УДК 004.056

**В. О. Паламарчук**
**В. В. Лукічов**

# МЕТОДИ ВИЯВЛЕННЯ ПРИХОВАНИХ ФУНКЦІЙ У ВІДКРИТОМУ КОДІ

Вінницький національний технічний університет

***Анотація***

*У даній роботі розглядаються методи пошуку прихованих функцій у програмах з відкритим кодом. Акцент робиться на важливості виявлення та аналізу можливих вразливостей та потенційно небезпечних частин програмного коду для забезпечення безпеки та надійності програмних систем. Описані різноманітні техніки, включаючи статичний та динамічний аналіз коду, аналіз з використанням символьних викликів, а також методи патерн-визначення для виявлення загроз безпеки та недоліків у програмному коді. Проаналізовано переваги та недоліки кожного методу, а також їхню ефективність у реальних умовах.*

**Ключові слова:** приховані функції, безпека коду, відкритий код, аналіз вихідного коду, статичний аналіз, динамічний аналіз.

***Abstract***

*This paper explores the methods for uncovering hidden functions in open-source software programs. Emphasis is placed on the importance of identifying and analyzing potential vulnerabilities and potentially hazardous portions of code to ensure the security and reliability of software systems. Various techniques are described, including static and dynamic code analysis, symbolic execution analysis, and pattern recognition methods for detecting security threats and flaws in the source code. The advantages and disadvantages of each method are analyzed, as well as their effectiveness in real-world scenarios.*

**Keywords:** hidden function discovery, code security, open source, code analysis, static analysis, dynamic analysis.

## Вступ

За останні кілька років програми з відкритим кодом набули неабиякої популярності у сфері розробки програмного забезпечення. Західні та світові компанії все частіше віддають перевагу відкритому програмному коду, розкриваючи свої рішення для широкого кола спільноти розробників. Це відкриття сприяє збільшенню інновацій та покращенню якості програмних продуктів.

Незважаючи на те що програми розробляються і перевіряються великими групами людей, завжди існує ймовірність появи нових вразливостей, які залишаються не виявлені розробниками та спільнотою. Програма з відкритим кодом, хоча й характерно швидке виявлення та усунення помилок, всеодно стають ціллю зловмисників. Відповідно, актуальним є виявлення прихованих функцій у відкритому коді, яке допомагає автоматизувати процес перевірки програм та забезпечити безпеку та їх надійність.

Для виявлення прихованих функцій у відкритому коді використовуються різні методи, включаючи аналіз самого коду — статичний та динамічний, виявлення вразливостей та потенційних загроз безпеці, а також порівняння з уже відомими шаблонами атак. Метою цього дослідження є аналіз існуючих методів виявлення прихованих функцій у відкритому коді та визначення серед них найбільш оптимального в тих чи інших випадках.

Цей вступ визначає актуальність теми та визначає мету та об'єкт дослідження. Виявлення прихованих функцій у відкритому коді відіграє важливу роль для безпеки програмного забезпечення, і розуміння різних методів для цього може сприяти покращенню цих зусиль.

## Результати дослідження

Відкрите програмне забезпечення може мати різні типи вразливостей, такі як переповнення буфера, ін’єкції, міжсайтовий скриптинг або використання небезпечних конфігурацій. Для сканування відкритого програмного забезпечення на наявність цих проблем необхідно використовувати різноманітні інструменти

та методи. Інструменти статичного аналізу сканують вихідний код відкритого програмного забезпечення на наявність потенційних вразливостей, тоді як інструменти динамічного аналізу сканують робоче або розгорнуте відкрите програмне забезпечення на наявність вразливостей. Крім того, ручне тестування включає в себе ручний огляд вихідного коду відкритого програмного забезпечення, документації або функціоналу на предмет вразливостей. Цей метод також може включати етичне вторгнення або тестування на проникнення для виявлення та використання будь-яких недоліків у відкритому програмному забезпеченні. Усі ці інструменти та методи можуть доповнювати один одного, надаючи більше інформації та контексту про проблеми. Приклади інструментів статичного аналізу — SonarQube, Coverity або OWASP Dependency Check; приклади інструментів динамічного аналізу — Nmap, ZAP або Metasploit. Крім статичного аналізу, існують інші методи, такі як динамічний аналіз, обернений аналіз, аналіз викликів функцій та інші. Зокрема, динамічний аналіз дозволяє аналізувати поведінку програми в реальному часі, що дозволяє виявляти шкідливі функції під час їх виконання. Обернений аналіз використовується для виявлення вразливостей та прихованих функцій шляхом зворотного інженерінгу вихідного коду. Аналіз викликів функцій дозволяє відстежувати, які функції викликаються в програмі, що може допомогти виявити підозрілі або шкідливі дії.

Статичний аналіз. Багато методик та інструментів перевіряють вихідний код на вразливості після його написання, що призводить до пізнього виявлення та генерації багато помилкових сигналів про вразливості. Питання статичного аналізу полягає в тому, що складно визначити, з якими саме вразливостями працює цей метод, і є проблеми з отриманням та підтримкою актуального інструментарію. Деякі дослідники вказують на можливості статичного аналізу коду для виявлення вразливостей, проте вони приходять до висновку, що інструменти не є достатньо ефективними. Під час тестування трьох широко використовуваних комерційних інструментів виявилося, що 27% вразливостей C/C++ та 11% вразливостей Java у їхньому наборі даних були пропущені. Деякі вразливості були навіть гірші за випадкове вгадування. Це підкреслює необхідність знаходження інших методів виявлення вразливостей, оскільки статичний аналіз не є безперечно ефективним. Проте статичний аналіз все ще корисний, оскільки деякі регулятивні вимоги потребують ведення інвентаризації компонентів відкритого програмного забезпечення для вирішення ризиків. Такі інструменти, як OWASP Dependency-Check, можуть аналізувати код і створювати звіти про пов'язані записи CVE.

Динамічний аналіз. У цьому випадку часто використовується метод фазування, де вхідні дані змінюються за допомогою випадкових значень для виявлення небажаної поведінки. Дослідники вразливостей, як правило, створюють власні інструменти фазування, розглядаючи це як частину процесу вивчення та віддаючи перевагу цьому підходу перед більш систематичними методами. Фазування не потребує великих знань для виконання, проте це не дозволяє контролювати виконання програми, і для отримання результатів потрібні значні зусилля. Деякі дослідники зазначають, що фазування не масштабується, якщо використовується динамічне символьне виконання, оскільки воно досліджує шляхи коду одночасно, що може призводити до значного збільшення робочого навантаження. Symbolic execution використовує символьні значення для змінних замість конкретних значень для виконання всіх шляхів у програмі.

Ручні огляди коду. Цей метод включає ручний перегляд вихідного коду в білому ящику і, отже, потребує значної людської праці. Незважаючи на це, ручний аналіз коду виявляє вразливості, і слід пам'ятати, що огляди коду, проведені особою з відповідними знаннями з безпеки, можуть бути єдиною ефективною стратегією для вирішення вразливостей.

Нові методи виявлення. Статичний аналіз генерує надмірну кількість помилкових сигналів про вразливості, динамічний аналіз не масштабується, а ручні огляди коду забирають багато часу. Дослідження новіших методів намагається вирішити ці проблеми за допомогою цікавих та новаторських підходів.

Розподілене попитно-орієнтоване тестування безпеки. Запропоноване розподілене попитно-орієнтоване тестування безпеки передбачає багато клієнтів, що використовують відкрите програмне забезпечення, та один основний тестовий сервер. Коли введення користувача має активувати новий шлях у програмі, воно відправляється на тестовий сервер для проведення тестів на безпеку. Потім застосовується символьне виконання до цього шляху, і якщо виявляється вразливість, генерується підпис і оновлюється на всіх клієнтах для захисту.

Машинне навчання. Машинне навчання - це тип штучного інтелекту, де комп'ютери використовують алгоритми для ітеративного навчання і впізнавання закономірностей. Дослідники використали машинне навчання для передбачення вразливостей великомасштабного програмного забезпечення, такого як

операційні системи, що дозволило знизити кількість помилкових сигналів про вразливості порівняно з традиційними методами аналізу.

Для ефективного сканування відкритого програмного забезпечення (Open source software, або OSS) на наявність вразливостей спершу потрібно визначити сферу дій та цілі, вибрати правильні інструменти та методи, налаштувати та запустити сканування, а також переглянути та повідомити про результати. Під час сканування повинні бути розглянуті, які компоненти програм, залежності або конфігурації підлягають скануванню, а також визначаються основні вимоги безпеки. Крім того, слід враховувати сумісність, продуктивність, зручність використання та вартість інструментів і методів. Після налаштування всіх параметрів, які відповідають конкретній програмі і вимогам безпеки потрібно періодично або безперервно запускати сканування. Нарешті, можна проаналізувати результати, визначити пріоритетність проблем, перевірити достовірність і серйозність вразливостей, задокументувати та повідомити результати зацікавленим сторонам, таким як розробники або менеджери.

**Висновки**

Під час проведення дослідження, було зроблено висновок, що комбінування статичного та динамічного аналізу забезпечує найбільш повне та ефективне виявлення вразливостей у відкритому коді. Статичний аналіз дозволяє виявляти проблеми на ранніх стадіях розробки, а динамічний аналіз дозволяє перевіряти безпеку додатка у реальних умовах. SonarQube і OWASP Dependency-Check показали хороші результати у виявленні потенційних вразливостей у відкритому коді. ZAP виявився дуже корисним для тестування веб-додатків, а Metasploit дозволив глибше дослідити можливі вразливості при динамічному аналізі. Використання обох методів забезпечує більш повний огляд безпеки додатка. Хоча ручні огляди є трудомісткими, вони залишаються важливими для виявлення складних та прихованих вразливостей, які можуть бути пропущені автоматизованими інструментами.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. Національна база даних вразливостей (NVD). URL: https://nvd.nist.gov
2. School of Control and Computer Engineering, North China Electric Power University, Beijing, China; State Grid Information & Telecommunication Branch, Beijing, China; Victoria University, AUSTRALIA Hua Wang, Editor. Detection of Open Source Software Vulnerabilities. PLoS One. 2019; 14(8): e0221530. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6707627/
3. Millar S. Vulnerability Detection in Open Source Software: An Introduction. Rapid7 LLC, 2022. arXiv:2203.16428v1 [cs.CR] 6 Mar 2022. URL: https://www.researchgate.net/publication/359614505_Vulnerability_Detection_in_Open_Source_Software_An_Introduction
4. Rival X., Yi K. Introduction to Static Analysis: An Abstract Interpretation Perspective. Cambridge, MA: The MIT Press, 2020. ISBN 9780262043410. URL: http://kwangkeunyi.snu.ac.kr/book-mititsa-sample-pp3-32.pdf
5. Conn R. The Source Code Analysis Tool Construction Project. Center for Technology Development and Transfer, Software Engineering Department, School of Science, Technology, and Engineering, Monmouth University. URL: https://dl.acm.org/doi/10.1145/269629.269645
6. Dewhurst R. Static Code Analysis. OWASP. Contributors: Kirsten S., Nick Bloor, Sarah Baso, James Bowie, Ram ch, Evgeniy Ryzhkov, Iberiam, Ann Campbell, Ejohn20, Jonathan Marcil, Christina Schelin, Jie Wang, Fabian, Achim, Dirk Wetter, kingthorin. URL: https://owasp.org/www-community/controls/Static_Code_Analysis



***Паламарчук Владислав Олегович*** — студент групи 1БКС-20б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, email: vladpalamarchuk2003@gmail.com.

***Лукічов Віталій Володимирович*** — к. т. н., доцент, доцент кафедри захисту інформації, Вінницький національний технічний університет, м. Вінниця, e-mail: lukichov.vitalyi@vntu.edu.ua.

***Vladyslav Palamarchuk*** — a student of the 1BKS-20b group, Faculty of Information Technologies, Vinnytsia National Technical University, Vinnytsia, email: vladpalamarchuk2003@gmail.com.

***Vitalii Lukichov*** —associate professor at the Department of Information Security, Vinnytsia National Technical University, Vinnytsia, email: lukichov.vitalyi@vntu.edu.ua.