостей типу переповнення буферу в програмному забезпеченні ґрунтується на інтеграції символьного виконання з алгоритмом Q-навчання, що дозволяє підвищити точність і достовірність аналізу та мінімізувати проблему експоненційного зростання кількості шляхів. Математичне моделювання вразливостей і побудова дерева обмежень створюють основу для формалізації процесу аналізу та генерації вхідних даних, а адаптивне симулятивне покриття забезпечує спрямованість пошуку у найбільш критичні області. Експериментальні результати підтвердили ефективність методу, показавши переваги над класичними підходами як за точністю, так і за здатністю виявляти приховані помилки. Практичне значення роботи полягає у можливості інтеграції запропонованого методу у сучасні інструменти кіберзахисту, що

дозволить своєчасно ідентифікувати критичні дефекти у програмному забезпеченні та мінімізувати ризики їх експлуатації зловмисниками. Подальші дослідження передбачають удосконалення алгоритмів для зменшення обчислювальних витрат, розширення набору тестових даних та перевірку універсальності методу для інших типів вразливостей. Отримані результати підтверджують перспективність поєднання символьного виконання та навчання з підкріпленням у сфері підвищення безпеки програмного забезпечення.

Література

1. Гнатюк С., Сидоренко В., Скуратівський А. Модель управління вимогами кібербезпеки при впровадженні програмного забезпечення / С. Гнатюк, В. Сидоренко, А. Скуратівський // Кібербезпека: освіта, наука, техніка. — 2025. — Вип. 4. — С. 715—726. — https://doi.org/10.28925/2663-4023.2025.28.841
2. Yongxu H., Ying Z., Pengzhi X., Zhen G. Research on Buffer Overflow Vulnerability Mining Method Based on Deep Learning // 2024 2nd International Conference on Big Data and Privacy Computing (BDPC), Macau, China. — 2024. — P. 28—36. — https://doi.org/10.1109/BDPC59998.2024.10649293
3. Al-Mandhari I., AlKalbani A., Al-Abri A. Association Rules for Buffer Overflow Vulnerability Detection Using Machine Learning / I. Al-Mandhari, A. AlKalbani, A. Al-Abri // In: Yang X. S., Sherratt R. S., Dey N., Joshi A. (eds) Proceedings of Eighth International Congress on Information and Communication Technology (ICICT 2023). Lecture Notes in Networks and Systems. — Vol. 696. — Springer, Singapore, 2024. — https://doi.org/10.1007/978-981-99-3236-8_48
4. Butt M. A., Ajmal Z., Khan Z. I., Idrees M., Javed Y. An In-Depth Survey of Bypassing Buffer Overflow Mitigation Techniques // Applied Sciences. — 2022. — Vol. 12, no. 13. — P. 6702. — https://doi.org/10.3390/app12136702
5. Lu G., Ju X., Chen X., Pei W., Cai Z. GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning // Journal of Systems and Software. — 2024. — Vol. 212. — https://doi.org/10.1016/j.jss.2024.112031
6. Hanif H., Maffeis S. VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection // 2022 International Joint Conference on Neural Networks (IJCNN), Padua, Italy. — 2022. — P. 1—8. — https://doi.org/10.1109/IJCNN55064.2022.9892280
7. Hin D., Kan A., Chen H., Babar M. A. LineVD: statement-level vulnerability detection using graph neural networks // Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22). — New York: ACM, 2022. — P. 596—607. — https://doi.org/10.1145/3524842.3527949
8. Wu Y., Zou D., Dou S., Yang W., Xu D., Jin H. VulCNN: an image-inspired scalable vulnerability detection system // Proceedings of the 44th International Conference on Software Engineering (ICSE '22). — New York: ACM, 2022. — P. 2365—2376. — https://doi.org/10.1145/3510003.3510229
9. Fu M., Tantithamthavorn C., Le T. et al. AIBugHunter: A Practical tool for predicting, classifying and repairing software vulnerabilities // Empirical Software Engineering. — 2024. — Vol. 29, no. 4. — https://doi.org/10.1007/s10664-023-10346-3
10. Li L., Ding S. H. H., Tian Y., Fung B. C. M., Charland P., Ou W., Song L., Chen C. VulANalyzeR: Explainable Binary Vulnerability Detection with Multi-task Learning and Attentional Graph Convolution // ACM Transactions on Privacy and Security. — 2023. — Vol. 26, no. 3. — Article 28. — 25 p. — https://doi.org/10.1145/3585386
11. Pereira J. D., Ivaki N., Vieira M. Characterizing Buffer Overflow Vulnerabilities in Large C/C++ Projects // IEEE Access. — 2021. — Vol. 9. — P. 142879—142892. — https://doi.org/10.1109/ACCESS.2021.3120349
12. Baradaran S., Heidari M., Kamali A., Mouzarani M. A unit-based symbolic execution method for detecting memory corruption vulnerabilities in executable codes // International Journal of Information Security. — 2023. — Vol. 22, no. 5. — P. 1277—1290. — https://doi.org/10.1007/s10207-023-00691-1
13. Rozi M. F., Ban T., Ozawa S., Yamada A., Takahashi T., Inoue D. Securing Code With Context: Enhancing Vulnerability Detection Through Contextualized Graph Representations // IEEE Access. — 2024. — Vol. 12. — P. 142101—142126. — https://doi.org/10.1109/ACCESS.2024.3467180
14. Wang P., Liu S., Liu A., Jiang W. Detecting security vulnerabilities with vulnerability nets // Journal of Systems and Software. — 2024. — Vol. 208. — Art. no. 111902. — https://doi.org/10.1016/j.jss.2023.111902
15. Hussain S., Nadeem M., Baber J., Hamdi M., Rajab A., Al Reshan M. S., Shaikh A. Vulnerability detection in Java source code using a quantum convolutional neural network with self-attentive pooling, deep sequence, and graph-based hybrid feature extraction // Scientific Reports. — 2024. — Vol. 14, no. 1. — Art. no. 7406. — https://doi.org/10.1038/s41598-024-56871-z
16. McCully G. A., Hastings J. D., Xu S. TEDVIL: Leveraging Transformer-Based Embeddings for Vulnerability Detection in Lifted Code // IEEE Access. — 2025. — Vol. 13. — P. 76894—76913. — https://doi.org/10.1109/ACCESS.2025.3565980
17. Gui B., Song W., Xiong H., Huang J. Automated Use-After-Free Detection and Exploit Mitigation: How Far Have We Gone? // IEEE Transactions on Software Engineering. — 2022. — Vol. 48, no. 11. — P. 4569—4589. — https://doi.org/10.1109/TSE.2021.3121994

References

1. Hnatyuk S., Sydorenko V., Skurativskyi A. Cybersecurity requirements management model in software implementation / S. Hnatyuk, V. Sydorenko, A. Skurativskyi // Cybersecurity: Education, Science, Technique. — 2025. — Issue 4. — P. 715—726. — https://doi.org/10.28925/2663-4023.2025.28.841

2. Yongxu H., Ying Z., Pengzhi X., Zhen G. Research on Buffer Overflow Vulnerability Mining Method Based on Deep Learning // 2024 2nd International Conference on Big Data and Privacy Computing (BDPC), Macau, China. — 2024. — P. 28—36. — https://doi.org/10.1109/BDPC59998.2024.10649293

3. Al-Mandhari I., AlKalbani A., Al-Abri A. Association Rules for Buffer Overflow Vulnerability Detection Using Machine Learning / I. Al-Mandhari, A. AlKalbani, A. Al-Abri // In: Yang X. S., Sherratt R. S., Dey N., Joshi A. (eds) Proceedings of Eighth International Congress on Information and Communication Technology (ICICT 2023). Lecture Notes in Networks and Systems. — Vol. 696. — Springer, Singapore, 2024. — https://doi.org/10.1007/978-981-99-3236-8_48

4. Butt M. A., Ajmal Z., Khan Z. I., Idrees M., Javed Y. An In-Depth Survey of Bypassing Buffer Overflow Mitigation Techniques // Applied Sciences. — 2022. — Vol. 12, no. 13. — P. 6702. — https://doi.org/10.3390/app12136702

5. Lu G., Ju X., Chen X., Pei W., Cai Z. GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning // Journal of Systems and Software. — 2024. — Vol. 212. — https://doi.org/10.1016/j.jss.2024.112031

6. Hanif H., Maffeis S. VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection // 2022 International Joint Conference on Neural Networks (IJCNN), Padua, Italy. — 2022. — P. 1—8. — https://doi.org/10.1109/IJCNN55064.2022.9892280

7. Hin D., Kan A., Chen H., Babar M. A. LineVD: statement-level vulnerability detection using graph neural networks // Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22). — New York: ACM, 2022. — P. 596—607. — https://doi.org/10.1145/3524842.3527949

8. Wu Y., Zou D., Dou S., Yang W., Xu D., Jin H. VulCNN: an image-inspired scalable vulnerability detection system // Proceedings of the 44th International Conference on Software Engineering (ICSE '22). — New York: ACM, 2022. — P. 2365—2376. — https://doi.org/10.1145/3510003.3510229

9. Fu M., Tantithamthavorn C., Le T. et al. AIBugHunter: A Practical tool for predicting, classifying and repairing software vulnerabilities // Empirical Software Engineering. — 2024. — Vol. 29, no. 4. — https://doi.org/10.1007/s10664-023-10346-3

10. Li L., Ding S. H. H., Tian Y., Fung B. C. M., Charland P., Ou W., Song L., Chen C. VulANalyzeR: Explainable Binary Vulnerability Detection with Multi-task Learning and Attentional Graph Convolution // ACM Transactions on Privacy and Security. — 2023. — Vol. 26, no. 3. — Article 28. — 25 p. — https://doi.org/10.1145/3585386

11. Pereira J. D., Ivaki N., Vieira M. Characterizing Buffer Overflow Vulnerabilities in Large C/C++ Projects // IEEE Access. — 2021. — Vol. 9. — P. 142879—142892. — https://doi.org/10.1109/ACCESS.2021.3120349

12. Baradaran S., Heidari M., Kamali A., Mouzarani M. A unit-based symbolic execution method for detecting memory corruption vulnerabilities in executable codes // International Journal of Information Security. — 2023. — Vol. 22, no. 5. — P. 1277—1290. — https://doi.org/10.1007/s10207-023-00691-1

13. Rozi M. F., Ban T., Ozawa S., Yamada A., Takahashi T., Inoue D. Securing Code With Context: Enhancing Vulnerability Detection Through Contextualized Graph Representations // IEEE Access. — 2024. — Vol. 12. — P. 142101—142126. — https://doi.org/10.1109/ACCESS.2024.3467180

14. Wang P., Liu S., Liu A., Jiang W. Detecting security vulnerabilities with vulnerability nets // Journal of Systems and Software. — 2024. — Vol. 208. — Art. no. 111902. — https://doi.org/10.1016/j.jss.2023.111902

15. Hussain S., Nadeem M., Baber J., Hamdi M., Rajab A., Al Reshan M. S., Shaikh A. Vulnerability detection in Java source code using a quantum convolutional neural network with self-attentive pooling, deep sequence, and graph-based hybrid feature extraction // Scientific Reports. — 2024. — Vol. 14, no. 1. — Art. no. 7406. — https://doi.org/10.1038/s41598-024-56871-z

16. McCully G. A., Hastings J. D., Xu S. TEDVIL: Leveraging Transformer-Based Embeddings for Vulnerability Detection in Lifted Code // IEEE Access. — 2025. — Vol. 13. — P. 76894—76913. — https://doi.org/10.1109/ACCESS.2025.3565980

17. Gui B., Song W., Xiong H., Huang J. Automated Use-After-Free Detection and Exploit Mitigation: How Far Have We Gone? // IEEE Transactions on Software Engineering. — 2022. — Vol. 48, no. 11. — P. 4569—4589. — https://doi.org/10.1109/TSE.2021.3121994