КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 1 (29), 2025
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2025.29.871
УДК 004.056

Лаптєв Олександр Анатолійович
д.т.н., с.н.с., доцент кафедри кібербезпеки та захисту інформації
Київський національний університет імені Тараса Шевченка, Київ, Україна
ORCID ID: 0000-0002-4194-402X
olaptiev@knu.ua

Гапон Андрій Олександрович
аспірант, кафедра безпеки інформаційних технологій
Харківський національний університет радіоелектроніки, Харків, Україна
ORCID ID: 0000-0003-2560-7426
andrii.hapon@nure.ua

Ткачов Андрій Михайлович
к.т.н., с.н.с. , доцент кафедри кібербезпеки
Національний технічний університет «Харківський політехнічний інститут», Харків, Україна
ORCID ID: 0000-0003-1428-0173
andrii.tkachov@khpi.edu.ua

МЕТОД ЗАХИСТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ ГІБРИДНОГО АНАЛІЗУ КОДУ

Анотація. У статті розглянуто актуальні питання захисту програмного забезпечення від шкідливого коду та виявлення його проявів у процесі розробки та експлуатації. Зазначено, що сучасні методи аналізу програмного забезпечення, зокрема статичний та динамічний аналіз, мають як переваги, так і суттєві обмеження, зокрема велику кількість хибних спрацювань, низьку ефективність проти поліморфних загроз та високі вимоги до обчислювальних ресурсів. В якості оптимального рішення запропоновано використання гібридного аналізу, який поєднує переваги різних підходів для підвищення точності виявлення вразливостей та зменшення кількості помилкових результатів. У роботі представлено математичну модель виявлення вразливостей на основі символьного виконання програми та комбінованого аналізу коду, а також розроблено алгоритми для побудови скороченого графа шляху програми, обчислення метрик відстані до потенційно небезпечних ділянок коду та реалізації спрямованого динамічного символьного виконання. Методика класифікації попереджень про вразливості передбачає поділ їх на три категорії: підтверджені, непідтверджені та потребуючі додаткової інспекції. Такий підхід дозволяє значно знизити трудомісткість аналізу, підвищити достовірність результатів та автоматизувати процес виявлення потенційно небезпечного коду. Особливої уваги приділено формалізації понять, пов’язаних із обмеженнями виконання шляху програми, символьними умовами та предикатами безпеки. Отримані результати демонструють ефективність гібридного аналізу при роботі з великими проектами, де важливими є як швидкодія, так і точність виявлення загроз. Розглянуто можливості модульної архітектури інструменту гібридного аналізу, що забезпечує гнучкість у розширенні функціоналу та інтеграції нових методів. Проведено аналіз ключових метрик вразливостей програмного коду, які можуть бути використані для оцінки безпеки програмного забезпечення. Запропоновано напрями подальших досліджень, зокрема удосконалення алгоритмів символьного виконання для врахування непрямих залежностей та антианалізних механізмів. Результати дослідження можуть бути використані при розробці нових та модернізації існуючих систем аналізу коду, спрямованих на підвищення рівня безпеки програмного забезпечення.

Ключові слова: шкідливий код; пошук сигнатур; виявлення шкідливого програмного забезпечення; класифікація програм; метрики коду; гібридний аналіз; штучні нейронні мережі; машинне навчання.

ВСТУП

Шкідливе програмне забезпечення — це зловмисна програма або код, які шкодять кінцевим пристроям. Якщо пристрій уражено шкідливим програмним забезпеченням, може відбуватися несанкціонований доступ, ураження даних або блокування пристрою, поки ви не сплатите викуп.

Люди, які поширюють шкідливе програмне забезпечення, або кіберзлочинці, умотивовані фінансово та використовуватимуть уражені пристрої для запуску атак, щоб, наприклад, отримувати банківські облікові дані, збирати й збувати персональні дані, продавати доступ до обчислювальних ресурсів або вимагати від жертв платіжну інформацію [1].

Безпека коду в контексті шкідливого програмного забезпечення (ШПЗ) означає заходи, які призначені для запобігання та захисту від вразливостей, вразливостей та шкідливих атак, які можуть бути використані для недозволеного доступу, пошкодження даних або порушення функціонування програмного продукту або системи в цілому.

Основними принципами безпеки коду з боку захисту від ШПЗ, є:

Виявлення та усунення вразливостей: Це включає перевірку коду на наявність вразливостей, таких як некоректна обробка введених даних, можливості переповнення буфера, недостатня перевірка автентичності тощо. Після виявлення вразливостей потрібно усунути їх шляхом внесення відповідних змін у код.

Мінімізація поверхні атаки: Цей принцип передбачає зменшення кількості місць у коді, які можуть бути використані для атак. Наприклад, це може включати використання обмежень доступу до файлів, відсутність зайвих прав користувача у виконанні програми тощо.

Використання безпечних практик програмування: До цього входить використання безпечних функцій та методів програмування, які запобігають вразливостям, таких як коректна обробка виключень, використання безпечних структур даних, уникання використання застарілих функцій з відомими вразливостями тощо.

Стійкість до атак: Код повинен бути написаний так, щоб він залишався функціональним і надійним навіть після спроб атаки. Наприклад, він повинен здійснювати коректну обробку некоректні вхідних даних та запобігати можливості витоку конфіденційної інформації.

Постійне оновлення та підтримка: Код програми повинен регулярно оновлюватися та підтримуватися для виправлення виявлених вразливостей та адаптації до нових методів атак.

Отже, безпека коду у контексті ШПЗ полягає в комплексному підході до захисту від потенційних загроз та мінімізації можливостей атак на програмний продукт.

Постановка проблеми. У сучасних умовах розвитку інформаційних технологій та зростання кіберзагроз актуальним є забезпечення високого рівня безпеки програмного забезпечення на всіх етапах його життєвого циклу. Особливу небезпеку становлять шкідливі програми, які можуть призводити до порушення конфіденційності, цілісності та доступності даних, а також до збоїв у роботі критично важливих систем. Ефективне виявлення та протидія таким загрозам вимагає застосування сучасних методів аналізу коду, здатних забезпечити високу точність виявлення вразливостей із мінімальною кількістю хибнопозитивних результатів.

На сьогодні найбільш поширеними є статичний та динамічний методи аналізу програмного забезпечення. Статичний аналіз забезпечує швидке виявлення потенційно небезпечних фрагментів коду без його виконання, однак має обмежену точність, особливо стосовно поліморфних та невідомих типів шкідливого коду. Динамічний

аналіз, хоча й дозволяє отримати більш точні результати через реальне виконання програми, характеризується високими вимогами до обчислювальних ресурсів і не завжди ефективний у разі використання зловмисниками антианалізних механізмів. Крім того, обидва підходи мають обмежену здатність адаптуватися до складних і швидко змінюваних архітектур програмних систем.

Важливим аспектом є необхідність оптимізації процесу аналізу для масштабних проектів, де великий обсяг коду потребує значних часових і обчислювальних витрат. Це зумовлює актуальність пошуку нових підходів, які б поєднували переваги різних методів аналізу, одночасно зменшуючи їхні недоліки. Перспективним напрямом у цьому контексті є гібридний аналіз, що інтегрує статичний аналіз із символьним виконанням програм, що дає змогу підвищити точність виявлення вразливостей, скоротити кількість помилкових спрацювань і забезпечити ефективне тестування коду в реальних умовах.

Таким чином, науково-прикладною проблемою є розробка методу захисту програмного забезпечення, заснованого на гібридному аналізі коду, який забезпечить підвищення ефективності виявлення шкідливого ПЗ, враховуючи поліморфізм коду, складність сучасних архітектур та обмеження існуючих методів аналізу.

Аналіз останніх досліджень і публікацій. 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], [14], [19], 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], [15], [17], 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]. Такий підхід дає змогу зменшити кількість хибних спрацювань, підвищити точність виявлення загроз і краще адаптуватися до складних архітектур програм.

Гібридний аналіз особливо актуальний для:

1. Підвищення рівня безпеки.
2. Зменшення кількості помилкових спрацювань.
3. Адаптації до складних та змінних архітектур.
4. Застосування у великих проектах.
5. Підвищення ефективності розробки.

Таким чином, дослідження та вдосконалення гібридних методів виявлення шкідливого програмного забезпечення є перспективним напрямом у сфері захисту програмних продуктів, що базується на інтеграції сучасних аналітичних та штучно-інтелектуальних технологій.

Метою даної статті є розробка методу захисту програмного забезпечення, заснованого на математичних моделях, визначенні ступеня поліморфізму програмного коду відносно баз шкідливого ПЗ та інтелектуальному аналізі шляхів виконання програм у реальному часі. Це дозволить виявляти та запобігати невідомим загрозам безпеки у програмних продуктах.

Для досягнення цієї мети необхідно вирішити низку завдань. По-перше, передбачається проведення аналізу методів статичного та динамічного виявлення ШПЗ. По-друге, потрібно визначити метрики коду для ідентифікації ШПЗ та обґрунтувати гібридний підхід до аналізу коду для захисту ПЗ. На основі цього буде розроблено вдосконалений метод захисту програмного забезпечення з використанням гібридного аналізу коду, що включає побудову програмної моделі розробленого методу та формування тестового набору для його верифікації. Завершальними етапами є проведення експерименту з дослідженням верифікації методу, отримання результатів та їх інтерпретація.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Основу запропонованого в роботі методу класифікації попереджень про потенційні вразливості програмного забезпечення складає модель виявлення вразливостей під час символьного виконання програми та алгоритм комбінованого аналізу коду [4], де кортеж операцій що складають програму P визначено як четвірку

$$P = \left\langle F, S, S_{begin}, S_{end} \right\rangle \quad (1)$$

де: S — безліч станів програми; S_{begin} — початковий стан програми; S_{end} — безліч кінцевих станів програми; F — безліч операцій, кожна з яких перекладає програму з одного стану в інше

$$f : S \to S, f \in F, \quad (2)$$

У такому випадку хід виконання програми можна визначити як послідовність переходів між станами програми $S_i \in S$, здійснюваних операціями $F_i \in F$

$$\left\{ f_{start}(S_{start}) \to S_1, f_1(S_1) \to S_2, \ldots, f_i(S_i) \to S_{end} \right\} \quad (3)$$

де: $S_{start} \in S$ — початковий стан програми; $S_{end} \in S$ — одне з кінцевих станів програми; $F_i \in F$ — безліч операцій програми, що перекладають один стан програми до іншого.

Стан програми визначено як,

$$S = \left\langle d, f \right\rangle | d \in D, f \in F \quad (4)$$

де d — підмножина множини даних, оброблюваних програмою; D — множина даних, оброблюваних програмою; f — наступна операція програми. Тоді виконання операції програми можна подати у вигляді

$$f_i(d_i) \to \left\langle d_i, f_i \right\rangle | d_i, d_j \in D, f_i, f_j \in F \quad (5)$$

Обмеженням шляху виконання у програмі Q це безліч обмежень на значення даних програми, отримане шляхом перетворення операцій, виконаних над даними програми на шляху виконання, елементи безлічі обмежень і однозначно описує виконання програми шляхом, що досягає стану S.

Стан вразливості це такий стан програми $S_{err} \in S$, при якому подальше виконання програми помилково. Безліч визначення операції f — це таке підмножина даних програми $D' \subseteq D$, на якому виконання операції f не призводить до досягнення стану вразливості . Для кожної операції f, що входить до множини операції програми F, існує безліч визначення даної операції D', і якщо множина D' не порожня, то виконання операції наводить програму в стан вразливості

$$f(d) \to S_{err} \mid f \in F, \forall d \notin D^I \wedge D^I \neq \varnothing \quad (6)$$

Можна сверджувати, що якщо множина визначення операції порожня, то виконання операції не залежить від даних програми, що відповідає визначенню операції на всій безлічі даних програми, і, щоб обчислити зовнішні дані програми, що призводять до виконання програми в стан вразливості, необхідно і достатньо для кожного типу вразливості визначити множину операцій, виконання яких може призводити до вразливості, та сформулювати умова виходу значень аргументів операції за межі множини визначення цих операцій і при цьому входять до безлічі значень даних програми.

Метрики вразливості програмного коду є важливим інструментом для кількісної оцінки безпеки програмного забезпечення. Вони дозволяють виявляти потенційні загрози в коді ще на етапі розробки та забезпечують можливість впровадження ефективних заходів для їх усунення. Метрики допомагають оцінити, наскільки програмний код вразливий до атак, яка частина коду потребує додаткової уваги, і які аспекти розробки слід покращити для зниження ризику [12] – [16].

Основні метрики вразливості програмного коду можна розділити на кілька категорій [12] – [16]:

1. Метрики кількості вразливостей
2. Метрики складності коду та архітектури
3. Метрики покриття тестами
4. Метрики динамічного аналізу
5. Метрики використання небезпечних API
6. Метрики оцінки вразливостей на основі репутації та історії

Нижче наведені основні формули для розрахунку метрик вразливості програмного коду [12] – [16]. Ці метрики використовуються для кількісного оцінювання безпеки програмного забезпечення та для вимірювання різних аспектів вразливості коду.

1. Кількість вразливостей на тисячу рядків коду ($KLOC$)

Ця метрика показує кількість вразливостей на 1000 рядків коду.

$$KLOC = \frac{Кількість\ вразлівостей}{Загальна\ кількість\ рядків\ коду} 1000 \tag{7}$$

де:

кількість вразливостей — загальна кількість знайдених вразливостей;

загальна кількість рядків коду — кількість рядків у програмному коді.

2. Цикломатична складність (McCabe's Cyclomatic Complexity)

Ця метрика вимірює кількість незалежних шляхів виконання в програмному коді і може використовуватися для оцінки складності та потенційної вразливості коду.

$$V(G) = E - N + 2P \tag{8}$$

де: E — кількість ребер (переходів між блоками) у графі шляху програми;

N — кількість вершин (блоків) у графі шляху програми;

P — кількість компонентів або функціональних частин програми;

3. Коефіцієнт підтримуваності (Maintainability Index, MI)

Цей індекс оцінює, наскільки легко підтримувати код, і враховує кілька факторів, включаючи цикломатичну складність, кількість рядків коду та кількість коментарів.

$$MI = 171 - 5.2 \times \ln(LOC) - 0.23 \times CC - 16.2 \times \ln(HV) + \\ +50 \times \sin(\sqrt{2.46 \times C}) \tag{9}$$

де: LOC — кількість рядків коду;

CC — цикломатична складність;

HV — об'єм Гальстеда (Halstead Volume), який обчислюється за формулою

$$HV = N \times \log 2(\eta) \quad (10)$$

де: N — загальна кількість операторів та операндів;

η — кількість унікальних операторів та операндів;

C — відсоток рядків коду, що є коментарями.

4. Покриття тестами (Test Coverage)

Ця метрика вимірює відсоток коду, який перевіряється автоматизованими тестами

$$Test\ Coverage = \frac{Кількість\ протестованих\ рядків\ коду}{Загальна\ кількість\ рядків\ коду} 100\% \quad (11)$$

де: Кількість протестованих рядків коду — кількість рядків коду, які були покриті тестами;

Загальна кількість рядків коду — кількість рядків у програмному коді.

5. Кількість викликів небезпечних API

Метрика підраховує використання небезпечних функцій або бібліотек у коді

$$Dangerous\ API\ sage = \frac{Кількість\ небезпечних\ викликів}{Загальна\ кількість\ викликів} 100\% \quad (12)$$

де: Кількість небезпечних викликів — кількість викликів відомих небезпечних функцій;

Загальна кількість викликів API — загальна кількість усіх викликів API у програмі.

6. Час виконання вразливого коду (Vulnerable Code Runtime (VCR))

Ця метрика вимірює, який відсоток часу виконання програми витрачається на виконання вразливих частин коду

$$VCR = \frac{Час\ виконання\ вразливих\ частин\ коду}{загальнийй\ час\ виконання\ програми} 100\% \quad (13)$$

де: Час виконання вразливих частин коду — сумарний час виконання тих частин програми, які містять потенційні вразливості;

Загальний час виконання програми — загальний час виконання всіх частин програми.

7. Індекс вразливості бібліотек (LVI)

Ця метрика оцінює рівень вразливості зовнішніх бібліотек, які використовуються у проекті

$$Library Vulnerability\ Index = \frac{Кількість\ відомих\ вразливостей\ у\ бібліотеках}{Загальна\ кількість\ використаних\ бібіліотек} 100\% \quad (14)$$

де: Кількість відомих вразливостей у бібліотеках — кількість вразливостей, виявлених у бібліотеках на основі публічних баз даних;

Загальна кількість використаних бібліотек — загальна кількість зовнішніх бібліотек, які використовуються у програмному проекті.

Застосування метрик вразливості дозволяє комплексно оцінити якість і безпеку програмного коду, своєчасно виявляти слабкі місця та підвищувати стійкість до атак. Вони сприяють глибшому розумінню архітектури системи, мінімізують ризики експлуатації вразливостей та несанкціонованого доступу. Регулярне використання таких метрик забезпечує об’єктивне оцінювання безпеки на різних етапах розробки, дозволяє порівнювати версії коду та спрямовувати зусилля команди на найбільш критичні

ділянки. Це підвищує ефективність роботи розробників і сприяє створенню надійного програмного забезпечення.

Для реалізації методу захисту програмного забезпечення на основі гібридного аналізу коду вирішено декілька завдань, розроблено набір алгоритмів, що дозволяють досягти мети класифікації попереджень про програмні вразливості .

Загальний алгоритм, що лежить в основі запропонованого методу, складається з кількох етапів.

Загальний алгоритм комбінованого аналізу:

1. Виявлення потенційних вразливостей у програмі методами статичного аналізу.
2. Статичний аналіз виконуваного коду програми для отримання скороченого графа шляху програми з метою обчислення шляхів, подій, що містять шлях для кожної знайденої за допомогою статичного аналізу вразливості .
3. Обчислення відстані кожного базового блоку програми, лежить на коліях, що увійшли до скороченого графа шляху програми, до точки на шляху подій для кожної вразливості , знайдену за допомогою статичного аналізу.
4. Реалізація спрямованого динамічного символьного виконання програми з метою генерації зовнішніх даних, що призводять до виконання програми по шляхах, що досягають шляху вразливості .
5. Генерація зовнішніх даних програми, що порушують предикат безпеки у точці реалізації вразливості.
6. Перевірка прояву вразливості у процесі виконання програми на зовнішніх даних, що згенеровані на етапі 5.

Реалізація методів статичного аналізу програм виходить за рамки даної роботи, у зв’язку з чим інформація про вразливості, знайдені методами статичного аналізу, буде використовуватися як вихідні дані для описаного методу.

У випадку, якщо шлях вразливості отримана від статичного інструменту аналізу вихідного коду програм, потрібно вирішити завдання зіставлення шлях виконання програми у вихідному тексті мовою високого рівня та у машинному коді.

Алгоритм побудови скороченого міжпроцедурного графа шляху програми:

1. Спочатку будується скорочений граф викликів підпрограми. Побудова починається з підпрограм, у яких знаходяться точки прояви вразливості у програмі. Далі граф доповнюється підпрограми, що викликають підпрограми, що містять точки прояви вразливості у програмі. Побудова графа продовжується до до тих пір, поки не буде досягнуто підпрограма, що містить точку входу в програму чи головну функцію обчислювального шляху.

Таким чином, з машинного коду програми вилучається скорочений граф викликів, що містить лише ті підпрограми, виконання яких необхідне прояви помилок, попередження про які отримані від статичного інструменту аналізу програм.

2. Далі для підпрограм, що увійшли до скороченого графа викликів, виробляється статичне вилучення скороченого міжпроцедурного графа шляху, що містить лише ті шляхи виконання, які включають шлях попередження про помилку, отриманої від інструменту статичного аналізу програм.

3. Після отримання скороченого міжпроцедурного графу шляху програми проводиться обчислення числової метрики відстані, що виражається у кількості умовних переходів від кожного базового блоку програми до кожної точки шляху подій попередження про помилку у програмі. Надалі відстань від базового блоку до точки у

програмі використовується як чисельна оцінка при виборі наступного шляху виконання для аналізу: спочатку вибираються шляхи, котрим ця оцінка менше.

За результатами виконання алгоритму будуються структури даних, необхідні для проведення спрямованого аналізу у процесі динамічного символьного виконання програми.

Для обчислення набору зовнішніх даних програми, що призводять до виконанню шляхом, що містить шлях попередження про помилку, отриману від інструменту статичного аналізу, використовується метод спрямованого ітеративного динамічного символьного виконання програми.

Алгоритм спрямованого динамічного символьного виконання:

1. Виробляється топологічне сортування точок подій на шляху попередження про помилку на графі шляху програми.
2. На першій ітерації аналізу програма запускається виконання з довільним набором зовнішніх даних.
3. Проводиться збір шляху виконання програми та перетворення їх у символьну формулу, у якій зовнішні дані програми представлені вільними (символьними) змінними.
4. Якщо шлях виконання у програмі не пройшов шляхом подій попередження про помилку в програмі, то проводиться інвертування умови для умовного переходу, що знаходиться в базовому блоці, з найменшою оцінкою відстані до наступної точки на шляху подій попередження про помилку у програмі.
5. Обчислюється новий набір зовнішніх даних програми з метою проведення виконання програми по новому шляху.
6. Здійснюється запуск програми на новому наборі зовнішніх даних та виконання алгоритму триває з етапу 3 доти, доки наступна точка на шляху подій попередження про помилку буде досягнуто або не залишиться шляхів, що досягають наступного точку на шляху попередження про помилку.
7. Робота алгоритму завершується при досягненні останньої точки шляху подій попередження про помилку — точки прояву вразливості.

Рис. 1 ілюструє роботу алгоритму. Сірим кольором позначені базові блоки по дорозі виконання програми на поточної ітерації. Чорним кольором позначений базовий блок, що містить операцію на шляху подій попередження про помилку у програмі.

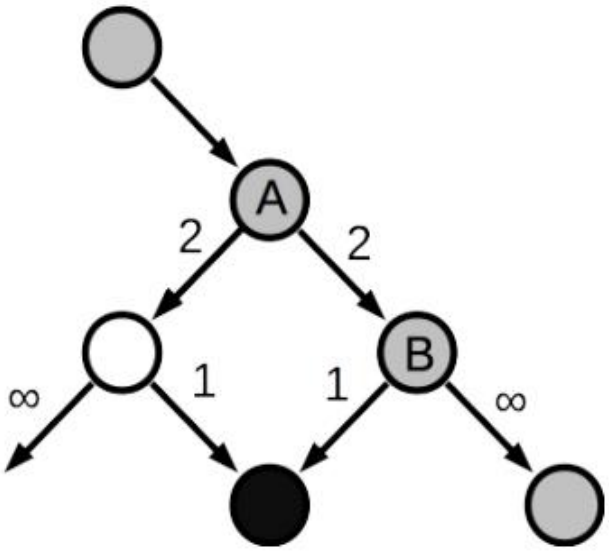

Рис. 1. Схема алгоритму спрямованого аналізу

Відстань від блоку А до цікавого базового блоку дорівнює двом переходам. Від блоку В — одному переходу. Відповідно спочатку для інвертування буде обрано умовний перехід у базовому блоці В, та був — у базовому блоці А.

РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ

Використання алгортиму спрямованого динамічного символьного виконання дозволяє значно зменшити вплив проблеми експоненційного зростання кількості шляхів для аналізу через обмеження кількості досліджуваних шляхів виконання у програмі набором шляхів, що приводять виконання в точку прояву вразливості.

Після того, як побудований набір зовнішніх даних, на якому відбувається виконання програми по вектору, що містить шлях подій попередження про помилку, провадиться доповнення предикату шляху умовою порушення предикату безпеки операції. Якщо умови в предикате шляхи та інвертований предикат безпеки спільні, то обчислюються зовнішні дані програми, що призводять до порушення предикату безпеки операції та, отже, до прояву вразливості. Якщо зовнішні дані програми, на яких виявляється помилка в програмі, побудовані, це означає підтвердження вразливості в програмі.

Метод класифікації побудований на базі моделі виявлення вразливостей у програмі та алгоритми спрямованого динамічного символьного виконання, що дозволяють обчислити зовнішні дані програми, на яких відбувається прояв вразливості у програмі.

Метод класифікації попереджень про вразливості інструменту статичного аналізу призначений для поділу всіх попереджень про вразливості на три класи, що не перетинаються:

1. Підтверджені попередження: для їх підтвердження вдалося побудувати зовнішні дані.
2. Непідтверджені попередження: їм вдалося показати несумісність умов на шляху подій.
3. Попередження, для яких не вдалося підібрати зовнішні дані програми, які проводять виконання по шляху попередження про помилку, і не вдалося довести несумісність умов на шляху подій.

Обговорення результатів. Для попереджень про вразливості, що потрапили до першого класу, обчислено набір зовнішніх даних програми, при подачі яких на вхід до програми вдається виявити помилку у програмі. Обчислений набір зовнішніх даних можна використовувати для налагодження програми та для навчання інструментів класифікації та ранжирування, заснованих на алгоритмах машинного навчання. Програмні вразливості, попередження про які потрапили до першого клас вимагають виправлення.

Попередження про вразливості другого класу, для яких вдалося довести методами розв’язання формул у теоріях несумісність умов на шляху подій у програмі від точки ініціалізації помилкової ситуації до точки прояву вразливості, є явними помилками першого роду (хибнопозитивними твердженнями про помилку). Такі попередження можуть бути використані для налагодження статичного аналізатора та як вибірка для навчання інструментів класифікації та ранжування, заснованих на алгоритмах машинного навчання.

Для попереджень про вразливості, що потрапили до третього класу, потрібно інспекція попереджень спеціалістом. Ця класифікація дозволяє обмежити безліч попереджень про вразливості статичного аналізатора, які необхідні дослідити фахівця, і, отже, знижує трудовитрати висококваліфікованого спеціаліста.

Запропоновані алгоритми та метод не залежать від реалізації методів статичного аналізу програм, що дозволяє застосовувати їх спільно з інструментами статичного аналізу програм для вихідного тексту програми, так і для виконуваного програмного коду.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Аналіз сучасних методів виявлення вразливостей у програмному забезпеченні показав, що найбільш ефективним підходом є гібридний аналіз, який поєднує статичний та динамічний методи. Такий підхід дозволяє зменшити кількість хибно позитивних результатів, характерних для статичного аналізу, і водночас підвищити продуктивність порівняно з динамічним аналізом. Основна перевага гібридного аналізу полягає в інтеграції символьного виконання зі статичним аналізом, що забезпечує більш точне виявлення потенційно небезпечних фрагментів коду.

Робота передбачає удосконалення математичної формалізації понять, пов’язаних із моделюванням обмежень виконання шляхів програми, символьних змінних та механізмів виявлення помилок. Це дало змогу розробити новий метод комбінованого аналізу, спрямований на підвищення точності виявлення вразливостей без значного зниження продуктивності.

Запропонована архітектура інструменту гібридного аналізу має модульну структуру, що забезпечує гнучкість у реалізації та можливість подальшого розширення системи за рахунок інтеграції нових компонентів. Також приділено увагу оптимізації використання обчислювальних ресурсів, що є ключовим фактором при масштабуванні технології для великих проектів.

Експериментальні дослідження підтвердили ефективність запропонованого методу в умовах реального виконання програм. Виявлено, що автоматизація процесів збору, аналізу та оцінки даних значно прискорює тестування та підвищує загальну якість захисту програмного забезпечення. Особливе значення надається усуненню вразливостей, оскільки саме через них зловмисники отримують доступ до критичних ресурсів системи.

У подальших дослідженнях планується розвивати алгоритми динамічного символьного виконання, особливо для виявлення вразливостей, що включають непрямі залежності між змінними, а також оптимізація методів аналізу для більш складних і розподілених систем. Реалізація цих напрямів забезпечить нові можливості для підвищення рівня безпеки програмного забезпечення в умовах сучасної цифрової інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Захисний комплекс Microsoft / Що таке шкідливе програмне забезпечення? https://www.microsoft.com/uk-ua/security/business/security-101/what-is-malware.
2. Static and Symbolic Analysis. URL: https://www.talkcrypto.org/blog /2019/03/15/static-and-symbolic-analysis/ .
3. Python Type Checking. URL: https://testdriven.io/blog/python-typechecking/.
4. Delmas, D. (2022). Static analysis of program portability by abstract interpretation (Doctoral dissertation). Sorbonne Université.
5. Generating and using a Callgraph, in Python. URL: https://cerfacs.fr/coop/pycallgraph.
6. Data Flow Analysis. URL: https://www.codingninjas.com/studio/library/data-flow-analysis.
7. Python Control Flow Statements and Loops. URL: https://pynative.com/python-control-flow-statements/ .
8. 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.

9. 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.
10. 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.
11. Rami Sihwail,Khairuddin Omar and Khairul Akram Zainol Ariffin,”A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis,” International Journal on Advanced Science, Engineering and Information Technology, vol. 8, no. 4-2, pp. 16621671, (2018).
12. Tahir, R. A study on malware and malware detection techniques. International Journal Education and Management Engineering (IJEME) 8(2), 20–30 (2018)
13. Chowdhury, I., & Zulkernine, M. (2011).Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. Journal of Systems Architecture, 57(3), 294–313.
14. Shin, Y., & Williams, L. (2008). Can traditional fault prediction models be used for vulnerability prediction?Empirical Software Engineering, 13, 497–530.
15. Zimmermann, T., Nagappan, N., Gall, H., Giger, E., & Murphy, B. (2010).Cross-project defect prediction: A large scale experiment on data vs. domain vs. process.
16. Scandariato, R., Walden, J., Hovsepyan, A., & Joosen, W. (2014).Predicting vulnerable software components via text mining.IEEE Transactions on Software Engineering, 40(10), 993–1006.
17. Лаптєв О., Зозуля С. Метод виключення відомих сигналів при скануванні заданого радіодіапазону. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка». 2023. Том 2 № 22. С. 31–38. https://doi.org/10.28925/2663-4023.2023.22.3138
18. Олександр Лаптєв, Віталій Савченко, Алла Кобозева, Анатолій Салій, Тимур Куртсеітов. Методи оцінки захищеності інформації в мережах зв’язку. Кібербезпека: освіта, наука, техніка. 2025. Том 3. № 27. С. 522-533. https://doi.org/10.28925/2663-4023.2025.27.767
19. О.Лаптєв, Т. Лаптєва, М. Браіловський. Методики розрахунків параметрів виявлення сигналів засобів негласного отримання інформації (джерел радіовипромінювання. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка». 2025. Том 4 № 28. 2025. С. 575–585. https://doi.org/10.28925/2663-4023.2025.28.812
20. O.Laptiev, N.Lukova-Chuiko, S.Laptiev, T.Laptieva, V.Savchenko, S.Yevseiev. Development of a method for detecting deviations in the nature of traffic from the elements of the communication network. International Scientific And Practical Conference “Information Security And Information Technologies”: Conference Proceedings. 13-19 September 2021. Kharkiv – Odesa, Ukraine. P.8-17, ISBN 978-966-676-818-9.
21. Kostiuk, Yu. V., Skladannyi, P. M., Bebeshko, B. T., Khorolska, K. V., Rzaieva, S. L., & Vorokhob, M. V. (2025). *Information and communication systems security*. [Textbook] Kyiv: Borys Grinchenko Kyiv Metropolitan University.
22. Kostiuk, Yu. V., Skladannyi, P. M., Hulak, H. M., Bebeshko, B. T., Khorolska, K. V., & Rzaieva, S. L. (2025). *Information security systems.* [Textbook] Kyiv: Borys Grinchenko Kyiv Metropolitan University.
23. Hulak, H. M., Zhyltsov, O. B., Kyrychok, R. V., Korshun, N. V., & Skladannyi, P. M. (2023). *Enterprise information and cyber security.* [Textbook] Kyiv: Borys Grinchenko Kyiv Metropolitan University.

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (28), 2025
ISSN 2663 - 4023

Oleksandr Laptiev
Doctor of Technical Science, Senior Researcher
Associate Professor the Department of Cyber Security and Information Protection
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine
ORCID ID: 0000-0002-4194-402X
olaptiev@knu.ua

Andrii Hapon
Ph.D Student (Postgraduate)
Department of Information Technology Security
Kharkiv National University of Radio Electronics, Kharkiv, Ukraine
ORCID ID: 0000-0003-2560-7426
andrii.hapon@nure.ua

Andrii Tkachov
Candidate of Technical Science, Senior Research Fellow
Associate professor of Cyber Security Department
National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine
ORCID ID: 0000-0003-1428-0173
andrii.tkachov@khpi.edu.ua

SOFTWARE PROTECTION METHOD BASED ON HYBRID CODE ANALYSIS

Abstract. Abstract. The article addresses current issues of software protection against malicious code and the detection of its manifestations during development and operation. It notes that modern methods of software analysis, particularly static and dynamic analysis, have both advantages and significant limitations, including a high number of false positives, low efficiency against polymorphic threats, and high computational resource requirements. As an optimal solution, the use of hybrid analysis is proposed, which combines the strengths of different approaches to improve the accuracy of vulnerability detection and reduce the number of erroneous results. The work presents a mathematical model for vulnerability detection based on symbolic execution and combined code analysis, as well as developed algorithms for constructing a reduced program path graph, calculating distance metrics to potentially dangerous code sections, and implementing directed dynamic symbolic execution. The methodology of vulnerability warning classification involves dividing them into three categories: confirmed, unconfirmed, and requiring additional inspection. This approach significantly reduces the complexity of analysis, improves the reliability of results, and automates the process of detecting potentially dangerous code. Particular attention is given to the formalization of concepts related to constraints on program path execution, symbolic conditions, and safety predicates. The obtained results demonstrate the effectiveness of hybrid analysis when working with large-scale projects where both speed and accuracy in threat detection are critical. The capabilities of the modular architecture of the hybrid analysis tool are examined, ensuring flexibility in expanding functionality and integrating new methods. An analysis of key software vulnerability metrics is conducted, which can be used to assess software security. Directions for further research are proposed, particularly improving symbolic execution algorithms to account for indirect dependencies and anti-analysis mechanisms. The research findings can be applied in the development of new systems and the modernization of existing code analysis tools aimed at enhancing software security.

Keywords: malicious code; signature search; detection of malicious software; program classification; code metrics; hybrid analysis; artificial neural networks; machine learning.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Microsoft Security Essentials / What is Malware? https://www.microsoft.com/uk-ua/security/business/security-101/what-is-malware.

2. Static and Symbolic Analysis. URL: https://www.talkcrypto.org/blog /2019/03/15/static-and-symbolic-analysis/ .
3. Python Type Checking. URL: https://testdriven.io/blog/python-typechecking/.
4. Delmas, D. 2022. Static analysis of program portability by abstract interpretation (Doctoral dissertation). Sorbonne Université.
5. Generating and using a Callgraph, in Python. URL: https://cerfacs.fr/coop/pycallgraph.
6. Data Flow Analysis. URL: https://www.codingninjas.com/studio/library/data-flow-analysis.
7. Python Control Flow Statements and Loops. URL: https://pynative.com/python-control-flow-statements/
8. 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.
9. Monat, R., Ouadjaout, A., Miné, A. 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.
10. B. Chess and G. McGraw, “Static analysis for security,” in IEEE Security & Privacy, vol. 2, no. 6, P. 76-79, Nov.-Dec. 2004, https://doi.org/ 10.1109/MSP.2004.111.
11. Rami Sihwail, Khairuddin Omar and Khairul Akram Zainol Ariffin,"A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis," International Journal on Advanced Science, Engineering and Information Technology, 2018. Vol. 8, no. 4-2, P. 16621671.
12. Tahir, R. A study on malware and malware detection techniques. International Journal Education and Management Engineering (IJEME) 2018. 8(2), P.20–30
13. Chowdhury, I., & Zulkernine, M. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. Journal of Systems Architecture, 2011. 57(3), P.294–313.
14. Shin, Y., & Williams, L. Can traditional fault prediction models be used for vulnerability prediction? Empirical Software Engineering, 2008. 13, P.497–530.
15. Zimmermann, T., Nagappan, N., Gall, H., Giger, E., & Murphy, B. 2010. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process.
16. Scandariato, R., Walden, J., Hovsepyan, A., & Joosen, W. Predicting vulnerable software components via text mining. IEEE Transactions on Software Engineering, 2014. 40(10), P.993–1006.
17. Laptiev O., Zozulya S. Method of excluding known signals when scanning a given radio band. Electronic professional scientific publication "Cybersecurity: education, science, technology". 2023. Vol. 2 No. 22. P. 31–38. https://doi.org/10.28925/2663-4023.2023.22.3138
18. Oleksandr Laptiev, Vitaliy Savchenko, Alla Kobozeva, Anatoliy Saliy, Timur Kurtseitov. Methods of assessing information security in communication networks. Cybersecurity: Education, Science, Technology. 2025. Vol. 3. No. 27. P. 522-533. https://doi.org/10.28925/2663-4023.2025.27.767
19. O. Laptiev, T. Laptieva, M. Brailovsky. Methods for calculating the parameters of detecting signals of means of covert information acquisition (radio emission sources. Electronic professional scientific publication "Cybersecurity: education, science, technology". 2025. Vol. 4 No. 28. 2025. P. 575–585. https://doi.org/10.28925/2663-4023.2025.28.812
20. O.Laptiev, N.Lukova-Chuiko, S.Laptiev, T.Laptieva, V.Savchenko, S.Yevseiev. Development of a method for detecting deviations in the nature of traffic from the elements of the communication network. International Scientific And Practical Conference “Information Security And Information Technologies”: Conference Proceedings. 13-19 September 2021. Kharkiv – Odesa, Ukraine. P.8-17, ISBN 978-966-676-818-9. Scopus
21. Kostiuk, Yu. V., Skladannyi, P. M., Bebeshko, B. T., Khorolska, K. V., Rzaieva, S. L., & Vorokhob, M. V. (2025). *Information and communication systems security*. [Textbook] Kyiv: Borys Grinchenko Kyiv Metropolitan University.
22. Kostiuk, Yu. V., Skladannyi, P. M., Hulak, H. M., Bebeshko, B. T., Khorolska, K. V., & Rzaieva, S. L. (2025). *Information security systems.* [Textbook] Kyiv: Borys Grinchenko Kyiv Metropolitan University.
23. Hulak, H. M., Zhyltsov, O. B., Kyrychok, R. V., Korshun, N. V., & Skladannyi, P. M. (2023). *Enterprise information and cyber security.* [Textbook] Kyiv: Borys Grinchenko Kyiv Metropolitan University.