УДК 004.056

А. В. Вовк

АНАЛІЗ МЕТОДІВ СТАТИЧНОГО ДОСЛІДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вінницький національний технічний університет

Анотація

У роботі проаналізовано сучасні методи статичного дослідження програмного забезпечення для виявлення вразливостей. Розглянуто техніки дизасемблювання, декомпілілювання, аналізу потоків даних та автоматизованого пошуку патернів вразливостей. Визначено обмеження статичного дослідження та тенденції розвитку інструментів автоматизації на основі штучного інтелекту.

Ключові слова: кібербезпека**,** статичне дослідження, вразливості програмного забезпечення, реверс-інжиніринг, безпека програмного забезпечення, аналіз потоків даних.

Abstract

The paper is devoted to the modern static software analysis methods in order to define vulnerabilities. Techniques of disassembling, decompilation, data flow analysis, and automated vulnerability patterns detection are examined. Limitations of static analysis and trends of AI-based automation tools' development are identified.

Keywords: cybersecurity, static analysis, software vulnerabilities, reverse engineering, software security, data flow analysis.

Вступ

Сучасні програмні засоби стають все складнішими, а разом з цим зростає і кількість вразливостей, які можуть використати зловмисники. Статичне дослідження програмного забезпечення – це техніка дослідження програм без їх запуску, яка дозволяє виявити потенційні слабкі місця в коді, зрозуміти механізми роботи системи та знайти точки для можливої експлуатації. За даними Verizon Data Breach Investigations Report 2023, у 74% випадків порушення безпеки так чи інакше задіяний людський фактор, включаючи використання вразливостей у програмному коді [1]. Отже, мета роботи – покращити безпеку програмного забезпечення шляхом аналізу сучасних методів статичного дослідження, які використовуються для виявлення вразливостей та можливих точок експлуатації.

Результати дослідження

На відміну від динамічного дослідження, коли потрібно запускати програму і спостерігати за її поведінкою, статичні методи дозволяють досліджувати структуру коду, алгоритми роботи та потенційні проблеми безпосередньо з виконуваних файлів або вихідного коду. Досвідчений аналітик може виявити більшість критичних вразливостей саме за допомогою статичного дослідження, не запускаючи програму взагалі [2].

Одними з базових технік для роботи з програмами, коли немає доступу до вихідного коду, є дизасемблювання і декомпілювання. Дизасемблер перетворює машинний код у мову асемблера, яку людина може читати і розуміти. Популярні інструменти типу IDA Pro, Ghidra від NSA та Binary Ninja дають можливість аналізувати виконувані файли для різних процесорів – x86, ARM, MIPS та інших [3, 4]. Ghidra, яку випустили у 2019 році як безкоштовний інструмент, стала революцією у світі реверс-інжинірингу – тепер професійний реверс-інжиніринг доступний не лише тим, хто може дозволити собі дорогі комерційні продукти. Декомпілятори здійснюють подальший крок у реконструкції програмного коду – вони відновлюють високорівневі конструкції з асемблерних інструкцій. Це значно полегшує аналіз програмної логіки, проте має суттєві обмеження. Оригінальні ідентифікатори змінних, коментарі та архітектура вихідного коду втрачаються в процесі компіляції, внаслідок чого декомпільований код не відповідає авторській структурі програми. Але навіть такий псевдокод дає набагато краще уявлення про те, що робить програма, ніж чистий асемблер [3].

Аналіз потоків управління та даних дозволяє зрозуміти, як саме програма виконується і як обробляються її вхідні дані. Граф потоку управління (Control Flow Graph) відображає усі можливі шляхи виконання – умовні переходи, цикли та виклики функцій, що допомагає глибше дослідити логіку роботи програми й виявити потенційно проблемні ділянки [5]. Водночас аналіз потоків даних

показує, як інформація змінюється та передається між функціями, що є особливо корисним для пошуку вразливостей, пов'язаних із користувацьким введенням. Інструменти на кшталт radare2 та angr дозволяють автоматизувати цей процес, зокрема завдяки техніці символьного виконання, коли програма «виконується» не з конкретними, а з узагальненими символьними значеннями, що дає змогу охопити велику кількість можливих сценаріїв поведінки одночасно [5].

Для практичного застосування методів статичного дослідження використовуються програми-монітори, які дозволяють відстежувати роботу системи захисту [6]. File Monitors забезпечують аналіз роботи систем захисту програмного забезпечення з файлами, Registry Monitors дозволяють реалізувати аналіз роботи із системними файлами операційної системи, а API Monitors призначені для відстежування викликів системних функцій одним чи декількома застосунками [6].

Пошук патернів вразливостей базується на виявленні типових небезпечних конструкцій коду, які програмісти повторюють з покоління в покоління. До найпоширеніших категорій помилок належать використання небезпечних функцій типу strcpy(), strcat(), gets() у C/C++ без перевірки довжини даних, відсутність валідації користувацького введення, помилки роботи з пам'яттю, застосування застарілих криптографічних алгоритмів та race conditions при роботі з файлами [7]. Для автоматизованого пошуку таких патернів використовуються інструменти статичного дослідження, зокрема Flawfinder для C/C++, який використовує базу даних відомих небезпечних функцій та сортує знайдені вразливості за рівнем ризику від 0 до 5, а також RATS (Rough Auditing Tool for Security), що підтримує аналіз коду на C/C++, Perl, PHP та Python і виявляє переповнення буферів, race conditions та інші проблеми безпеки [7]. Однак такі інструменти використовують лексичний аналіз і не мають інформації про потік управління, потік даних чи типи даних, тому неминуче генерують багато хибних позитивних результатів і пропускають реальні вразливості. Дослідження показують, що різні інструменти мають різний баланс між повнотою виявлення та кількістю помилкових спрацьовувань – наприклад, Cppcheck виявляє найбільшу кількість вразливостей, але генерує найбільше false positives порівняно з Flawfinder та RATS [7]. Тому досвідчені дослідники завжди поєднують автоматизований пошук з ретельним ручним аналізом контексту, що дозволяє точніше оцінити, чи є знайдена проблема реальною загрозою, а не лише артефактом автоматичного аналізу.

Аналіз форматів файлів і протоколів – окрема важлива галузь статичного дослідження. Багато вразливостей виникають саме через неправильну обробку спеціально підготовлених файлів або мережевих пакетів. Дослідник може проаналізувати, яким чином програма здійснює обробку полів структур даних, чи передбачено перевірку їхніх розмірів і типів, а також чи враховано граничні умови, недотримання яких може спричинити переповнення буфера [5]. Інструменти типу 010 Editor з підтримкою шаблонів дозволяють детально досліджувати бінарні файли і знаходити невідповідності між тим, як файл повинен виглядати, і тим, як він виглядає насправді. Саме такі невідповідності часто стають точкою входу для атак.

Для систематизації та порівняння розглянутих підходів було складено порівняльну таблицю основних методів статичного дослідження (табл. 1).

Таблиця 1 – Порівняльна характеристика методів статичного дослідження ПЗ

Метод	Переваги	Обмеження	Основні інструменти
Дизасемблювання	Детальний аналіз машинного коду; підтримка різних архітектур (x86, ARM, MIPS); повний контроль над аналізом	Висока складність інтерпретації; трудомісткість; потребує глибоких знань архітектури	IDA Pro, Ghidra, Binary Ninja, radare2
Декомпілювання	Високорівневе представлення логіки; краща читабельність порівняно з асемблером; швидше розуміння алгоритмів	Втрата оригінальної структури, імен змінних та коментарів; неточність відновлення складних конструкцій	Ghidra Decompiler, Hex-Rays IDA, RetDec
Аналіз потоків управління (Control Flow)	Візуалізація логіки програми; виявлення "мертвого коду" та "недосяжного коду"; аналіз складності алгоритмів	Складність для великих програм; проблеми з динамічними викликами та непрямими переходами	IDA Pro CFG, Ghidra Graph View, angr CFG
Аналіз потоків даних (Data Flow)	Виявлення шляхів обробки користувацького введення; аналіз рівня верифікування даних	Обмеження через динамічне завантаження; складність міжпроцедурного аналізу; проблема "вибуху стану" (state explosion)	radare2, angr, CodeQL, Joern

Метод	Переваги	Обмеження	Основні інструменти
Символьне виконання	Покриття множини шляхів виконання одночасно; автоматична генерація PoC; виявлення складних вразливостей	Проблеми масштабованості (path explosion); високі вимоги до ресурсів; складність налаштування	angr, KLEE, Triton, Manticore
Пошук патернів вразливостей	Швидкість аналізу; автоматизація; не потребує глибоких знань; легка інтеграція в CI/CD	Високий рівень помилок другого роду (20-40%); пропуск складних вразливостей; лексичний аналіз без семантики	Flawfinder, RATS, Cppcheck, Semgrep, Bandit
Аналіз форматів файлів та протоколів	Виявлення невідповідностей формату; аналіз граничних умов; пошук вразливостей парсингу	Потребує знання специфікацій форматів; складність застосування для малопоширених форматів	010 Editor, Kaitai Struct, Wireshark, Scapy

Методи статичного дослідження становлять значну загрозу для безпеки програмного забезпечення, оскільки дозволяють зловмисникам досліджувати код без необхідності виконання програми, створення тестового середовища чи підготовки спеціальних вхідних даних. Ця особливість робить статичне дослідження ефективним інструментом для несанкціонованого визначення архітектури програми, виявлення вразливих функцій обробки зовнішніх даних та дослідження слабких місць у механізмах аутентифікації [2]. Однак складність сучасних програмних систем створює природні бар'єри для зловмисників: динамічне завантаження коду, механізми рефлексії та віртуальні машини утруднюють повноцінне статичне дослідження, а залежність поведінки програми від конфігураційних файлів і мережевих запитів обмежує можливості виявлення вразливостей без виконання коду [2, 6]. Застосування технік обфускування може суттєво ускладнити статичне дослідження, змушуючи потенційних зловмисників поєднувати статичні методи з динамічними [2, 6].

Масштаб сучасних програмних систем також впливає на ефективність зловмисного використання статичного дослідження. Застосунки, що містять багато рядків коду і тисячі взаємопов'язаних компонентів, потребують значних часових ресурсів для повного ручного аналізу, що може тривати місяці, навіть для досвідчених зловмисників. Хоча автоматизовані інструменти частково вирішують цю проблему, вони генерують значну кількість помилкових результатів, що вимагає додаткової експертної верифікації кожного потенційно вразливого місця [7].

Розвиток технологій статичного дослідження створює нові виклики для захисту програмного забезпечення. Інтеграція методів штучного інтелекту підвищує ефективність автоматизованого пошуку вразливостей, вдосконалені системи деобфускування знижують ефективність традиційних захисних механізмів, а гібридні підходи, що поєднують статичне і динамічне дослідження, дозволяють обходити окремі типи захисту [7]. Технології символьного виконання автоматизують генерацію експлойтів для специфічних шляхів виконання програми, тоді як спеціалізовані інструменти для мобільних застосунків, IoT-пристроїв і вбудованих систем розширюють потенційну поверхню атаки на програмне забезпечення [7].

Висновки

У роботі проаналізовано основні методи статичного дослідження програмного забезпечення, які можуть бути використані зловмисниками для виявлення вразливостей та вивчення механізмів захисту систем. Розглянуто техніки дизасемблювання, декомпілювання, аналізу потоків даних та пошуку патернів вразливостей, що становлять загрозу для безпеки програмного забезпечення при несанкціонованому використанні.

Встановлено, що природні обмеження статичного дослідження, зокрема складність сучасних програмних систем та значні часові витрати на дослідження великих застосунків, створюють певні бар'єри для зловмисників. Однак розвиток систем автоматизації на основі машинного навчання, вдосконалення гібридних підходів та поява спеціалізованих інструментів для нових платформ знижують ефективність традиційних захисних механізмів, що вимагає розробки нових методів протидії несанкціонованому статичному дослідженню програмного забезпечення. Саме тому підвищується актуальність досліджень, спрямованих на удосконалення та подальший розвиток методів захисту програмного забезпечення від несанкціонованого дослідження.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. 2023 Data Breach Investigations Report / Verizon Business. New York : Verizon, 2023. 89 p. URL: https://www.verizon.com/business/resources/reports/2023-data-breach-investigations-report-dbir.pdf (Last accessed: 13.10.2025).

2. Eilam E. Reversing: Secrets of Reverse Engineering. Indianapolis : Wiley Publishing, 2005. 592 p.
3. Eagle C. The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler. 2nd ed. San Francisco : No Starch Press, 2011. 672 p.
4. National Security Agency. Ghidra Software Reverse Engineering Framework. URL: https://ghidra-sre.org/ (Last accessed: 13.10.2025).
5. Shoshitaishvili Y. et al. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. IEEE Symposium on Security and Privacy. 2016. P. 138–157.
6. Каплун В. А., Дмитришин О. В., Баришев Ю. В. Захист програмного забезпечення : лаборатор. практикум. Вінниця : ВНТУ, 2017. 75 с.
7. Kaur A., Nayyar R. A Comparative Study of Static Code Analysis tools for Vulnerability Detection in C/C++ and JAVA Source Code. Procedia Computer Science. 2020. Vol. 171. P. 2023–2029. URL: https://www.researchgate.net/publication/341903726_A_Comparative_Study_of_Static_Code_Analysis_tools_for_Vulnerability_Detection_in_CC_and_JAVA_Source_Code (Last accessed: 13.10.2025).

Вовк Анастасія Вячеславівна – студентка групи 1БС-22б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: vovkn4172@gmail.com

Науковий керівник: ***Баришев Юрій Володимирович*** – к. т. н., доцент кафедри захисту інформації, Вінницький національний технічний університет, м. Вінниця, email: yuriy.baryshev@vntu.edu.ua.

Anastasiia Vovk – student of group 1BS-22b, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: vovkn4172@gmail.com

Supervisor: ***Yurii Baryshev*** – PhD (eng), associated professor of information protection department, Vinnytsia National Technical University, Vinnytsia, email: yuriy.baryshev@vntu.edu.ua.