КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (34), 2026
ISSN 2663 – 4023
DOI 10.28925/2663-4023.2026.34.1302
УДК 004.056.53:004.415.2

Полотай Орест Іванович
кандидат технічних наук, доцент, доцент кафедри управління інформаційною безпекою
Львівський державний університет безпеки життєдіяльності, Львів, Україна
ORCID:0000-0003-4593-8601
orest.polotaj@gmail.com

Кухарська Наталія Павлівна
кандидат фізико-математичних наук, доцент, доцент кафедри безпеки інформаційних технологій
Національний університет Львівська Політехніка, Львів, Україна
ORCID:0000-0002-0896-8361
kukharska.n@gmail.com

Маслова Наталія Олександрівна
кандидат технічних наук, доцент, доцент кафедри управління інформаційною безпекою
Львівський державний університет безпеки життєдіяльності, Львів, Україна
ORCID:0000-0002-9078-0973
masgpp2@gmail.com

Сєдін Євген Олександрович
кандидат технічних наук, старший викладач кафедри управління інформаційною безпекою
Львівський державний університет безпеки життєдіяльності, Львів, Україна
ORCID:0009-0004-9654-0577
ye.sedin@ldubgd.edu.ua

Николайчук Максим Ігорович
викладач кафедри управління інформаційною безпекою
Львівський державний університет безпеки життєдіяльності, Львів, Україна
ORCID:0009-0004-6313-5196
nykolaychuk.maksym.i@ldubgd.edu.ua

ДОСЛІДЖЕННЯ МЕТОДІВ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ СИСТЕМ НА БАЗІ ОПЕРАЦІЙНОЇ СИСТЕМИ WINDOWS ІЗ РОЗРОБЛЕННЯМ ПРОГРАМНОГО ЗАСОБУ АВТОМАТИЗОВАНОГО АНАЛІЗУ

Анотація. У статті досліджено сучасні методи виявлення вразливостей програмного забезпечення та інформаційно-комунікаційних систем, а також розроблено програмний засіб автоматизованого аналізу потенційних загроз на базі операційної системи Windows. Проведено порівняльний аналіз основних підходів до виявлення вразливостей, зокрема статичного та динамічного аналізу, визначено їх переваги, недоліки та сфери практичного застосування. На основі проведеного дослідження розроблено програмний засіб мовою Python, який автоматизує процес аналізу вихідного коду, параметрів операційної системи та мережевої конфігурації з метою виявлення потенційних вразливостей. Реалізований алгоритм забезпечує аналіз програмного коду на предмет використання потенційно небезпечних конструкцій, алгоритму хешування MD5 та випадків жорсткого кодування паролів. Крім аналізу програмного забезпечення, здійснюється перевірка параметрів операційної системи Windows а також аналіз відкритих мережевих портів і конфігурації мережевих інтерфейсів. За результатами аналізу програмний засіб формує текстовий та HTML-звіти, які містять перелік виявлених потенційних вразливостей, оцінку рівня ризику, статистичну інформацію щодо результатів аналізу та рекомендації з усунення виявлених недоліків. Запропонований підхід дає змогу автоматизувати процес первинного аудиту безпеки програмного забезпечення та інформаційно-комунікаційних систем, та підвищити ефективність виявлення потенційних загроз.

Ключові слова: операційні системи, кібербезпека, кіберзагрози, безпека програмного забезпечення, програмування, Windows, інформаційно-комунікаційні системи, Python.

ВСТУП

Сучасне програмне забезпечення та інформаційно-комунікаційні системи стали невід'ємною частиною роботи підприємств, державних установ і звичайних користувачів. Разом із розвитком цифрових технологій зростає і кількість кіберзагроз, які можуть призводити до втрати даних, порушення роботи систем або несанкціонованого доступу до інформації. Однією з основних причин успішних кібератак є наявність вразливостей у програмному забезпеченні або неправильне налаштування операційної системи.

Своєчасне виявлення таких вразливостей є важливим етапом забезпечення інформаційної безпеки. На сьогодні існує велика кількість методів і програмних засобів для аналізу безпеки, однак багато з них є складними у використанні, потребують спеціальних знань або мають обмеження щодо функціональних можливостей. Тому дослідження сучасних методів виявлення вразливостей та їх практичне застосування залишається актуальним завданням.

Постановка проблеми. Незважаючи на постійний розвиток технологій захисту інформації, проблема наявності вразливостей у програмному забезпеченні залишається актуальною. Навіть сучасні програмні продукти можуть містити помилки, які виникають під час проєктування, розроблення або оновлення. Такі недоліки можуть бути використані зловмисниками для отримання несанкціонованого доступу до системи, викрадення даних або порушення її нормальної роботи.

Окрему увагу варто приділити операційній системі Windows, яка є однією з найпоширеніших платформ як серед домашніх користувачів, так і в корпоративному середовищі. Через свою популярність вона часто стає об'єктом кібератак. При цьому небезпеку можуть становити не лише вразливості встановленого програмного забезпечення, а й неправильні налаштування самої операційної системи, відкриті мережеві служби, застарілі компоненти або відсутність необхідних оновлень.

Для виявлення потенційних загроз сьогодні використовуються різні методи аналізу та спеціалізовані програмні засоби. Проте значна частина таких рішень орієнтована на професійних фахівців, потребує складного налаштування або є комерційною. Через це використання подібних інструментів не завжди є зручним для навчальних цілей, невеликих організацій або користувачів, які лише починають працювати у сфері інформаційної безпеки.

Ще однією проблемою є те, що багато засобів аналізу зосереджені лише на окремих аспектах безпеки. Одні перевіряють вихідний код програм, інші аналізують мережеву конфігурацію або стан операційної системи. Для отримання повної картини нерідко доводиться використовувати декілька різних інструментів, що ускладнює процес оцінювання захищеності інформаційно-комунікаційної системи. Також не слід забувати і необхідність проведення пост-аналізу здійснених інцидентів кібербезпеки для подальшого уникнення в майбутньому таких випадків [15].

У зв'язку з цим виникає потреба у дослідженні існуючих методів виявлення вразливостей та створенні програмного засобу, який поєднуватиме автоматизований аналіз програмного забезпечення, окремих параметрів безпеки операційної системи Windows і мережевого середовища. Такий підхід дозволить спростити первинне оцінювання рівня захищеності системи та своєчасно виявляти потенційні проблеми, що потребують усунення.

Аналіз останніх досліджень і публікацій. Проблема виявлення вразливостей програмного забезпечення та інформаційно-комунікаційних систем є одним із найбільш актуальних напрямів сучасних досліджень у сфері кібербезпеки. У наукових роботах значна увага приділяється розробленню методів автоматизованого аналізу програмного коду, пошуку потенційних вразливостей на ранніх етапах життєвого циклу програмного забезпечення та впровадженню сучасних засобів тестування безпеки. Особливий інтерес становлять методи статичного (SAST) та динамічного (DAST) аналізу, а також їх поєднання з інтерактивним аналізом (IAST), що дозволяє підвищити ефективність виявлення потенційних загроз.

Серед українських науковців варто відзначити роботу А. Гапона, В. Федорченка та О. Сєвєрінова, у якій розглянуто сучасні методи та засоби статичного й динамічного аналізу програмного коду, наведено їх переваги, недоліки та особливості практичного застосування. Автори підкреслюють, що використання автоматизованих засобів аналізу дозволяє виявляти значну частину помилок ще на етапі розроблення програмного забезпечення [10].

Вагомий внесок у розвиток методів автоматизованого виявлення вразливостей зробили Є. Кубюк та Г. Кисельов, які здійснили порівняльний аналіз сучасних підходів до пошуку вразливостей програмного коду на основі абстрактного синтаксичного дерева (AST) та графів залежностей програм (PDG). Результати дослідження свідчать, що комбінування різних методів аналізу дозволяє підвищити точність виявлення потенційних вразливостей порівняно із застосуванням лише одного підходу [3].

У більш нових дослідженнях значна увага приділяється використанню штучного інтелекту для автоматизації аналізу безпеки. Зокрема, А. Дрозд та Д. Микуляк запропонували інтелектуальну систему автоматичного виявлення вразливостей вебзастосунків, яка поєднує традиційні методи аналізу безпеки із сучасними моделями машинного навчання та великими мовними моделями. Автори відзначають перспективність інтеграції таких підходів у процес безпечної розробки програмного забезпечення [12].

Серед зарубіжних досліджень важливе місце займає емпірична робота, опублікована у журналі Computers & Security, у якій проаналізовано понад сотню наукових публікацій, присвячених методам пошуку вразливостей програмного забезпечення. Автори порівнюють ефективність статичного, динамічного аналізу, конколічного виконання та фаззинг-тестування й доходять висновку, що жоден із методів не є універсальним, а найкращі результати забезпечує їх комплексне використання. Подібних висновків дійшли також автори дослідження, опублікованого в Journal of Systems and Software, які наголошують на необхідності поєднання різних підходів до автоматизованого пошуку вразливостей залежно від особливостей програмного забезпечення [4].

Проведений аналіз показує, що більшість сучасних досліджень зосереджена або на аналізі вихідного коду, або на виявленні вразливостей вебзастосунків із використанням окремих спеціалізованих інструментів. Водночас недостатньо уваги приділено створенню комплексних програмних засобів, які б поєднували аналіз програмного забезпечення, окремих параметрів безпеки операційної системи Windows та мережевого середовища в межах єдиного програмного рішення. Саме цей напрям є перспективним і визначає доцільність виконання даного дослідження.

Мета статті. Метою цієї роботи є дослідження методів виявлення вразливостей програмного забезпечення та інформаційно-комунікаційних систем на базі операційної системи Windows, а також розроблення власного програмного засобу автоматизованого аналізу. Передбачається проаналізувати існуючі підходи до пошуку потенційних вразливостей і створити програму, яка зможе автоматично перевіряти окремі аспекти безпеки системи та формувати результати аналізу у зручному для користувача вигляді.

ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

Однією з основних причин виникнення інцидентів інформаційної безпеки є наявність вразливостей у програмному забезпеченні та інформаційно-комунікаційних системах. Вразливість являє собою недолік або помилку в архітектурі, програмному коді, конфігурації чи налаштуваннях системи, яка за певних умов може бути використана для порушення конфіденційності, цілісності або доступності інформації. Подібні недоліки можуть виникати як на етапі проєктування програмного забезпечення, так і під час його розроблення, тестування, впровадження або експлуатації [10, 6].

У сучасних операційних системах, зокрема Windows, джерелом потенційних вразливостей можуть бути не лише програмні помилки, а й неправильно налаштовані служби, використання застарілих компонентів, несвоєчасне встановлення оновлень безпеки, відкриті мережеві порти, надмірні привілеї користувачів та некоректні параметри безпеки. Саме тому забезпечення захищеності сучасної інформаційно-комунікаційної системи потребує комплексного аналізу як програмного забезпечення, так і операційної системи та мережевого середовища, у якому вона функціонує [6, 5].

За останні роки кількість зареєстрованих вразливостей постійно зростає, що пов'язано як зі збільшенням складності програмних систем, так і з активним розвитком методів їх дослідження. Сучасні міжнародні підходи до безпечної розробки рекомендують здійснювати пошук потенційних вразливостей протягом усього життєвого циклу програмного забезпечення, а не лише на завершальному етапі тестування. Такий підхід дозволяє суттєво зменшити витрати на усунення виявлених недоліків і знизити ризик успішної реалізації кіберзагроз [6].

Зважаючи на різноманітність можливих джерел виникнення вразливостей, сьогодні застосовуються різні методи їх виявлення, які відрізняються принципами роботи, точністю отриманих результатів, вимогами до програмного забезпечення та сферою застосування. Саме тому доцільним є проведення порівняльного аналізу існуючих методів автоматизованого виявлення вразливостей, визначення їх переваг, недоліків та можливостей практичного використання під час аналізу програмного забезпечення й інформаційно-комунікаційних систем на базі операційної системи Windows [10, 6].

Ефективність забезпечення безпеки програмного забезпечення значною мірою залежить від своєчасного виявлення потенційних вразливостей. Для цього сьогодні використовуються різні методи аналізу, які відрізняються принципом роботи, етапом застосування та типом інформації, що використовується під час дослідження. Вибір конкретного методу залежить від особливостей програмного продукту, доступності його вихідного коду, середовища виконання та вимог до точності

результатів. У практиці забезпечення інформаційної безпеки найчастіше застосовується поєднання декількох методів, що дозволяє отримати більш повну картину стану захищеності системи [10, 6].

Одним із найпоширеніших підходів є статичний аналіз програмного забезпечення (Static Application Security Testing, SAST). Його особливістю є дослідження вихідного коду, проміжного коду або виконуваних файлів без запуску програми. Під час такого аналізу перевіряється логіка роботи застосунку, правильність використання функцій, потенційно небезпечні конструкції коду, помилки автентифікації, некоректна робота з пам'яттю, можливість SQL-ін'єкцій та інші типові вразливості. Перевагою SAST є можливість виявлення проблем ще на етапі розроблення програмного забезпечення, що значно зменшує витрати на їх усунення [10, 6].

Іншим важливим методом є динамічний аналіз (Dynamic Application Security Testing, DAST), який виконується під час роботи програмного забезпечення. На відміну від статичного аналізу, цей підхід не потребує доступу до вихідного коду та дозволяє оцінити поведінку застосунку в реальних умовах експлуатації. За допомогою DAST можна виявляти помилки конфігурації, проблеми автентифікації, некоректну обробку вхідних даних, а також окремі види мережевих атак. Однак цей метод не завжди дозволяє точно визначити місце виникнення вразливості у програмному коді, що є одним із його основних недоліків [10, 7].

Останніми роками широкого поширення набули інтерактивний аналіз безпеки (Interactive Application Security Testing, IAST), фаззинг-тестування (Fuzz Testing), аналіз програмних залежностей (Software Composition Analysis, SCA) та методи, що використовують алгоритми машинного навчання. Вони дозволяють підвищити точність виявлення потенційних загроз шляхом поєднання переваг статичного й динамічного аналізу, автоматичного дослідження поведінки програм та перевірки сторонніх бібліотек на наявність відомих вразливостей. Крім того, дедалі більше уваги приділяється аналізу конфігурації операційної системи, мережевих служб і параметрів безпеки, оскільки значна частина сучасних інцидентів пов'язана саме з помилками налаштування середовища виконання програмного забезпечення [6].

Таким чином, жоден із існуючих методів не забезпечує повного виявлення всіх можливих вразливостей. Саме тому в сучасних інформаційно-комунікаційних системах дедалі частіше застосовується комплексний підхід, який поєднує декілька методів аналізу та дозволяє оцінити безпеку програмного забезпечення, операційної системи й мережевого середовища одночасно. Такий підхід буде покладено в основу програмного засобу, що розроблятиметься у практичній частині даної роботи.

Статичний аналіз програмного забезпечення (Static Application Security Testing, SAST) є одним із найбільш поширених методів виявлення потенційних вразливостей ще до запуску програмного продукту. Основною особливістю цього підходу є аналіз вихідного коду, проміжного представлення або виконуваних файлів без виконання самої програми. Це дозволяє знаходити помилки програмування, небезпечні конструкції коду, порушення правил безпечної розробки та потенційні вразливості ще на етапі створення програмного забезпечення, що значно знижує вартість їх подальшого усунення [17, 9].

Принцип роботи більшості сучасних SAST-інструментів ґрунтується на побудові внутрішньої моделі програми, аналізі потоків керування (Control Flow Graph), потоків даних (Data Flow Analysis), абстрактних синтаксичних дерев (Abstract Syntax Tree) та перевірці програмного коду за набором наперед визначених правил безпеки. Під час аналізу здійснюється пошук типових помилок, серед яких переповнення буфера, некоректне використання пам'яті, SQL-ін'єкції, використання небезпечних функцій, помилки автентифікації та слабкі криптографічні алгоритми. Завдяки цьому SAST дозволяє виявляти значну частину потенційних вразливостей ще до введення програмного забезпечення в експлуатацію [9].

На сьогодні існує значна кількість інструментів статичного аналізу, серед яких широкого поширення набули CodeQL, SonarQube, Semgrep, Coverity, Fortify Static Code Analyzer та Clang Static Analyzer [17]. Кожен із них використовує власні алгоритми аналізу та підтримує різні мови програмування, однак усі вони реалізують спільний принцип автоматизованого пошуку потенційних дефектів програмного коду. Сучасні дослідження показують, що використання таких засобів значно підвищує якість програмного забезпечення, хоча проблема хибнопозитивних та хибнонегативних результатів досі залишається актуальною [1].

Попри високу ефективність, статичний аналіз має і певні обмеження. Він не враховує реальну поведінку програми під час виконання, тому не здатний виявити окремі помилки конфігурації, логічні недоліки або вразливості, що проявляються лише за певних умов експлуатації. Саме тому сучасні стандарти безпечної розробки рекомендують поєднувати статичний аналіз із динамічним тестуванням та іншими методами оцінювання безпеки програмного забезпечення. Комплексний підхід забезпечує значно вищий рівень виявлення потенційних загроз порівняно із використанням лише одного методу аналізу [6].

Динамічний аналіз безпеки додатків (Dynamic Application Security Testing, DAST) передбачає тестування працюючої системи без доступу до її вихідного коду. Інструменти DAST взаємодіють із програмою як звичайний користувач або потенційний зловмисник, надсилаючи спеціально сформовані запити та аналізуючи реакцію системи. Такий підхід дозволяє виявляти SQL-ін'єкції, міжсайтовий скриптинг (XSS), помилки автентифікації, неправильні налаштування безпеки, проблеми керування сесіями та інші вразливості, що проявляються під час виконання програми.

Основною перевагою DAST є можливість оцінювання реальної поведінки програмного забезпечення в умовах, максимально наближених до експлуатації. При цьому метод не потребує доступу до вихідного коду, що робить його універсальним для тестування сторонніх програмних продуктів. Недоліком DAST є неможливість точно визначити місце виникнення вразливості в програмному коді, а також необхідність наявності розгорнутого тестового середовища. Саме тому на практиці динамічний аналіз найчастіше використовується разом зі статичним аналізом, що дозволяє підвищити повноту виявлення потенційних загроз [2].

В таблиці 1 показано опис основних методів виявлення вразливостей програмного забезпечення.

Таблиця 1

Порівняння основних методів виявлення вразливостей

Метод	Принцип роботи	Потрібен доступ до коду	Етап застосування	Основні переваги	Основні недоліки	Приклади інструментів
SAST	Аналіз вихідного або проміжного коду без запуску програми	Так	Розроблення CI/CD	Раннє виявлення помилок, швидке виправлення, аналіз усього коду	Хибнопозитивні результати, не враховує поведінку програми	SonarQube, CodeQL, Semgrep, Fortify
DAST	Аналіз працюючої програми шляхом надсилання тестових запитів	Ні	Тестування, перед релізом	Аналізує реальну поведінку системи, не потребує коду	Не визначає місце помилки в коді, потребує тестового середовища	OWASP ZAP, Burp Suite, Acunetix
IAST	Аналіз програми під час виконання із використанням спеціальних агентів	Частково	Тестування	Висока точність, визначає місце вразливості	Потребує інтеграції у програму	Contrast Security, Invicti IAST
SCA	Аналіз сторонніх бібліотек та залежностей	Ні	Розроблення, CI/CD	Виявляє відомі CVE у бібліотеках	Не аналізує власний програмний код	Snyk, OWASP Dependency-Check, Dependabot

Як видно з таблиці 1, кожен із розглянутих методів має власні особливості, переваги та обмеження. Статичний аналіз (SAST) дозволяє виявляти потенційні вразливості ще на етапі розроблення програмного забезпечення, тоді як динамічний аналіз (DAST) орієнтований на дослідження поведінки вже працюючого застосунку. Інші методи, зокрема IAST, SCA та Fuzz Testing, спрямовані на підвищення ефективності пошуку вразливостей за рахунок аналізу середовища виконання, програмних залежностей або автоматичної генерації тестових даних. Саме тому в сучасних інформаційно-комунікаційних системах найкращі результати забезпечує комплексне використання декількох методів.

На рисунку 1 представлено порівняння існуючих методів виявлення вразливостей програмного забезпечення.

Рис. 1. Порівняння методів виявлення вразливостей

На рисунку 1 наведено узагальнену схему класифікації методів виявлення вразливостей програмного забезпечення. Вона демонструє, що всі сучасні підходи можна умовно поділити на методи, які виконують аналіз без запуску програми, та методи, що досліджують її поведінку під час виконання. Поєднання цих підходів дозволяє отримати більш повну інформацію про рівень захищеності програмного забезпечення та інформаційно-комунікаційної системи, а також підвищує ймовірність виявлення потенційних загроз.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

За результатами проведеного аналізу існуючих методів виявлення вразливостей було прийнято рішення розробити власний програмний засіб автоматизованого аналізу, який поєднуватиме перевірку програмного забезпечення, параметрів безпеки операційної системи Windows та окремих характеристик мережевого середовища. Основною метою розроблення є створення простого у використанні інструменту, здатного автоматизувати первинний аналіз системи та виявляти потенційні загрози інформаційній безпеці.

Для реалізації програмного засобу було обрано мову програмування Python, яка широко використовується у сфері кібербезпеки завдяки простоті синтаксису, великій кількості готових бібліотек та підтримці роботи з операційною системою Windows [16]. Крім того, Python дозволяє швидко реалізовувати інструменти аналізу без необхідності використання складних програмних платформ, що є важливою перевагою під час створення дослідницьких програмних засобів.

Під час розроблення передбачається використання низки стандартних і спеціалізованих модулів Python. Зокрема, модуль os буде застосовано для роботи з файловою системою, *re* – для пошуку потенційно небезпечних конструкцій за допомогою регулярних виразів, *ast* – для аналізу структури вихідного коду Python-програм, *psutil* – для отримання інформації про активні процеси, мережеві з'єднання та системні ресурси, *subprocess* – для виконання системних команд Windows, а *socket* – для аналізу мережевих параметрів. Формування підсумкового звіту планується реалізувати засобами стандартних бібліотек Python із можливістю експорту результатів у текстовий або HTML-формат.

Структура програмного засобу (рис. 2) передбачає модульну побудову, що спрощує його подальше розширення та супровід. До складу програми входитимуть модуль аналізу програмного коду, модуль перевірки параметрів безпеки операційної системи Windows, модуль аналізу мережевого середовища та модуль формування звітів. Кожен із модулів виконуватиме незалежний аналіз, після чого отримані результати будуть об'єднуватися для формування загальної оцінки рівня захищеності досліджуваної системи.

Рис. 2. Структура програмного засобу виявлення вразливостей

Модуль аналізу програмного коду здійснюватиме пошук потенційно небезпечних конструкцій, серед яких використання функцій *eval(), exec()*, небезпечної десеріалізації за допомогою *pickle*, жорстко прописаних паролів, слабких алгоритмів хешування та інших типових помилок програмування [13, 14]. Модуль аналізу операційної системи виконуватиме перевірку стану брандмауера Windows, наявності встановлених оновлень безпеки, параметрів автозавантаження, активних служб та окремих налаштувань

захисту. У свою чергу, мережевий модуль забезпечуватиме отримання інформації про відкриті порти, активні мережеві з'єднання та конфігурацію мережевих інтерфейсів.

Запропонована архітектура дозволяє реалізувати комплексний підхід до аналізу безпеки програмного забезпечення та інформаційно-комунікаційних систем. На відміну від вузькоспеціалізованих інструментів, які виконують лише окремі види перевірок, розроблений програмний засіб об'єднуватиме результати аналізу декількох компонентів системи та формуватиме єдиний звіт із переліком виявлених потенційних вразливостей, рівнем їх критичності та рекомендаціями щодо усунення.

На рис. 3 показано схематичний опис роботи запропонованого програмного засобу.

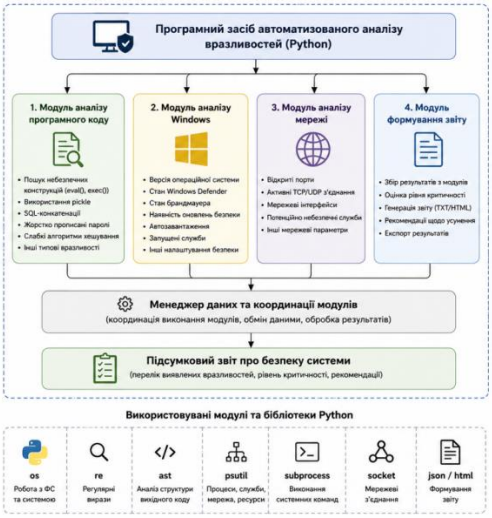

Рис. 3. Опис структури програмного засобу виявлення вразливостей

На наступному рисунку показано опис алгоритму роботи *програмного засобу виявлення вразливостей* (рис. 4).

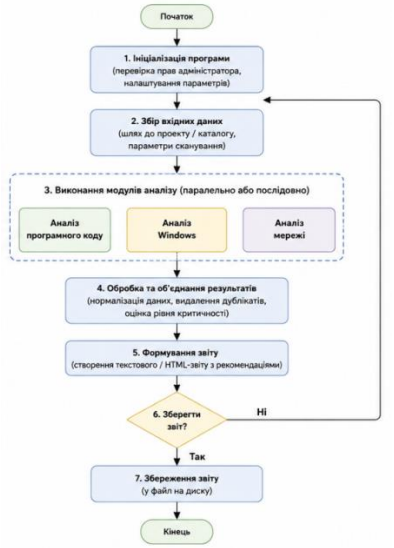

Рис. 4. Опис алгоритму роботи програмного засобу

Алгоритм роботи розробленого програмного засобу складається з кількох послідовних етапів. На початку роботи виконується ініціалізація програми, під час якої перевіряються необхідні параметри запуску, наявність доступу до системних ресурсів та встановлюються початкові налаштування аналізу. Після цього користувач задає шлях до каталогу або проєкту, який необхідно дослідити, а також обирає параметри сканування.

На наступному етапі здійснюється збір вхідних даних. Програма отримує інформацію про структуру файлів, параметри операційної системи Windows та мережеве середовище. Далі запускаються модулі аналізу, які можуть виконуватися послідовно або паралельно залежно від конфігурації системи.

Модуль аналізу програмного коду перевіряє файли на наявність потенційно небезпечних конструкцій, зокрема використання функцій eval(), exec(), небезпечної десеріалізації, SQL-конкатенацій, жорстко прописаних паролів та слабких алгоритмів хешування. Модуль аналізу Windows збирає інформацію про версію операційної системи, стан Windows Defender, брандмауера, активних служб, автозавантаження та наявність оновлень безпеки. Мережевий модуль аналізує відкриті порти, активні TCP/UDP-з'єднання, мережеві інтерфейси та потенційно небезпечні служби.

Рис. 5. Фрагмент стартового коду main.py

Після завершення роботи всіх модулів виконується обробка та об'єднання результатів. На цьому етапі програма нормалізує отримані дані, видаляє дублікати, визначає рівень критичності виявлених проблем та формує узагальнену оцінку захищеності системи.

Завершальним етапом є формування підсумкового звіту, який формується за допомогою створеного модуля *network_generator.py*, який також написаний мовою Python (рис. 6).

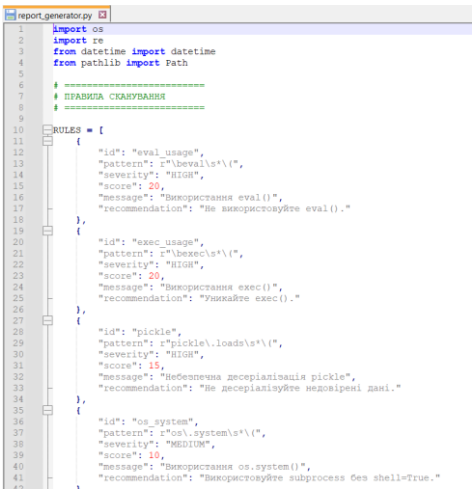

Рис. 6. Фрагмент стартового коду network_generator.py

Програма створює текстовий та HTML-звіт (рис. 7), у якому міститься перелік виявлених потенційних вразливостей, їх рівень критичності та рекомендації щодо усунення. Після підтвердження користувачем звіт зберігається на диску, що завершує роботу програмного засобу.

Для демонстрації роботи програмного засобу в межах тестового середовища було навмисно створено зразковий файл *vulnerable.py*, розміщений у директорії *samples*. Даний файл містить штучно внесені вразливості, які імітують типові помилки безпечного програмування та потенційні вектори атак. Його основною метою є забезпечення контрольованого прикладу для виявлення, аналізу та валідації функціональності сканера вразливостей, а також перевірки коректності класифікації знайдених загроз.

WinSecure Analyzer

Дата: 02.07.2026 18:16:49

Risk Score: 14/100

Рівень ризику: Високий

Результат	Рекомендація
[HIGH] Використання eval() -> рядок 2	Не використовуйте eval().
[HIGH] Використання exec() -> рядок 3	Уникайте exec().
[HIGH] Небезпечна десеріалізація pickle -> рядок 5	Не десеріалізуйте недовірені дані.
[MEDIUM] Використання os.system() -> рядок 6	Використовуйте subprocess без shell=True.
[INFO] Використання subprocess -> рядок 7	Перевіряйте параметри subprocess.
[WARNING] Порт 135 (RPC) -> рядок 13	Обмежте RPC.
[WARNING] Порт 139 (NetBIOS) -> рядок 14	Вимкніть NetBIOS за потреби.
[WARNING] Порт 445 (SMB) -> рядок 15	Закрийте SMB за відсутності потреби.

Висновок

Рекомендується усунути виявлені недоліки та повторно провести аналіз.

Рис. 7. Результат дослідження програмним засобом тестового файлу

Отримані результати свідчать про наявність ряду критичних і середньорівневих проблем безпеки, а також окремих інформаційних зауважень щодо стану системи. Інтегральний показник ризику становить 14 зі 100, однак загальна інтерпретація результатів вказує на високий рівень ризику через наявність критичних вразливостей у коді.

У процесі аналізу було виявлено використання небезпечних конструкцій динамічного виконання коду, таких як *eval* та *exec*, що може призводити до виконання довільного коду у випадку компрометації вхідних даних. Також зафіксовано застосування небезпечної десеріалізації через *pickle*, яка створює ризик виконання шкідливих інструкцій при обробці недовірених даних. Додатково виявлено використання застарілого криптографічного алгоритму MD5, що не відповідає сучасним вимогам криптографічної стійкості, а також виклики системних команд через os.system і subprocess без достатнього рівня перевірки параметрів. Окремо відзначено наявність жорстко закодованих паролів у вихідному коді, що є порушенням базових принципів безпечного зберігання секретів.

Окрім програмних вразливостей, аналіз мережевої конфігурації показав наявність відкритих служб на портах 135, 139 та 445, які характерні для Windows-середовища. Такі сервіси можуть становити потенційний ризик у разі відсутності належного обмеження доступу або використання в небезпечних мережевих сегментах. Найбільш критичними з точки зору безпеки є вразливості, пов’язані з виконанням довільного коду та небезпечною десеріалізацією, оскільки вони безпосередньо можуть призвести до повної компрометації системи.

У підсумку результати аналізу вказують на необхідність усунення виявлених недоліків, передусім у частині контролю виконання коду, безпечної обробки даних та обмеження доступу до мережевих сервісів. Повторне тестування після внесення змін дозволить оцінити ефективність виправлень та зниження загального рівня ризику.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У ході виконання дослідження було проаналізовано сучасні методи виявлення вразливостей програмного забезпечення та інформаційно-комунікаційних систем, а також визначено їх особливості, переваги й обмеження. Проведений аналіз показав, що жоден із існуючих підходів не є універсальним, тому для досягнення кращого результату доцільно поєднувати декілька методів перевірки. Саме комплексний підхід дозволяє виявляти більшу кількість потенційних загроз як у програмному коді, так і в конфігурації операційної системи та мережі.

Практичним результатом роботи стало розроблення власного програмного засобу мовою Python, який автоматизує процес пошуку потенційних вразливостей. Реалізований програмний засіб виконує статичний аналіз програмного коду, перевіряє окремі параметри безпеки операційної системи Windows, аналізує мережеву конфігурацію, формує інтегральну оцінку рівня ризику та автоматично генерує звіт із рекомендаціями щодо усунення виявлених недоліків. Проведене тестування підтвердило працездатність розробленого рішення та можливість його використання для первинного аудиту безпеки.

Отримані результати показують, що навіть відносно простий програмний засіб може суттєво спростити процес аналізу захищеності програмного забезпечення та інформаційно-комунікаційних систем. Автоматизація таких перевірок дозволяє швидше виявляти потенційні проблеми, зменшує ймовірність пропуску небезпечних елементів і допомагає сформувати загальне уявлення про рівень захищеності досліджуваного об'єкта.

У подальшому роботу доцільно розвивати в напрямі розширення функціональних можливостей програмного засобу. Зокрема, перспективним є додавання підтримки динамічного аналізу програмного забезпечення, перевірки вебзастосунків, інтеграції з базами відомих вразливостей, автоматичного формування рекомендацій на основі міжнародних стандартів кібербезпеки, а також використання методів машинного навчання для виявлення нових типів загроз. Це дозволить зробити програмний засіб

більш універсальним і придатним не лише для навчальних досліджень, а й для практичного застосування під час проведення аудиту інформаційної безпеки.

Таким чином, запропонований алгоритм забезпечує комплексний автоматизований аналіз програмного забезпечення, операційної системи Windows та мережевого середовища, що дозволяє отримати більш повну інформацію про стан безпеки досліджуваної інформаційно-комунікаційної системи. Експериментальна перевірка працездатності розробленого програмного засобу підтвердила можливість його використання для аналізу програмного коду, оцінювання стану захищеності операційної системи Windows та контролю окремих параметрів мережевої безпеки. Практична цінність роботи полягає у створенні універсального програмного засобу, який може використовуватися як допоміжний інструмент під час проведення аудиту інформаційної безпеки, навчальних досліджень і попереднього оцінювання рівня захищеності програмного забезпечення та інформаційно-комунікаційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Baca, D., Carlsson, B., Petersen, K., & Lundberg, L. (2019). Benchmarking static code analyzers. *Information and Software Technology, 112*, 68-85.
2. Grossman, J., Hansen, R., Petkov, P., Rager, A., & Fogie, S. (2007). *XSS attacks: Cross site scripting exploits and defense*. Syngress.
3. Kubiuk, Y., & Kyselov, G. (2021). Comparative analysis of approaches to source code vulnerability detection based on deep learning methods. *Technology Audit and Production Reserves, 3*(2[59]), 19-23. https://doi.org/10.15587/2706-5448.2021.233534
4. Lu, G., Ju, X., Chen, X., Pei, W., & Cai, Z. (2024). GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning. *Journal of Systems and Software, 212*, Article 112031. https://doi.org/10.1016/j.jss.2024.112031
5. MITRE Corporation. (n.d.). *Common Weakness Enumeration (CWE)*. https://cwe.mitre.org/
6. National Institute of Standards and Technology. (2022). *Secure Software Development Framework (SSDF) version 1.1: Recommendations for mitigating the risk of software vulnerabilities* (NIST Special Publication 800-218). https://doi.org/10.6028/NIST.SP.800-218
7. OWASP Foundation. (2023). *OWASP Web Security Testing Guide (WSTG)* (Version 4.2).
8. *Static analysis techniques for embedded, cyber-physical, and electronic software systems: A comprehensive survey*. (2025). *Electronics, 15*(5).
9. Youn, D., Lee, S., & Ryu, S. (2023). Declarative static analysis for multilingual programs using CodeQL. *Software: Practice and Experience, 53*(7), 1472-1495. https://doi.org/10.1002/spe.3199
10. Hapon, A. O., Fedorchenko, V. M., & Sievierinov, O. V. (2023). Methods and tools for static and dynamic code analysis. *Radioelectronics, 212*, 7-13. https://doi.org/10.30837/rt.2023.1.212.01
11. Horiuk, N. V., & Lavrovskyi, I. M. (2021). Static analysis of software source code using Fortify Static Code Analyzer. *Modern Information Security, 2*. https://doi.org/10.31673/2409-7292.2021.020910
12. Drozd, A., & Mykuliak, D. (2026). Intelligent computer system for automatic detection of web application vulnerabilities and threat classification. *Measuring and Computing Devices in Technological Processes, 1*, 368-376. https://doi.org/10.31891/2219-9365-2026-85-45
13. Diachenko, O. O., & Hrabovskyi, O. V. (2025). Software code quality assessment using static analysis. *Collection of Scientific Works of the Odesa State Academy of Technical Regulation and Quality, 1*, 129–135. https://doi.org/10.32684/2412-5288-2025-1-26-129-135
14. Kuievda, Yu. V., Tatarin, Ye. O., & Selin, Yu. M. (2025). A system for assessing the quality of object-oriented program code based on static analysis and metric visualization. *Applied Issues of Mathematical Modelling, 8*(2), 154-165. https://doi.org/10.32782/mathematical-modelling/2025-8-2-16
15. Polotai, O. I. (2023). The use of computer forensics to ensure effective investigation of information and cybersecurity incidents. *Bulletin of Lviv State University of Life Safety, 28*, 73–80. https://doi.org/10.32447/20784643.28.2023.07
16. Polotai, O. I., Kukharska, N. P., Tkachenko, A. M., Siedin, Ye. O., & Nikolaichuk, M. I. (2026). Methods for automating cybersecurity incident investigations based on Windows operating system logs using Python to support information security management. *Cybersecurity: Education, Science, Technique, 33*(1), 414-426. https://doi.org/10.28925/2663-4023.2026.33.1219
17. Spys, D. V., & Ilienko, A. V. (2025). Systematization and classification of modern methods for static analysis of web applications. *Cybersecurity: Education, Science, Technique, 2*(30), 157-179. https://doi.org/10.28925/2663-4023.2025.30.957

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (34), 2026
ISSN 2663 – 4023

Valerii Lakhno
Doctor of Technical Sciences, Professor, Professor of Department of Computer Systems and Networks
National University of Life and Environmental Sciences of Ukraine, Kyiv, Ukraine
ORCID: 0000-0001-9695-4543
lva964@nubip.edu.ua

Valerii Hladkykh
Associated Professor, Department of Computer Systems and Networks,
National University of Life and Environmental Sciences of Ukraine, Kyiv, Ukraine
ORCID: 0000-0002-5868-9504
v.hladkykh@nubip.edu.ua

Andrii Sahun
Associated Professor, Department of Computer Systems and Networks,
National University of Life and Environmental Sciences of Ukraine, Kyiv, Ukraine
ORCID: 0000-0002-5151-9203
a.sagun@nubip.edu.ua

MULTILEVEL ARCHITECTURE FOR BEHAVIORAL ARTIFACT COLLECTION IN HETEROGENEOUS NETWORKS TO FORM DIGITAL FOOTPRINTS

Abstract. The object of research is the process of generating a digital evidence array within heterogeneous information and communication systems under post-quantum cyber threats. The subject of research encompasses architectural solutions and methods for the multilevel collection of behavioral artifacts for decision support systems (DSS). The relevance of this work stems from the need to develop alternative security verification methods for network nodes in cases where current cryptographic identification (TLS, PKI) may be compromised by an adversary with access to quantum resources. Unlike existing methods focused on analyzing individual log types, this paper proposes the concept of a sensor hierarchy covering the network, system, and application levels of the infrastructure. The scientific novelty of the results lies in the formalization of the procedure for transforming disparate raw security events into structured "digital footprints." Specifically, a multilevel monitoring architecture has been developed, enabling flexible adjustments to the intensity of artifact collection depending on the network segment type and the predicted attack vector (EDoS impacts, Supply Chain Attacks). For the first time, a methodology for weighting behavioral features based on their informativeness is described, ensuring the minimization of computational costs for data processing while maintaining high reliability in forming input parameters for game-theoretic security models. The practical significance of the work lies in the development of software module structures for data aggregation that can be integrated into existing SIEM/IDS platforms. The implementation of the proposed architecture using the Python language has enabled the automation of detecting anomalous deviations in node behavior, even when digital certificates formally remain valid. Experimental validation of the architecture confirmed its ability to ensure the integrity of the DSS information base, even under conditions of changing heterogeneous environment topology and significant network noise levels.

Keywords: monitoring architecture, behavioral artifacts, digital footprints, heterogeneous networks, post-quantum security, sensor hierarchy, data aggregation.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Baca, D., Carlsson, B., Petersen, K., & Lundberg, L. (2019). Benchmarking static code analyzers. *Information and Software Technology, 112*, 68-85.
2. Grossman, J., Hansen, R., Petkov, P., Rager, A., & Fogie, S. (2007). *XSS attacks: Cross site scripting exploits and defense*. Syngress.
3. Kubiuk, Y., & Kyselov, G. (2021). Comparative analysis of approaches to source code vulnerability detection based on deep learning methods. *Technology Audit and Production Reserves, 3*(2[59]), 19-23. https://doi.org/10.15587/2706-5448.2021.233534

4. Lu, G., Ju, X., Chen, X., Pei, W., & Cai, Z. (2024). GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning. *Journal of Systems and Software, 212*, Article 112031. https://doi.org/10.1016/j.jss.2024.112031
5. MITRE Corporation. (n.d.). *Common Weakness Enumeration (CWE)*. https://cwe.mitre.org/
6. National Institute of Standards and Technology. (2022). *Secure Software Development Framework (SSDF) version 1.1: Recommendations for mitigating the risk of software vulnerabilities* (NIST Special Publication 800-218). https://doi.org/10.6028/NIST.SP.800-218
7. OWASP Foundation. (2023). *OWASP Web Security Testing Guide (WSTG)* (Version 4.2).
8. *Static analysis techniques for embedded, cyber-physical, and electronic software systems: A comprehensive survey*. (2025). *Electronics*, *15*(5).
9. Youn, D., Lee, S., & Ryu, S. (2023). Declarative static analysis for multilingual programs using CodeQL. *Software: Practice and Experience*, *53*(7), 1472-1495. https://doi.org/10.1002/spe.3199
10. Hapon, A. O., Fedorchenko, V. M., & Sievierinov, O. V. (2023). Methods and tools for static and dynamic code analysis. *Radioelectronics*, *212*, 7-13. https://doi.org/10.30837/rt.2023.1.212.01
11. Horiuk, N. V., & Lavrovskyi, I. M. (2021). Static analysis of software source code using Fortify Static Code Analyzer. *Modern Information Security*, *2*. https://doi.org/10.31673/2409-7292.2021.020910
12. Drozd, A., & Mykuliak, D. (2026). Intelligent computer system for automatic detection of web application vulnerabilities and threat classification. *Measuring and Computing Devices in Technological Processes*, *1*, 368-376. https://doi.org/10.31891/2219-9365-2026-85-45
13. Diachenko, O. O., & Hrabovskyi, O. V. (2025). Software code quality assessment using static analysis. *Collection of Scientific Works of the Odesa State Academy of Technical Regulation and Quality*, *1*, 129–135. https://doi.org/10.32684/2412-5288-2025-1-26-129-135
14. Kuievda, Yu. V., Tatarin, Ye. O., & Selin, Yu. M. (2025). A system for assessing the quality of object-oriented program code based on static analysis and metric visualization. *Applied Issues of Mathematical Modelling*, *8*(2), 154-165. https://doi.org/10.32782/mathematical-modelling/2025-8-2-16
15. Polotai, O. I. (2023). The use of computer forensics to ensure effective investigation of information and cybersecurity incidents. *Bulletin of Lviv State University of Life Safety*, *28*, 73–80. https://doi.org/10.32447/20784643.28.2023.07
16. Polotai, O. I., Kukharska, N. P., Tkachenko, A. M., Siedin, Ye. O., & Nikolaichuk, M. I. (2026). Methods for automating cybersecurity incident investigations based on Windows operating system logs using Python to support information security management. *Cybersecurity: Education, Science, Technique, 33*(1), 414-426. https://doi.org/10.28925/2663-4023.2026.33.1219
17. Spys, D. V., & Ilienko, A. V. (2025). Systematization and classification of modern methods for static analysis of web applications. *Cybersecurity: Education, Science, Technique*, *2*(30), 157-179. https://doi.org/10.28925/2663-4023.2025.30.957

Отримано редакцією журналу / Received: 22.01.26
Прорецензовано / Revised: 05.02.26
Схвалено до друку / Accepted: 24.09.26