УДК 004.056.53
DOI: 10.31673/2409-7292.2025.030422

Гапон А. О.

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ, ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ МЕТОДУ ЗАХИСТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ ГІБРИДНОГО АНАЛІЗУ

Зростання кіберзагроз, особливо в умовах активного поширення шкідливого програмного забезпечення, призводить до серйозних наслідків, серед яких несанкціонований доступ до конфіденційних систем, масове викрадення або втрата критичних даних, а також їх шифрування з метою вимагання. Ці події не лише завдають значної економічної шкоди, але й кваліфікуються як кримінальні правопорушення в багатьох юрисдикціях, що підкреслює їхню правову та соціальну значущість. У цьому контексті захист програмного забезпечення набув стратегічного значення, особливо на етапах його розробки, коли є можливість проактивного запобігання потенційним вразливостям. Сучасні методи аналізу коду, зокрема статичний і динамічний, демонструють суттєві обмеження у боротьбі з поліморфним та метаморфним шкідливим ПЗ. Статичний аналіз, спираючись на сигнатури, не в змозі ефективно виявляти нові форми загроз через відставання баз даних вірусів та високий рівень хибнопозитивних результатів. Динамічний аналіз, хоча й дозволяє фіксувати поведінкові ознаки шкідливого коду, є ресурсомістким, уповільнює процес тестування та чутливий до антиемуляційних технік, що приховують справжню природу загрози. Для подолання цих проблем пропонується гібридний метод аналізу коду, який синергетично поєднує переваги статичного, динамічного та семантичного підходів. Такий підхід забезпечує комплексне виявлення загроз на основі одночасного аналізу структури коду та його поведінки під час виконання, що суттєво підвищує точність детектування, зменшує кількість помилкових результатів і забезпечує ширше охоплення потенційних ризиків. Особливе значення має його застосування для раннього виявлення небезпек у широко використовуваних бібліотеках з відкритим кодом, де ризики ланцюга поставок є найвищими. Впровадження гібридного аналізу забезпечує суттєве підвищення загального рівня безпеки програмного забезпечення, оптимізацію витрат на тестування, скорочення часу на верифікацію та підвищення довіри до отриманих результатів. Цей напрямок є особливо актуальним для великомасштабних проектів із мікросервісною архітектурою та інтенсивним використанням open-source компонентів, де потреба в надійному захисті від еволюціонуючих кіберзагроз є критично важливою. Таким чином, розвиток і практичне впровадження гібридного аналізу коду становить наукову та прикладну цінність у забезпеченні кіберстійкості сучасних і перспективних інформаційних систем.

Ключові слова: шкідливе програмне забезпечення, захист програмного забезпечення, статичний аналіз коду, динамічний аналіз коду, гібридний аналіз коду, вразливості безпеки, шкідливі паттерни, поліморфні віруси, безпека коду.

Вступ

Шкідливе програмне забезпечення (ШПЗ) — це зловмисні програми або фрагменти коду, спрямовані на пошкодження кінцевих пристроїв. У разі інфікування пристрою ШПЗ може спричинити несанкціонований доступ, компрометацію даних або блокування системи до сплати викупу жертвою.

Особи, які розповсюджують ШПЗ (кіберзлочинці), керуються фінансовими мотивами та використовують інфіковані пристрої для викрадення банківських реквізитів, збору та незаконного збуту персональних даних, продажу доступу до обчислювальних потужностей, вимагання платіжної інформації [1, 15].

З юридичної точки зору, діяльність, пов’язана зі ШПЗ, кваліфікується як кримінальне правопорушення згідно з Кримінальним кодексом України:

–Ст. 361 (незаконне втручання у роботу ЕОМ/мереж, включаючи втручання у документообіг суду – ст. 376-1);

–Ст. 361-1 (розробка шкідливих програмних/технічних засобів та незаконне отримання інформації – ст. 359);

–Ст. 361-2 (поширення інформації з обмеженим доступом);

–Ст. 363-1 (пошкодження телекомунікаційних мереж – ст. 360);

–закону про оперативно-розшукову діяльність (ст. 8, п. 18, 20 – негласне зняття інформації).

Безпека коду в контексті захисту від ШПЗ – це набір заходів, спрямованих на запобігання вразливостям, протидію шкідливим атакам, запобігання несанкціонованому доступу, пошкодженню даних або порушенню функціонування ПЗ/систем.

Ключові принципи захисту коду:

– виявлення та усунення вразливостей. Систематична перевірка коду на наявність слабких місць (некоректна обробка вхідних даних, переповнення буфера, недостатня автентифікація тощо) та їх подальша ліквідація;

– мінімізація поверхні атаки. Зменшення кількості потенційних векторів атаки шляхом обмеження прав доступу, мінімізації привілеїв користувачів тощо;

– дотримання безпечних практик програмування. Використання стійких функцій, коректна обробка винятків, застосування безпечних структур даних, відмова від застарілих компонентів із відомими вразливостями;

– стійкість до атак. Забезпечення функціональності та надійності коду навіть під час атак (наприклад, коректна обробка невірних вхідних даних, запобігання витоку конфіденційної інформації);

– регулярне оновлення та підтримка. Своєчасне виправлення виявлених недоліків і адаптація до нових загроз.

Таким чином, захист коду від ШПЗ є актуальною науково-првктичною задачею та базується на всебічному підході до нейтралізації потенційних ризиків, зменшення можливостей для атак на програмні продукти.

Аналіз літературних даних та формулювання проблеми

З поширенням інформаційних технологій зростає і рівень кіберзагроз, що зумовлює потребу в ефективному захисті програмного забезпечення. Виявлення вразливостей на етапі розробки є критичним для забезпечення конфіденційності, цілісності й доступності даних. Одним з ефективних методів є статичний аналіз [2-9,12], що дозволяє виявляти помилки без виконання коду шляхом лексичного, синтаксичного, семантичного аналізу та перевірки на дотримання стандартів. Проте цей метод має обмеження – велика кількість хибних спрацювань, залежність від повноти баз патернів, низька ефективність проти поліморфних загроз.

З метою подолання цих недоліків використовується динамічний аналіз [10-11], що досліджує поведінку коду під час його виконання в контрольованому середовищі. Це дозволяє виявляти небезпечні дії, проте метод потребує значних обчислювальних ресурсів і вразливий до антианалізних технік з боку ШПЗ.

Оптимальним підходом стає гібридний аналіз, який поєднує переваги статичного, динамічного та семантичного аналізу [4, 14]. Такий підхід дає змогу зменшити кількість хибних спрацювань, підвищити точність виявлення загроз і краще адаптуватися до складних архітектур програм.

Мета роботи та цілі дослідження.

Швидкий розвиток інформаційних технологій супроводжується паралельним вдосконаленням методів кібератак, що потребує постійного оновлення підходів до захисту програмного забезпечення. Особливу увагу сьогодні приділяють ранньому виявленню вразливостей ще на етапі розробки, оскільки це дозволяє запобігти потенційним загрозам конфіденційності, цілісності та доступності даних. Серед сучасних методів захисту особливе місце займає статичний аналіз коду, який дозволяє виявляти потенційні вразливості без необхідності виконання коду, аналізуючи його на відповідність стандартам безпеки та виявляючи помилки розробки. Однак сучасні інструменти часто орієнтовані на перевірку стилю коду та загальної якості, що обмежує їх ефективність у боротьбі з кіберзагрозами.

Традиційний підхід до виявлення шкідливого програмного забезпечення (ШПЗ) базується на використанні баз паттернів –специфічних фрагментів коду або послідовностей байт, що характеризують відомі вразливості.

© Гапон А. О. (2025) Експериментальне дослідження, програмна реалізація та оцінка ефективності застосування методу захисту програмного забезпечення на основі гібридного аналізу. Сучасний захист інформації, 3(63), 27–36. https://doi.org/10.31673/2409-7292.2025.030422

Процес оновлення таких баз потребує значних ресурсів і експертних знань, оскільки включає ізольоване тестування підозрілих фрагментів коду.

Особливо критичною є проблема поліморфізму шкідливого коду, коли нові версії загроз можуть істотно відрізнятися від відомих зразків або бути реалізованими на інших мовах програмування.

Для підвищення ефективності виявлення загроз пропонуються евристичні методи, проте їх можливості обмежені через відсутність публічної інформації про технології, що використовуються антивірусними компаніями.

Перспективним напрямком є застосування штучного інтелекту, зокрема нейронних мереж, для класифікації коду на основі аналізу паттернів із визначенням коефіцієнта подібності. Однак статичний аналіз має принципове обмеження - неможливість точно визначити шляхи виконання програми без її запуску.

Це обмеження компенсується динамічним аналізом, який передбачає виконання коду в контрольованому середовищі з моніторингом його поведінки.

Найперспективнішим напрямком є гібридний підхід, який поєднує переваги статичного та динамічного аналізу. Такий метод дозволяє:

1. Виявляти ширший спектр вразливостей;
2. Знижувати кількість помилкових спрацьовувань;
3. Адаптуватися до змін у архітектурі ПЗ;
4. Ефективно працювати з великими проектами;
5. Скорочувати час виявлення та усунення помилок.

Таким чином, розробка сучасних методів захисту програмного забезпечення на основі гібридного аналізу коду є актуальним науковим завданням, що має значний практичний потенціал для підвищення кібербезпеки в умовах постійно еволюціонуючих загроз.

Виклад основного матеріалу

Реалізація та оцінка ефективності запропонованого методу проводилася на базі інструментів статичного аналізу вихідного тексту програм та інструмент динамічного символьного виконання.

Інструмент статичного аналізу вихідного коду програм є ядро аналізу програм, що реалізує:

- алгоритми аналізу абстрактного синтаксичного дерева для легковажного аналізу вихідного коду програм;

- алгоритми аналізу шляху програми, чутливого до шляхів та контексту під час проведення міжпроцедурного аналізу;

- набір аналізаторів, призначених виявлення вразливостей різних типів у вихідному коді програм.

Інформація про програму зберігається у вигляді графу викликів. Граф викликів є орієнтованим графом, у якому вершинам відповідають підпрограми, а ребрам – операції викликів підпрограм. Кожна підпрограма ідентифікується адресою точки входу до підпрограми. Далі код кожної підпрограми зберігається як орієнтованого графа, у якому вершинами є базові блоки, а ребрам відповідають переходи між базовими блоками. Для кожного базового блоку зберігається адреса першої інструкції блоку та його довжина. Дана структура, за наявності інформації про відповідність рядків та адрес, дозволяє зіставити шлях виконання у вихідному коді програми шляху виконання у машинному коді.

Насамперед варто врахувати, що кількість шляхів у машинному коді більша, ніж у коді, що виконується. Це з тим, що складні умовні висловлювання породжують додаткові шляхи в машинному коді, що з необхідністю обчислення складних умовних виразів.

Для кожного попередження про помилку проводиться аналіз шляху подій. Для кожної точки на шляху подій, яка виражена у вигляді позиції у вихідному тексті програми, визначається базовий блок, який потрапляє подію з шляху. Для цього використовується

© Гапон А. О. (2025) Експериментальне дослідження, програмна реалізація та оцінка ефективності застосування методу захисту програмного забезпечення на основі гібридного аналізу. Сучасний захист інформації, 3(63), 27–36. https://doi.org/10.31673/2409-7292.2025.030422

інформація про зіставлення виконуваного коду та вихідного тексту програми, проводиться визначення, яким підпрограм відповідає кожен базовий блок, що містить точку події з шляху подій попередження про помилку. Після чого проводиться топологічна сортування підпрограм на графі викликів з метою побудови шляхів на графі викликів від точки входу в програму або процедуру обчислювального шляху до першої підпрограми, в якій знаходиться перший базовий блок, що містить подію шляху попередження про помилку. Далі будуються шляхи на графі викликів до всіх підпрограм, що містять базові блоки, в яких розташовані точки подій шляху попередження про помилку. Таким чином, будується скорочений граф викликів для попередження про помилку.

Далі для кожної підпрограми здійснюється обчислення шляхів, що проводять виконання через базові блоки подій шляху попередження про помилку і через базові блоки, що містять операції виклику підпрограм, що увійшли до скороченого графа викликів. Таким чином, будується скорочений міжпроцедурний граф шляху програми. Після побудови скороченого графа шляху програми для кожного базового блоку, що відповідає вершині графа, проводиться обчислення та збереження відстані у вигляді кількості переходів від базового блоку до кожного з базових блоків, що містять точку події з шляху попередження про помилку. Інформація про відстань використовується в процесі динамічного виконання програми для вибору наступного шляху для аналізу, і, таким чином, реалізується алгоритм спрямованого динамічного символьного виконання.

Розробка програмної реалізації інструменту гібридного аналізу програмного забезпечення

Інструмент гібридного аналізу програмного забезпечення повинен являти собою систему аналізу додатків режиму користувача реалізовану у вигляді модульної архітектури і побудовану на базі динамічної інструментації програм, символьних обчислень для представлення семантики операцій і вирішувачів формул в теоріях логіки першого порядку для цілісної арифметики, бітових векторів, масивів.

Модульна архітектура інструменту що повинна дозволяти вирішити дві системні задачі наведено на рис. 1.

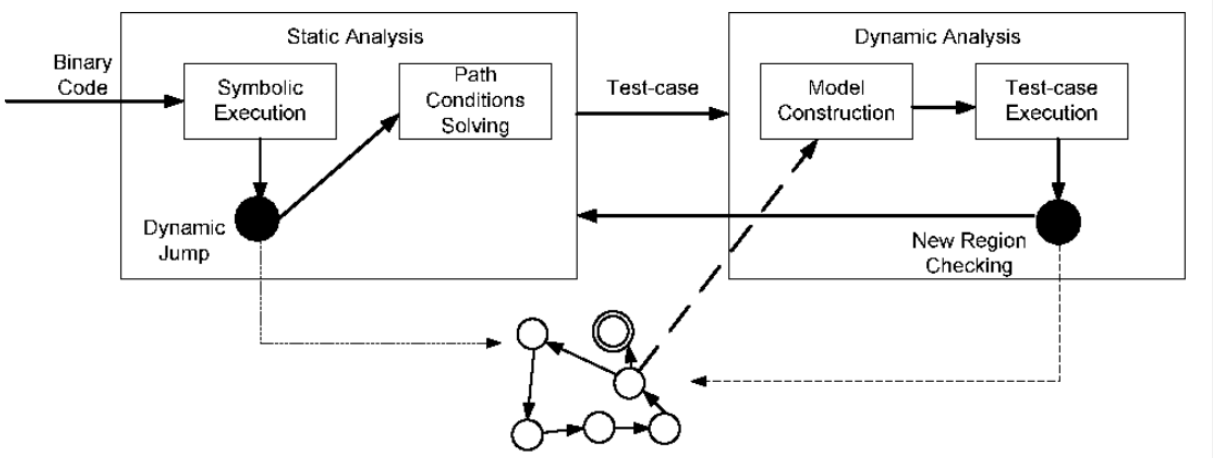

Рис. 1. Модульна схема гібридного аналізу програмного забезпечення

1. Абстракція вхідних та вихідних інтерфейсів кожного модуля дозволяє вирішити задачу заміни реалізації модулів, не змінюючи загальний алгоритм роботи інструменту:

- для вирішення завдання збору шляху виконання програм, що виконуються на різних операційних системах та апаратних платформах, достатньо реалізувати спеціалізацію модуля шляхвання без зміни інших модулів інструмента;

- для підключення нового інструменту розв'язання формул у теоріях достатньо реалізувати спеціалізацію модуля генерації набору даних для кожного інструмента, що

підтримується, рішення формул у теоріях без необхідності зміни алгоритмів роботи інших модулів;

- для реалізації можливості використання альтернативних алгоритмів вибору наступного шляху для аналізу достатньо реалізувати спеціалізацію модуля оцінки набору даних;

- для реалізації алгоритмів виявлення вразливостей достатньо реалізувати спеціалізацію модуля детектора помилок.

2. Незалежність алгоритмів роботи модулів один від одного дозволяє реалізувати роботу модулів у паралельному режимі та використовувати можливості розподілених обчислень шляхом виділення окремих обчислювальних шляхів або вузлів у хмарі для кожного модуля.

Створення повноцінної програми для гібридного аналізу потребує комбінування інструментів для статичного та динамічного аналізу коду, а також автоматизації цього процесу. Програму можна реалізувати за допомогою Python, оскільки він добре підходить для інтеграції з багатьма інструментами для аналізу коду, а також автоматизації через поєднання безперервної інтеграції ПЗ (CI) та безперервного розгортання ПЗ (CD) у процесі розробки.

Ці практики дозволяють командам розробників швидше, надійніше та з меншою кількістю вразливостей доставляти програмне забезпечення. Вони сприяють оптимізованому робочому процесу, що покращує якість коду та прискорює випуск оновлень.

Програма, яка реалізує базовий підхід до гібридного аналізу з використанням статичного аналізу через API платформи с открытым исходным кодом для непрерывного анализа и измерения качества программного кода SonarQube, та динамічного аналізу веб-додатка через OWASP ZAP API може бути інтегрована в процесі забезпечення послідовної й автоматизованої збірки, упаковування і тестування застосунків.

Вимоги:

1. SonarQube для статичного аналізу коду;
2. OWASP ZAP для динамічного аналізу веб-додатків;
3. Python 3 для реалізації автоматизації.

1. Конфігурація SonarQube та OWASP ZAP:

```
import requests
import json
import time
# Конфігурація SonarQube API
SONARQUBE_URL = 'http://localhost:9000'
SONARQUBE_TOKEN = 'your_sonarqube_token'
# Конфігурація OWASP ZAP API
ZAP_URL = 'http://localhost:8080'
ZAP_API_KEY = 'your_zap_api_key'
# Шляхи до проекту та URL для динамічного аналізу
PROJECT_PATH = '/path/to/your/project'
TARGET_URL = 'http://example.com';
```

2. Функція статичного аналізу через SonarQube:

```
def run_sonar_analysis(project_key):
  # Запуск аналізу проекту
  sonar_url = f"{SONARQUBE_URL}/api/project_analyses/search"
  headers = {
    'Authorization': f'Bearer {SONARQUBE_TOKEN}'
  }
    # Запит на аналіз проекту
  response = requests.get(sonar_url, headers=headers, params={"project": project_key})
    if response.status_code == 200:
```

```
        analysis_data = response.json()
        print("Статичний аналіз виконано успішно.")
        return analysis_data
    else:
        print(f"Помилка запуску статичного аналізу: {response.status_code}")
        return None;
```

3. Функція динамічного аналізу через OWASP ZAP:

```
def run_zap_scan(target_url):
    zap_url = f"{ZAP_URL} / JSON / ascan / action / scan /"
    params = {
        'url': target_url,
        'apikey': ZAP_API_KEY
    }
    # Запуск активного сканування
    response = requests.get(zap_url, params=params)
    if response.status_code == 200:
        scan_id = response.json()['scan']
        print(f"Запущено
        динамічне
        сканування(ID: {scan_id}).")
        # Очікуємо завершення сканування
        while True:
            status_url = f"{ZAP_URL} / JSON / ascan / view / status /?scanId = {scan_id} & apikey =
{ZAP_API_KEY}"
            status_response = requests.get(status_url)
            status = status_response.json()['status']
            print(f"Сканування
            триває: {status} %")
            if status == '100':
        print("Динамічне
        сканування
        завершено.")
        break
    time.sleep(10)
# Отримання звіту
report_url = f"{ZAP_URL} / JSON / core / view / alerts /?baseurl = {target_url} & apikey =
{ZAP_API_KEY}"
report_response = requests.get(report_url)
return report_response.json()['alerts']
else:
print(f"Помилка
запуску
динамічного
аналізу: {response.status_code}")
return None;
```

4. Об'єднання результатів статичного та динамічного аналізу:

```
def correlate_results(static_results, dynamic_results):
    print("Аналіз результатів...")
    # Формування висновків на основі обох аналізів
    if static_results:
```

```
    print(f”Результати статичного аналізу: {len(static_results['analyses'])} знайдених
вразливостей.”)
  else:
    print(“Статичний аналіз не дав результатів.”)
  if dynamic_results:
    print(f”Результати динамічного аналізу: {len(dynamic_results)} знайдених вразливостей.”)
  else:
    print(“Динамічний аналіз не дав результатів.”)
  # Кореляція даних та виведення фінальних висновків
  if static_results and dynamic_results:
    print(“Об'єднані результати показують більш комплексну картину безпеки.”)
  else:
    print(“Потрібно додаткове тестування для повної картини.”);
     5. Основна функція:
     def main():
  # Виконання статичного аналізу
  static_results = run_sonar_analysis('my_project_key')
    # Виконання динамічного аналізу
  dynamic_results = run_zap_scan(TARGET_URL)
    # Об'єднання результатів
  correlate_results(static_results, dynamic_results)
if __name__ == '__main__':
  main()
```

Опис роботи програми

1. Конфігурація та налаштування. Налаштовання SonarQube та OWASP ZAP для проведення аналізу. У коді це виконується за допомогою API.

2. Статичний аналіз. Програма звертається до SonarQube API для аналізу вихідного коду на наявність вразливостей, використовуючи ваш проект.

3. Динамічний аналіз. Використовуючи OWASP ZAP API, програма виконує динамічне тестування додатка, перевіряючи його поведінку під час виконання.

4. Кореляція результатів. Програма об'єднує результати статичного та динамічного аналізів, щоб надати комплексний звіт про вразливості. У якості метрики було обрано ціікломаатичну складність коду, що більш відповідає меті та сутності дослідження.

Такий підхід дозволяє контролювати потік позначених даних програми та знизити складність символьної формули шляхом виключення шляху даних програми, що не залежить від зовнішніх даних програми. З іншого боку, даний підхід призводить до ускладнення алгоритмів поширення поміченості по шляху даних програми і виникнення проблеми недостатньої поміченості шляху даних, що, в результаті, призводить до обриву шляху помічених даних, пов'язаному з наявністю в програмі непрямих залежностей шляху помічених даних з управління та даними. Даний метод дозволяє одночасно перевірити можливості інструментів для виявлення вразливостей у реальних програмах та перевірити програми, доступні для використання, на наявність вразливостей та вразливостей. На даний момент немає загальноприйнятого міжнародного стандарту для перевірки якості інструментів аналізу.

У зв'язку з цим розробники інструментів створюють свої та використовують доступні тестові набори для інструментів, а також стандартною практикою оцінки стала перевірка розроблених інструментів аналізу на наборі програм, доступних для використання.

Результати досліджень

Під час оцінки ефективності запропонованого методу захисту програмного забезпечення на основі гібридного аналізу коду було проведено дослідження на прикладі різноманітних утиліт командного рядка з відкритим вихідним кодом.

© Гапон А. О. (2025) Експериментальне дослідження, програмна реалізація та оцінка ефективності застосування методу захисту програмного забезпечення на основі гібридного аналізу. Сучасний захист інформації, 3(63), 27–36. https://doi.org/10.31673/2409-7292.2025.030422

Метою дослідження було оцінити здатність генерувати зовнішні дані для виявлення програмних вразливостей, таких як переповнення буфера, неправильне використання вказівників на звільнені блоки пам’яті, розіменування нульових покажчиків і витоки ресурсів. Аналіз проводився шляхом поєднання статичного аналізу програмного коду та динамічного символьного виконання з використанням програмного забезпечення та метрик, що були розроблені у попередніх роботах.

Дослідження підтверджує, що більшість попереджень, згенерованих статичним аналізом, не становлять реальних загроз. Це підкреслює необхідність вдосконалення комбінованих методів аналізу, які зменшать кількість хибних спрацьовувань і підвищать ефективність виявлення критичних помилок.

Зв'язок між типом вразливості та її підтвердженням показує, що деякі категорії, як-от витоки ресурсів і проблеми з вказівниками, мають більший ризик хибних попереджень порівняно з іншими.

Ці результати свідчать, що хоча статичний аналіз є важливим інструментом для виявлення можливих проблем у програмному коді, динамічний аналіз та комбіновані підходи є необхідними для підтвердження реальних загроз. Такий підхід дозволяє зменшити кількість хибних позитивних результатів і зосередитися на виправленні тих помилок, які становлять реальну небезпеку.

Обговорення результатів.

Результати експериментів було проаналізовано за допомогою розробленого програмного забезпечення.

Графік, що показує порівняння кількості попереджень про можливі вразливості та підтверджених реальних вразливостей для різних типів помилок: переповнення буфера, використання вказівників на звільнені блоки пам'яті, неправильне розіменування нульових покажчиків і витоки, наведено на рис. 2.

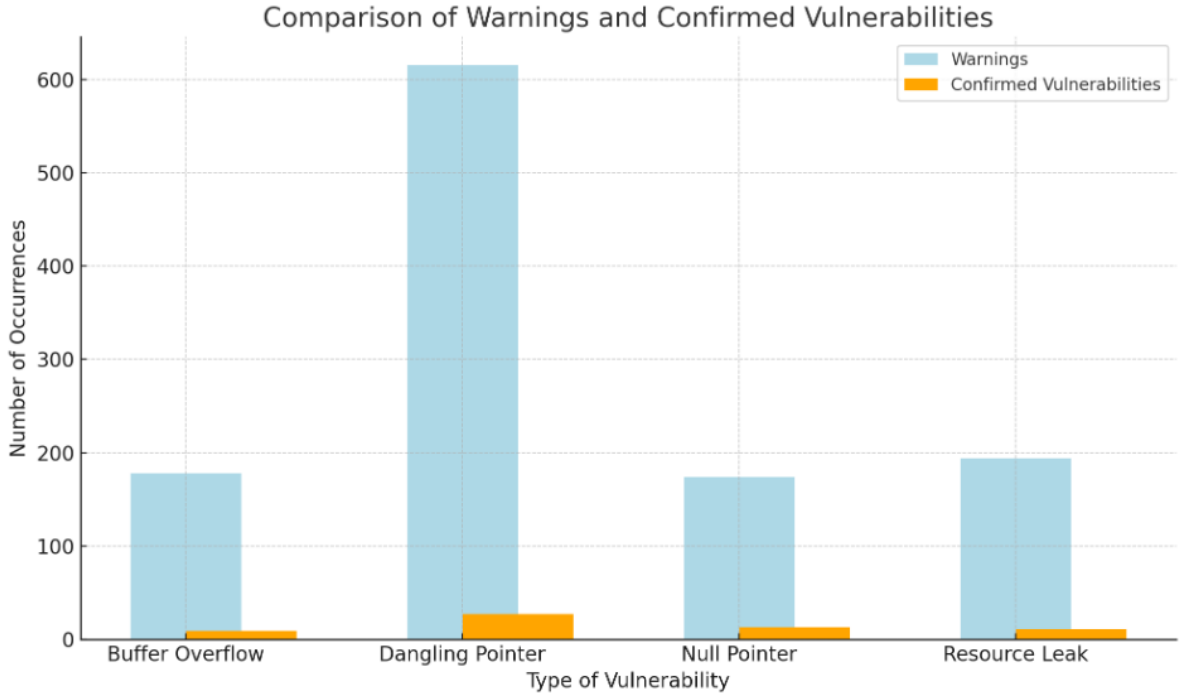

Рис. 2. Порівняння кількості попереджень про можливі вразливості та підтверджених реальних вразливостей

Як видно, кількість попереджень значно перевищує кількість підтверджених вразливостей, що свідчить про важливість гібридного аналізу для виявлення реальних загроз.

Виявлені помилки під час аналізу програмного забезпечення, такі як переповнення буфера, використання "завислих" покажчиків, розіменування нульових покажчиків та витоки ресурсів, є критичними вразливостями, які можуть стати потенційними точками входу для шкідливого програмного забезпечення.

Кожна з цих вразливостей відкриває можливість для атак на систему. Програми з такими помилками можуть стати цілями для шкідливих програм або атак зловмисників, оскільки експлуатація цих дефектів дозволяє їм отримати доступ до привілейованих ресурсів або навіть виконувати свій код на рівні ядра операційної системи. Тому такі вразливості є критичними точками входу для шкідливого програмного забезпечення, і їх усунення є обов'язковим кроком для забезпечення безпеки програмного коду.

Незважаючи на відомі обмеження інструменту динамічного символьного виконання, вдалося побудувати зовнішні дані програми для проходження по шляху подій попередження про помилку для декількох типів помилок, що підтверджує застосування розроблених алгоритмів для класифікації попереджень про вразливості запропонованим методом, підвищує точність результатів статичного аналізу програм і дозволяє використовувати побудовані зовнішні дані програми її налагодження. Також вдалося виявити кілька критичних вразливостей часу виконання програми, які не виявлені методами статичного аналізу програм.

Висновки

Проведений аналіз продемонстрував ефективність застосування статичного та динамічного аналізу для виявлення вразливостей у програмному забезпеченні.

Розроблено модуль зіставлення вектору подій попередження про вразливість. Впровадження модуля, що використовує абстрактне синтаксичне дерево, дозволяє значно скоротити кількість шляхів виконання, які потрібно досліджувати методом динамічного символьного виконання. Застосування міжпроцедурного аналізу та графу викликів забезпечило чітке відстеження шляхів виконання, що є критично важливим для виявлення та виправлення вразливостей. Також додаткові модулі, такі як модуль генерації запитів і оцінки шляхів, дозволили зменшити кількість помилок та підвищити якість кінцевого продукту. Зокрема, використання інформації про непрямі залежності надає нові можливості для покращення точності аналізу.

Представлено архітектуру інструменту гібридного аналізу. Модульна архітектура забезпечує гнучкість у реалізації різних компонентів інструменту, що дозволяє легко інтегрувати нові модулі або змінювати існуючі без значних змін у загальному алгоритмі роботи. Це підвищує надійність та ефективність аналізу, оскільки можливе одночасне використання різних алгоритмів та інструментів.

Запропоновано комплексний підхід до збору та аналізу даних. Всі етапи, починаючи від збору шляху виконання до оцінки результатів, свідчать про значний прогрес у використанні автоматизованих рішень для виявлення вразливостей. Автоматизація процесу, дозволяє швидше та ефективніше тестувати та впроваджувати програмне забезпечення.

Перспективи подальших досліджень. У розділі також підкреслено важливість подальшої розробки методів, що враховують непрямі залежності, а також вдосконалення алгоритмів динамічного символьного виконання для розв’язання складних завдань виявлення вразливостей у програмному забезпеченні.

Таким чином, реалізація та оцінка запропонованого методу підтверджує його ефективність у забезпеченні більш надійного захисту програмного забезпечення, зокрема через автоматизовані підходи до виявлення вразливостей та їх усунення. Результати проведених досліджень відкривають нові перспективи для подальшої оптимізації та розвитку методів гібридного аналізу. Таким чином, експериментальна перевірка запропонованого підходу підтвердила його ефективність у забезпеченні безпеки програмного забезпечення, зокрема завдяки автоматизованому виявленню та виправленню вразливостей. Отримані результати створюють основу для подальшого вдосконалення методів гібридного аналізу.

© Гапон А. О. (2025) Експериментальне дослідження, програмна реалізація та оцінка ефективності застосування методу захисту програмного забезпечення на основі гібридного аналізу. Сучасний захист інформації, 3(63), 27–36. https://doi.org/10.31673/2409-7292.2025.030422

Перелік посилань

1. Захисний комплекс Microsoft / Що таке шкідливе програмне забезпечення? https://www.microsoft.com/uk-ua/security/business/security-101/what-is-malware.

2. Python Type Checking. URL: https://testdriven.io/blog/python-typechecking/ (дата звернення 16.04.2024).

3. Delmas, D. (2022). Static analysis of program portability by abstract interpretation (Doctoral dissertation). Sorbonne Université.

4. Generating and using a Callgraph, in Python. URL: https://cerfacs.fr/coop/pycallgraph (дата звернення 16.04.2024).

5. Data Flow Analysis. URL: https://www.codingninjas.com/studio/library/data-flow-analysis (дата звернення 16.04.2024).

6. Python Control Flow Statements and Loops. URL: https://pynative.com/python-control-flow-statements/ (дата звернення 16.04.2024).

7. Akhtar, M. S., & Feng, T. (2022). Malware analysis and detection using machine learning algorithms. Symmetry, 14(11), 2304. URL: https://doi.org/10.3390/sym14112304 (дата звернення 16.04.2024).

8. Monat, R., Ouadjaout, A., Miné, A. (2021). A Multilanguage Static Analysis of Python Programs with Native C Extensions. In: Drăgoi, C., Mukherjee, S., Namjoshi, K. Static Analysis. SAS 2021. Lecture Notes in Computer Science(), vol 12913. Springer, Cham. URL: https://doi.org/10.1007/978-3-030-88806-0_16.

9. Infographic Open source linters, tools for code analysis 2021. URL: https://www.promyze.com/open-source-linters-2021/ (дата звернення 16.04.2024).

10. Vassallo, C., Panichella, S., Palomba, F., et al. (2020). How developers engage with static analysis tools in different contexts. Empirical Software Engineering, 25, 1419-1457.

11. B. Chess and G. McGraw, “Static analysis for security,” in IEEE Security & Privacy, vol. 2, no. 6, pp. 76-79, Nov.-Dec. 2004, doi: 10.1109/MSP.2004.111.

12. Лаптєв, О. А., Колесник, В. В., Ровда, В. В., & Половінкін, М. І. Метод підвищення захисту особистих даних за рахунок синтезу резильєнтних віртуальних спільнот. 2024. Сучасний захист інформації. 4(60). С. 141-146. https://doi.org/10.31673/2409-7292.2024.040015.

13. Лаптєв О.А., Марченко В.В. Застосування завад для захисту інформації від витоку радіоканалом. Сучасний захист інформації. 2025. №1. С.89-97. https://doi.org/10.31673/2409-7292.2025.013057.

14. Дробик О. В., Лаптєв О. А., Пархоменко І. І., Богуславська О. В., Пепа Ю. В., Пономаренко В. В. Розпізнавання радіосигналів на основі апроксимації спектральної функції у базисі передатних функцій резонансних ланок другого порядку. Сучасний захист інформації. 2024. №2. С.13-23. https://doi.org/10.31673/2409-7292.2024.020002.

15. Аль-Дальваш А., Петченко М.В., Лаптєв О.А. Метод детектування цифрових радіосигналів за допомогою диференціального перетворення. Сучасний захист інформації. 2025. №1. С.285-291. https://doi.org/10.31673/2409-7292.2025.014329.

Надійшла 11.06.2025