Issue 18, part 1, 2025

https://doi.org/10.23939/sisn2025.18.209

УДК 004.415.5:004.056

ІНТЕГРАЦІЯ ЗАСОБІВ АНАЛІЗУ ВИХІДНОГО КОДУ У ІННОВАЦІЙНІЙ МЕТОДОЛОГІЇ DEVSECOPS

Олексій Дуда[2], Микола Орлов[1], Ігор Павлів[3]

[1] Тернопільський національний технічний університет імені Івана Пулюя,
кафедра комп'ютерних наук, Тернопіль, Україна
[2, 3] Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж, Львів, Україна
[1] E-mail: oleksij.duda@gmail.com, ORCID: 0000-0003-2007-1271
[2] E-mail: orlovnv86@gmail.com, ORCID: 0009-0007-9835-6177
[3] E-mail: micky.ukr@gmail.com, ORCID: 0009-0003-0957-0843

У статті розглянуто актуальність інтеграції засобів аналізу вихідного коду, зокрема статичного (SAST) та динамічного (DAST), у сучасні процеси безпечної розробки програмного забезпечення на основі інноваційної методології DevSecOps. Виконано огляд наукових підходів та сучасних практик інтеграції інструментів безпеки в CI/CD-конвеєри, проаналізовано переваги та обмеження SAST і DAST, а також окреслено тенденції розвитку комбінованих методів безпеки. У статті побудовано узагальнену модель інтеграції цих інструментів у DevSecOps-середовище, наведено результати експериментальних досліджень щодо ефективності застосування SAST та DAST окремо та у поєднанні. Сформульовано практичні рекомендації з оптимізації вибору інструментів і підвищення рівня безпеки програмного забезпечення з урахуванням продуктивності CI/CD-процесів.

Ключові слова: DevSecOps, CI/CD, SAST, DAST, безпека програмного забезпечення, автоматизоване тестування, інтеграція інструментів безпеки.

Постановка проблеми

В нинішніх умовах розроблення програмного забезпечення питання безпеки стають критично важливими на кожному етапі життєвого циклу програмного продукту. З розвитком хмарних технологій та децентралізованих систем, зростає потреба в автоматизації процесів безпеки та їх інтеграції у методологію Development (розроблення) and Operations (операції) (далі DevOps) пайплайн, візуалізації позиції потенційних клієнтів у процесі продажу. Як відповідь на цей виклик у фахових колах з'явився інноваційний підхід реалізації методології DevOps, який поєднує кібербезпеку та розроблення програмного забезпечення і отримав назву DevSecOps. Він орієнтований на інтеграцію функцій безпеки як основного компонента процесу розроблення, тестування та доставки програмного забезпечення. В рамках цього підходу ключову роль відіграють засоби аналізу коду на вразливості, як от статичний та динамічний аналіз коду. Вони дають можливість виявляти проблеми з безпекою на різних етапах розроблення програмного продукту і забезпечують максимальне охоплення і усунення потенційних загроз, це особливо стосується розроблення інформаційних систем для медичної галузі.

Аналіз останніх досліджень та публікацій

Актуальність інтеграції засобів автоматизованого аналізу безпеки вихідного коду в сучасні процеси створення програмного забезпечення визначається як зростанням складності програмних систем, так і збільшенням кількості кібератак, які вишукують вразливості у програмних продуктах.

Традиційні моделі забезпечення безпеки, що були притаманні класичним підходам до розроблення програмного забезпечення, передбачали проведення перевірок та аудитів безпеки переважно на завершальних етапах життєвого циклу. Проекти безпеки відкритих веб-застосунків (OWASP) широко визнані за свою роль у виявленні найкритичніших вразливостей у сфері безпеки веб-застосунків. У дослідженні (Li & Li, 2025). пропонується комплексний аналіз OWASP Top 10, систематично висвітлює ключові зміни у моделях вразливостей та нові виклики безпеці, ґрунтуючись на чотирьох основних факторах, що впливають на розвиток ландшафту кібербезпеки, зокрема, ризики, пов'язані зі штучним інтелектом та машинним навчанням, складність безпеки API та хмарних технологій, зростання частоти атак на ланцюги постачання програмного забезпечення та вплив нормативно-правових актів та систем дотримання вимог. Yadati (2021) описує практичний підхід до інтеграції інструментів динамічного тестування безпеки в CI/CD-конвеєри, демонструючи приклад безперервного тестування безпеки на основі кейс-стаді. Дослідник вважає, що безперервна інтеграція (CI) та безперервна доставка (CD) є ключовими практиками в DevOps, що забезпечення високої якості безпеки в програмних системах стає все більш важливим. DevSecOps прагне інтегрувати безпеку в практики DevOps, а автоматизоване тестування безпеки є життєво важливою сферою досліджень.

Незважаючи на велику кількість досліджень, що стосуються тестування безпеки та практики CI/CD, досить мало публікацій, що розглядають обидві підходи разом, і більшість зосереджується лише на статичному аналізі коду, нехтуючи методами динамічного тестування. На основі дослідження підходу до інтеграції автоматизованих методів динамічного тестування в конвеєр CI/CD дослідниками наведено емпіричний аналіз накладних витрат, визначено проблеми у спільноті DevSecOps та запроповано шляхи їх вирішення (Yadati, 2021). Campbell (2021) акцентує увагу на важливості інтеграції безпеки у всі етапи DevOps-процесів, що становить основу концепції DevSecOps. Цей підхід підкреслює принцип «зсуву вліво», де тестування безпеки, оцінка вразливостей та перевірки відповідності автоматизовані та виконуються на ранніх етапах та безперервно впродовж усього процесу розробки. Інтегруючи інструменти безпеки, такі як статичне та динамічне тестування безпеки застосунків, управління вразливостями та автоматизація безпеки інфраструктури, у робочі процеси безперервної інтеграції та безперервного розгортання (CI/CD), організації можуть виявляти та усувати ризики майже в режимі реального часу. Незважаючи на очевидні переваги, на думку дослідника, DevSecOps створює такі проблеми, як управління складними ланцюжками інструментів, усунення прогалин у навичках та забезпечення послідовного охоплення безпеки в різних середовищах (Campbell, 2021).

У досліджені Research Publication (2019) проаналізовано переваги впровадження безпеки впродовж усього життєвого циклу розроблення програмного забезпечення, зосереджуючись на безперервному тестуванні безпеки, автоматизованому скануванні вразливостей та захищеній інфраструктурі як коду (IaC). Дослідження демонструє, що DevSecOps покращує виявлення та пом'якшення загроз, одночасно сприяючи культурі спільної відповідальності між командами розробки, експлуатації та безпеки. Завдяки ранній та постійній інтеграції безпеки організації можуть досягти більш надійних та безпечних конвеєрів доставки програмного забезпечення. Дослідниками надаються практичні поради та рекомендації щодо ефективного впровадження DevSecOps.

Інтеграція засобів автоматичного аналізу вихідного коду у DevSecOps в основному реалізується за допомогою двох класів інструментів - SAST (Static Application Security Testing) і DAST (Dynamic Application Security Testing). Статичний аналіз коду (SAST) дозволяє на ранніх етапах виявляти структурні дефекти, небезпечні шаблони програмування та порушення вимог безпеки у вихідному коді, ще до його компіляції та запуску (Charoenwet, Thongtanunam, Pham, & Treude, 2024). Динамічний аналіз, у свою чергу, забезпечує виявлення вразливостей під час виконання програмного продукту, коли програма взаємодіє з навколишнім середовищем. Rangnau, Buijtenen, Fransen та Turkmen (2020) наголошують, що класичні методи управління безпекою не можуть встигати за швидким життєвим циклом розробки програмного забезпечення (SDLC). Нова тенденція DevSecOps спрямована на інтеграцію методів безпеки в існуючі практики DevOps. Поєднання цих підходів у межах безперервного CI/CD-конвеєра створює передумови для суттєвого підвищення рівня безпеки програмних систем. В останні роки

дослідження у цій сфері зосереджуються на методах інтеграції засобів SAST та DAST у автоматизовані пайплайни, які забезпечують їх безперервне застосування без значного впливу на швидкість збірки програмного продукту. Наприклад, Angermeir, Fischbach, Moyón та Méndez Fernández (2024) відзначають, що безперервне розроблення програмного забезпечення все частіше застосовується у високорегульованих сферах, що підвищує потребу в безперервній відповідності. Дотримання, зокрема, правил безпеки – головна проблема у високорегульованих сферах – робить безперервну відповідність безпеці високою актуальністю для галузі та досліджень. Однією з ключових перешкод для впровадження безперервного розроблення програмного забезпечення в галузі є ресурсомісткий та схильний до помилок традиційних ручних дій щодо дотримання вимог безпеки. Дослідники уточнили концепт «безперервна відповідність вимогам безпеки», запропонували дослідницьку дорожню карту для вирішення проблем за допомогою автоматизованої безперервної відповідності вимогам безпеки (Angermeir, Fischbach, Moyón та Méndez Fernández, 2024).

Basu, Sengupta, Bhattacharya, Srivastava, Mishra та Daniher (2023) розглядають підходи до підвищення рівня безпеки DevSecOps шляхом захисту CI/CD-конвеєрів у хмарних середовищах. У статті (Basu, Sengupta, Bhattacharya, Srivastava, Mishra, & Daniher, 2023) підкреслюється важливість впровадження практик, що в першу чергу ставляться до безпеки, у робочі процеси CI/CD на хмарних платформах для посилення методологій DevSecOps. Автоматизуючи заходи безпеки на кожному етапі конвеєра, організації можуть зменшувати ризики, підтримувати відповідність нормативним вимогам та підвищувати цілісність системи. Дослідники заглиблюються в ключові стратегії та інструменти інтеграції безпеки в хмарні процеси CI/CD, пропонуючи практичні поради щодо посилення DevSecOps у постійно мінливому технологічному середовищі. Pochu і Kathram (2024) пропонують комплексний підхід до інтеграції вимог безпеки на ранніх етапах життєвого циклу, що дозволяє проектувати більш захищене програмне забезпечення. Ними досліджується інтеграція вимог безпеки на ранніх етапах життєвого циклу розроблення програмного забезпечення (SDLC), аналізуючи важливі компоненти комплексної стратегії забезпечення якості (QA), з акцентом на безпеці на кожному етапі. Спираючись на висновки з дослідження "Розробка комплексної стратегії QA для безпечного програмного забезпечення: висновки з управління SQA" (Banik & Kothamali, 2019), авторами статті запропоновано комплексну структуру, яка включає раннє визначення вимог безпеки, постійний моніторинг та застосування тестування, зорієнтованого на безпеку. Цей підхід спрямований на пом'якшення ризиків, зменшення вразливостей та забезпечення відповідності нормативним стандартам, таким як GDPR та HIPAA.

Таким чином, виявлення вразливостей на пізніх стадіях збільшує вартість і час усунення дефектів, ускладнює контроль якості програмного продукту та призводить до зниження загальної швидкості виведення продукту на ринок. Використання спеціалізованих API, плагінів і контейнеризації дозволяє впроваджувати інструменти аналізу безпеки на різних етапах CI/CD із мінімальною затримкою, а автоматизоване формування звітності полегшує спільну роботу команд розробників та спеціалістів з безпеки. Проте у публікаціях наголошується на низці проблем, пов’язаних із використанням цих технологій, зокрема на зростанні кількості хибнопозитивних результатів, збільшенні часу аналізу та складності інтеграційної інфраструктури. Результати сучасних досліджень свідчать про значні досягнення у створенні і впровадженні інструментів SAST і DAST у DevSecOps-середовищах. Відзначається, що в основі DevSecOps лежить принцип «security as code», який передбачає, що безпека програмного забезпечення забезпечується автоматизованими методами на кожному етапі життєвого циклу – починаючи з фази проєктування і до етапу експлуатації.

Водночас залишаються невирішеними такі науково-практичні задачі, як розроблення системних методів порівняння ефективності різних комбінацій інструментів у різних сценаріях розробки; побудова універсальних метрик, що відображають баланс між якістю результатів безпекового аналізу та продуктивністю конвеєра; формування моделей впливу інтеграції інструментів безпеки на часові та ресурсні характеристики процесу розробки програмного забезпечення. Ці аспекти зумовлюють потребу подальших наукових досліджень, спрямованих на вдосконалення методів інтеграції автоматизованих засобів безпеки, розробку аналітичних моделей оцінки їх ефективності та впровадження експериментальних платформ для апробації інноваційних рішень у межах DevSecOps.

Формулювання цілі статті.

Метою статті є обґрунтування та експериментальне дослідження підходів до інтеграції засобів статичного (SAST) та динамічного (DAST) аналізу вихідного коду у процеси DevSecOps, з визначенням ефективності їх використання у безперервних CI/CD-конвеєрах та формуванням рекомендацій щодо оптимального поєднання цих інструментів для підвищення рівня безпеки програмного забезпечення.

Завдання статті полягають у проведенні огляду сучасних досліджень і практик щодо використання SAST і DAST у DevSecOps-середовищах; визначенні переваг та обмежень кожного з підходів (SAST і DAST) у контексті інтеграції в автоматизовані CI/CD-процеси.

Виклад основного матеріалу

DevSecOps – це методологія розроблення програмного забезпечення, яка інтегрує практики безпеки на кожному етапі життєвого циклу його розроблення. DevSecOps вважається еволюційною методологією DevOps, яка інтегрує практики безпеки безпосередньо в життєвий цикл розробки та експлуатації програмного забезпечення. Оскільки в IT проєктах все частіше впроваджують гнучкі та безперервні моделі доставки, традиційний підхід до вирішення питань безпеки на пізніх етапах процесу розробки став недостатнім та ризикованим (Campbell, 2021). Характерною рисою DevSecOps є те, що безпека стає відповідальністю всіх учасників процесу, починаючи з розробників і закінчуючи командами операційної підтримки та безпеки. Метою запровадження цього інноваційного підходу є забезпечення постійного моніторингу та оцінки ризиків, пов'язаних із безпекою, без шкоди для швидкості процесів розроблення і доставки продукту. В традиційних моделях розроблення програмного забезпечення безпека, зазвичай, розглядалася як окремий етап, що слідує після етапів завершення розроблення та тестування, що формувало її як слабку ланку в загальному життєвому циклі. Такий підхід не тільки підвищує вартість виправлення вразливостей на пізніх стадіях, але й збільшує ризики отримання не якісного продукту через можливі додаткові часові затримки. У сучасних умовах розроблення програмного забезпечення важливо інтегрувати функцію його безпеки на ранніх етапах з метою уникнення фактів накопичення помилок, прискорення виявлення загроз та забезпечення їх оперативного усунення (див. рис. 1).

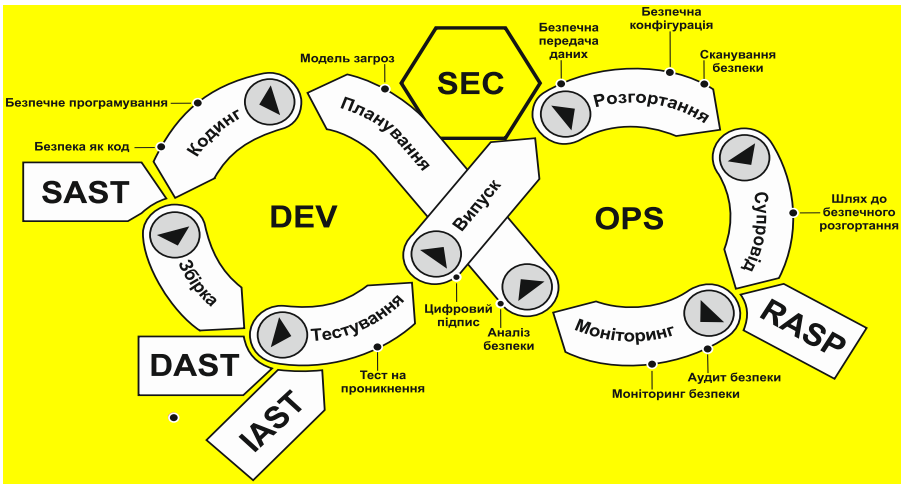

Рис. 1. Впровадження SAST, DAST, IAST та RASP на стадіях життєвого циклу розроблення програмного забезпечення

Оскільки вразливості у програмному забезпеченні та застосунках є основними векторами зовнішніх атак, то убезпечення застосунків стало ключовим завданням для більшості компаній. Одним із важливих способів зниження цього ризику є використання інструментів тестування безпеки застосунків (AST). Це завдання вирішується шляхом використання інструментів автома-

тизованого тестування безпеки SAST, DAST, IAST та RASP, які допомагають командам розробників і тестувальників виявляти проблеми ще до того, як продукт набуде статусу прийнятого у промислову експлуатацію. Впродовж останніх декількох років RASP (Runtime Application Self-Protection – самозахист застосунків під час виконання) активно обговорюється серед дослідників та фахівців з безпеки програмного забезпечення.

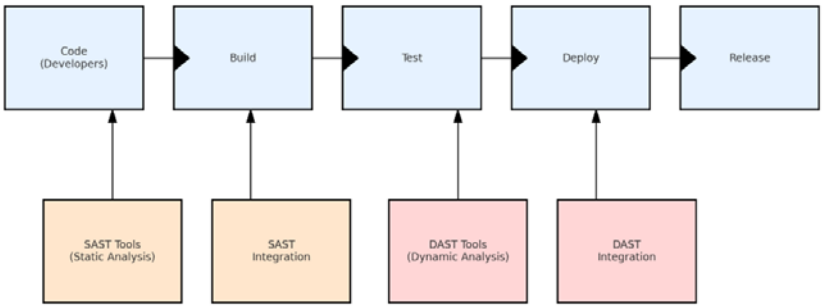

Рис.2. Схема інтеграції SAST і DAST у DevSecOps

SAST (Static Application Security Testing) – це процес аналізу вихідного коду програмного продукту на наявність вразливостей без його реального виконання. SAST інтегрується на ранніх етапах (код, збірка), а DAST – на пізніх (тестування, деплой, реліз). Цей метод дає змогу виявляти уразливості на рівні коду, зокрема, йдеться про такі вразливості як SQL-ін'єкції, міжсайтовий скринінг – XSS, некоректне управління пам'яттю та інші проблеми, які можуть бути не ідентифіковані, або проігноровані під час ручного тестування. SAST зазвичай впроваджується на ранніх стадіях життєвого циклу розроблення програмного забезпечення (ЖЦ), зокрема на етапах написання та тестування коду (див. Рис. 1). Динамічне забезпечення безпечності застосунків (Dynamic Application Security Testing, далі DAST) – це метод тестування безпеки програмного продукту під час його виконання. DAST дає можливість виявляти вразливості в реальному масштабі часу, коли програмний продукт взаємодіє з різними елементами середовища і проводить його перевірки з метою отримання коректної вичерпної відповіді на запитання чи можна експлуатувати програмний продукт з недоліками в реальних робочих умовах. Цей метод є ефективнішим на пізніх стадіях (SDLC). Мова йде про етапи тестування та валідації в реальних умовах, оскільки надає можливості виявляти вразливості, які залежать від середовища виконання або налаштувань конфігурації коду. Інтерактивне тестування безпеки застосунків IAST – це інструмент AST, розроблений для сучасних веб та мобільних застосунків. Він працює безпосередньо всередині програмно-алгоритмічних засобів, аналізуючи її поведінку під час виконання та повідомляючи про виявлені вразливості в режимі реального часу. IAST є відносно новим підходом, який поєднує риси статичного (SAST) та динамічного (DAST) аналізу, що є більш усталеними методами.

Термін RASP був вперше поданий в звіті компанії Gartner у 2012 році. У глосарії Gartner RASP визначається як: «технологія безпеки, вбудована або пов’язана з програмою чи середовищем виконання програми, яка здатна контролювати виконання програми, виявляти та запобігати атакам у режимі реального часу». RASP – це метод захисту застосунків із середини під час їх виконання, що працює на відміну від класичних підходів, як от мережеві брандмауери чи системи запобігання вторгненням, які діють ззовні. RASP автоматично зупиняє шкідливу активність без необхідності втручання користувача, забезпечуючи «самозахист» за рахунок адаптації до поточної ситуації.

Інтеграція процесів SAST, DAST та IAST у інноваційній методології DevSecOps забезпечує реалізацію постійного моніторингу і перевірку наявності вразливостей впродовж усього циклу розробки програмного продукту. Це дає можливість загалом знизити ризики за рахунок виявлення проблем на ранніх етапах виробничого процесу, що суттєво зменшує витрати на виправлення виявлених вразливостей. Автоматизація цих процесів забезпечує реалізацію принципу безперервної безпеки, даючи

команді можливість зосередитися на розробці та покращенні продукту, уникаючи процедур постійного ручного тестування. Окрім того, використання автоматизованих засобів аналізу дає змогу покращити відповідність результуючого програмного коду стандартам безпеки та регуляторним вимогам, що є критично важливим у багатьох галузях, наприклад, фінанси, медицина чи енергетика.

IAST був розроблений, щоб поєднати переваги SAST і DAST, водночас мінімізуючи їхні недоліки. Як і DAST, IAST проводить тестування під час виконання програмного коду, але, на відміну від нього, може точно вказати на проблемний рядок коду. Як і SAST, IAST аналізує сам код, однак робить це вже після збірки, в динамічному середовищі, завдяки інструментуванню коду. IAST легко інтегрується в CI/CD-процеси, добре масштабується, підтримує автоматизацію та може використовуватись як вручну, так і автоматизовано, що робить його гнучким і ефективним інструментом убезпечення сучасних застосунків. Технологія RASP інтегрується безпосередньо в застосунок: її функції з моніторингу, виявлення та запобігання загрозам додаються на сервери, де працюють програмно-алгоритмічні засоби. Всі запити до сервера перехоплюються RASP для перевірки на безпеку, після чого застосовуються відповідні захисні дії. Аналіз запитів виконується безпосередньо в застосунку. При цьому архітектура програмно-алгоритмічних засобів не змінюється. Залежно від типу впровадження, RASP може підтримувати різний рівень охоплення, продуктивності, інтеграції та надання сповіщень у режимі реального часу. Таким чином, інтеграція підходів SAST, DAST, IAST та RASP у інноваційній методології DevSecOps підвищує загальний рівень безпеки та стійкості використовуваних програмних продуктів, зменшуючи час і витрати на усунення вразливостей та підвищуючи якість програмного забезпечення.

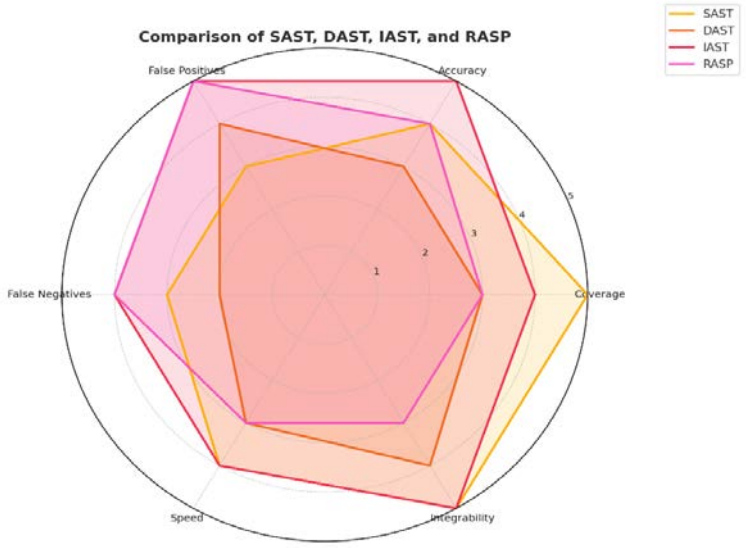

Рис.3. Порівняльна діаграма SAST, DAST, IAST, RASP за параметрами охоплення, точність, False Positives, False Negatives, швидкість, інтеграційність.

Підхід SAST передбачає аналіз вихідного коду, байт-коду або бінарних файлів програмного продукту без необхідності його реального запуску. Інструмент SAST сканує код, забезпечуючи аналіз його структури, синтаксису та логіки на предмет наявності попередньо відомих шаблонів, що можуть призводити до вразливостей, зокрема вище згадані SQL-ін'єкції, міжсайтовий скриптинг (XSS), вразливе управління пам'яттю тощо. Процес сканування може бути автоматизованим і інтегрованим у пайплайн постійної доставки коду (CI/CD), що дає розробникам можливість отримувати зворотний зв’язок практично миттєво після внесення змін у код.

Інструменти SAST аналізують код у середовищі розробки і можуть забезпечити високу точність при виявленні вразливостей, характерних для статичної структури програмного забезпечення. Оскільки підхід SAST передбачає реалізацію без запуску коду, він дає змогу виявляти вразливості програмного

забезпечення на ранніх етапах його розроблення. Переваги SAST полягають у тому, що цей підхід забезпечує раннє виявлення вразливостей, дозволяючи знаходити потенційні проблеми ще на етапі написання коду, що значно знижує вартість і ризики їх виправлення у майбутньому. Інструменти SAST здатні здійснювати глибокий аналіз коду, включно з бібліотеками та сторонніми компонентами, що дає можливість виявляти навіть приховані вразливості. Автоматизація процесів дозволяє інтегрувати SAST у розробку програмного забезпечення і завдяки цьому автоматизувати перевірки безпеки коду, скорочуючи час на ручну роботу. Завдяки інтеграції з CI/CD SAST надає змогу автоматизувати перевірки, включити їх у пайплайн і отримувати зворотний зв'язок у реальному часі. Цей підхід може виявляти різноманітні вразливості, такі як SQL-ін'єкції, міжсайтовий скриптинг (XSS), вразливості до відмови в обслуговуванні (DoS), некоректне управління пам'яттю та інші. Використання SAST сприяє написанню безпечнішого та надійнішого коду, що зменшує технічні витрати на його подальше обслуговування, а також забезпечує широке охоплення, оскільки дозволяє аналізувати всі частини коду, включаючи модулі, які складно протестувати динамічними методами.

Недоліки SAST полягають у тому, що результати аналізу можуть містити значну кількість хибнопозитивних спрацювань, оскільки перевірка базується на заздалегідь визначених правилах і шаблонах, що вимагає додаткових ручних перевірок для уточнення. Цей підхід не виявляє вразливості, які залежать від середовища виконання, оскільки обмежується аналізом вихідного коду і не дозволяє ідентифікувати проблеми, що виникають у процесі взаємодії з середовищем або зовнішніми системами. Крім того, під час використання складних фреймворків можуть виникати обмеження, оскільки не всі інструменти SAST здатні повністю підтримувати всі мови програмування та фреймворки, що ускладнює і подовжує процес аналізу.

Підхід DAST (Dynamic Application Security Testing) фокусується на тестуванні програмного продукту під час його роботи в реальному середовищі. Метод передбачає проведення атак на працюючий програмний продукт для виявлення вразливостей під час його використання. DAST імітує дії зловмисників, виконуючи тести на впровадження шкідливих даних, виявлення логічних помилок або неправильне налаштування засобів безпеки. DAST використовується для виявлення вразливостей, які можуть проявитися лише під час виконання програми: неправильна автентифікація, некоректна обробка даних з зовнішніх джерел або помилки управління сеансами. Цей підхід ідеально підходить для перевірки вже працюючих програмних продуктів у тестовому або продакшн середовищі.

Переваги DAST полягають у тому, що аналіз програмного забезпечення здійснюється у реальному середовищі, що дозволяє виявляти вразливості безпосередньо під час фактичного функціонування застосунку і робить цей метод особливо ефективним для виявлення проблем, пов'язаних із середовищем або налаштуванням конфігурації. Цей підхід добре підходить для тестування вебзастосунків, забезпечує роботу з веб-інтерфейсами та API і дає змогу виявляти такі загрози безпеки, як SQL-ін'єкції, міжсайтовий скриптинг (XSS) та інші. Виконання тестів на реальному працюючому застосунку знижує кількість хибнопозитивних результатів, адже виявлені вразливості є більш релевантними і рідко призводять до помилкових тривог. Разом з тим, DAST має і недоліки, серед яких обмеженість покриття коду, адже він не аналізує внутрішню структуру програмного забезпечення і тому не завжди дозволяє знаходити вразливості, пов'язані з логікою або синтаксисом програми. Виявлення проблем відбувається на пізніх етапах життєвого циклу розроблення, що може затримувати випуск продукту і збільшувати вартість їх усунення, а також неможливість знайти приховані вразливості, які не проявляються у процесі виконання програми.

Підхід IAST (Interactive Application Security Testing) зазвичай впроваджується шляхом встановлення агентів і сенсорів у застосунок після завершення його збірки. Агенти контролюють роботу програмно-алгоритмічних засобів та аналізують вхідний і вихідний трафік з метою виявлення потенційних загроз безпеці. Вони здійснюють це, зіставляючи шаблони або сигнатури атак із частинами коду, що дає змогу виявляти складні вразливості, які інші методи можуть пропустити.

Результати тестування IAST, як правило, надаються негайно – через вебінтерфейс, панель моніторингу або у вигляді звітів, які не впливають на швидкість CI/CD-конвеєра. Крім того, IAST можна інтегрувати з іншими системами відстеження помилок або вразливостей для більш зручного управління

результатами. Основною особливістю IAST є те, що на відміну від SAST і DAST, він працює безпосередньо всередині самого застосунку. Проте IAST не аналізує повністю всю кодову базу – він зосереджується лише на тих ділянках, які були активовані під час тестування. Це робить його значно швидшим за SAST, однак забезпечує менше охоплення, оскільки не всі частини коду можуть бути протестовані. IAST – це технологія, створена з урахуванням потреб розробників, яка сприяє впровадженню практики раннього тестування безпеки (shift-left) у процесі розробки. Попри численні переваги, IAST також має свої обмеження. Розгляньмо основні сильні та слабкі сторони цього підходу.

Переваги IAST полягають у тому, що він забезпечує мінімальну кількість помилкових спрацювань і на відміну від SAST, який часто генерує велику кількість хибнопозитивних попереджень, цей підхід гарантує дуже низький рівень помилкових результатів. Для компаній, де щодня з'являються тисячі повідомлень про потенційні загрози, точність виявлення вразливостей є вирішальною для зменшення інформаційного шуму та запобігання перевантаженню команди безпеки. Важливою перевагою є також оперативний зворотний зв'язок: щоб не відставати від швидкого темпу розробки, команди потребують інструментів, які надають результати перевірки в реальному часі безпосередньо розробникам, і IAST відповідає цій вимозі. Крім того, він легко інтегрується з CI/CD-процесами, що дозволяє виявляти та усувати проблеми ще на ранніх етапах розробки і таким чином заощаджувати ресурси компанії. Іншим важливим плюсом є добра масштабованість, адже IAST легко масштабувати у межах усієї організації – його можна налаштувати для кожного розробника, надаючи точну інформацію про місце розташування вразливості у коді, що дає змогу програмістам швидко її усунути без обов'язкової участі фахівців з безпеки. Водночас цей підхід має і свої недоліки. Серед них обмежена підтримка мов програмування та технологій, оскільки IAST інтегрується безпосередньо у тестований застосунок і його використання залежить від мови програмування та специфіки архітектури, через що він працює лише з певними сучасними фреймворками і не може самостійно забезпечити повне охоплення безпеки, а отже ефективніше працює у поєднанні з іншими інструментами AST. Крім того, для коректної роботи IAST необхідне добре структуроване та зріле середовище розробки і тестування програмного забезпечення, тому цей інструмент найкраще підходить для організацій, які вже мають налагоджені сучасні процеси. Ще однією проблемою є обмежене поширення IAST на ринку, оскільки його використання ще не стало масовим через недостатнє охоплення середовищ програмування, складність оцінювання ефективності інвестицій та відсутність чітко визначених сценаріїв, де його застосування дає очевидну перевагу.

Підхід RASP полягає у вбудовуванні захисного механізму безпосередньо в застосунок або його середовище виконання. На відміну від SAST, DAST чи IAST, які переважно виявляють вразливості під час розробки чи тестування, RASP діє в режимі реального часу – безпосередньо під час роботи програмно-алгоритмічних засобів. RASP постійно моніторить виконання коду, аналізує запити та дії користувачів і здатний зупинити потенційно шкідливу активність ще до того, як вона завдасть шкоду.

Інструменти RASP інтегруються у застосунок як агенти або бібліотеки, що дає їм можливість мати доступ до внутрішньої логіки та контексту виконання. Завдяки RASP можна точно визначати, чи є певна дія атакою (наприклад, спробою SQL-ін'єкції чи XSS) і вжити відповідних заходів – заблокувати запит, записати інцидент у журнал або повідомити систему моніторингу безпеки. Такий підхід дозволяє RASP не лише виявляти, а й активно протидіяти атакам у режимі реального часу, підвищуючи загальний рівень захищеності застосунка вже на етапі його експлуатації. RASP – це технологія, що зосереджується на безпеці під час виконання застосунка та працює безпосередньо всередині середовища виконання. Її головною метою є виявлення та запобігання атакам у режимі реального часу, що робить її надзвичайно цінною для захисту застосунків на етапі продуктивної експлуатації. Хоча підхід має значні переваги, існують також і певні обмеження. Переваги RASP полягають у тому, що цей підхід забезпечує захист у режимі реального часу, дозволяючи виявляти та нейтралізовувати атаки в момент їх здійснення, незалежно від того, чи були вони відомі заздалегідь. Це дає високий рівень безпеки для застосунків у продуктивному середовищі, особливо проти нових або цільових атак, здатних обійти інші шари захисту. Завдяки роботі безпосередньо у середовищі виконання RASP має повний доступ до контексту, що дає змогу точно ідентифікувати справжні атаки та мінімізувати кількість хибно-

позитивних спрацювань, істотно знижуючи навантаження на команди безпеки і дозволяючи їм зосередитися на дійсно критичних інцидентах. Додатковою перевагою є автоматизоване реагування, оскільки інструменти RASP можуть самостійно блокувати шкідливу активність, переривати сесії або змінювати поведінку застосунка, щоб запобігти експлуатації вразливостей без необхідності втручання розробників чи адміністраторів. Разом з тим, RASP має і недоліки. Його робота всередині застосунка може впливати на продуктивність, особливо у системах з високим навантаженням або обмеженими ресурсами. Крім того, він забезпечує захист лише тих частин застосунка, в які безпосередньо інтегрований, тому при наявності зовнішніх компонентів або інтеграцій повний захист усієї системи не завжди можливий. Ще одним обмеженням є залежність від середовища, оскільки ефективність RASP значною мірою залежить від мови програмування, фреймворку та конфігурації, що може ускладнювати його впровадження в розподілених або різнорідних IT-середовищах. В табл. 1 подано порівняльну характеристику SAST, DAST, IAST та RASP. На рис 4. подано блок-схему процесів статичного, динамічного та інтерактивного аналізу програмного коду з подальшим самозахистом застосунку.

Таблиця 1

Порівняльна характеристика підходів SAST, DAST та IAST

	Покриття	Рівень хибно-позитивних спрацьовувань	Рівень хибно-негативних спрацьовувань	Можливість практичного використання вразливості	Видимість коду	Інтеграція в життєвий цикл розробки програмного забезпечення (SDLC)
SAST	Так	Високий	Середній	Ні	Так	Так
DAST	Ні	Середній	Високий	Так	Ні	Так
IAST	Ні	Низький	Низький	Так	Так	Так
RASP	Ні	Низький	Низький	Так	Ні	Частково

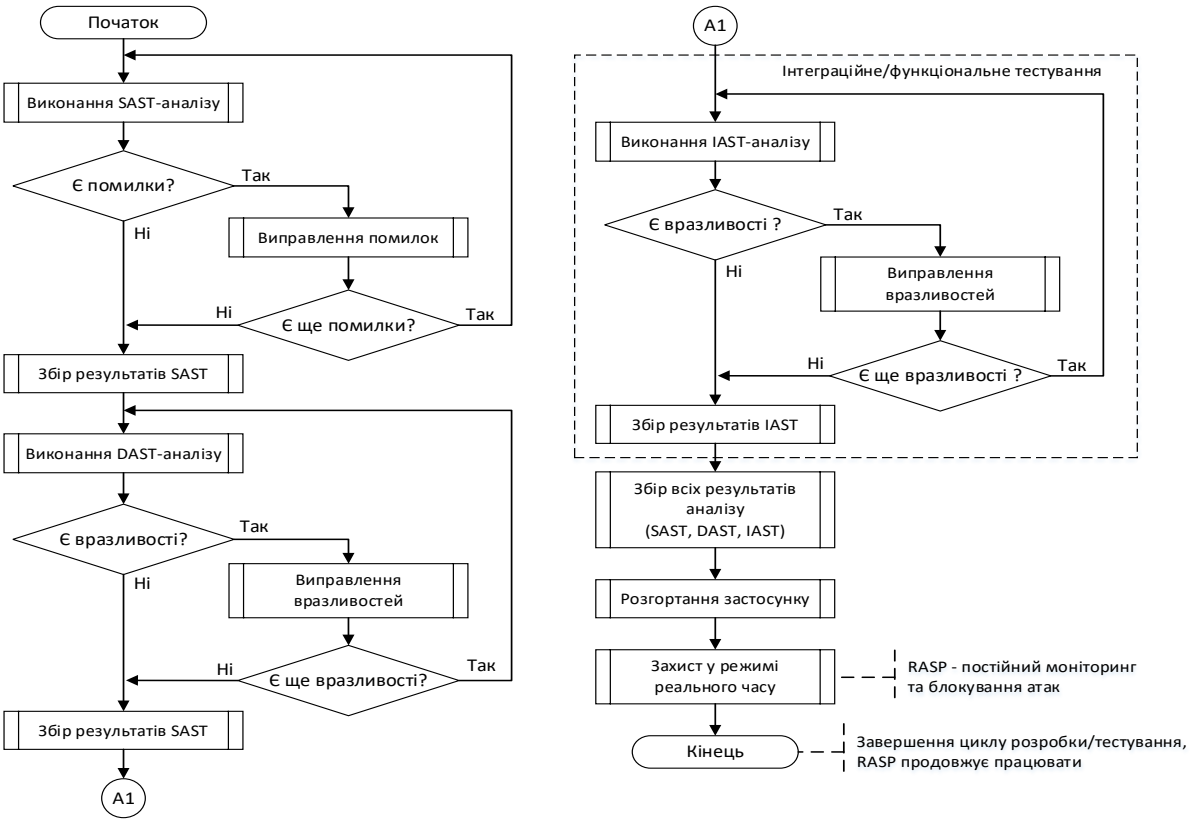

Рис. 4. Блок-схема процесу статичного та динамічного аналізу програмного коду

Порівняльний аналіз SAST, DAST, IAST та RAST у контексті DevSecOps

Порівняльний аналіз SAST (статичного аналізу коду), DAST (динамічного аналізу застосунків), IAST (інтерактивного аналізу застосунків) у рамках DevSecOps та RASP (самозахисту застосунків) є критичним для вибору правильних інструментів і методів забезпечення безпеки на всіх етапах життєвого циклу розробки програмного забезпечення (SDLC). Кожен із цих підходів має свої особливості, переваги та обмеження. У цьому розділі розглянемо, процеси які використовують три методи, їхню продуктивність, типи вразливостей, які вони виявляють, витрати на виправлення виявлених проблем, а також ризики хибнопозитивних і хибнонегативних ситуацій.

Одним із ключових аспектів інтеграції засобів безпеки у DevSecOps є визначення, на яких етапах життєвого циклу розробки слід впроваджувати підходи SAST і DAST. Статичний аналіз коду (SAST) проводиться на ранніх етапах розроблення, ще до того, як код зібрано або розгорнуто. SAST можна вбудовувати в процеси безперервної інтеграції (CI), забезпечуючи сканування під час кожного сеансу змін у репозиторії. Це надає можливість виявляти вразливості безпосередньо в коді та надавати зворотний зв'язок розробникам ще на стадії написання коду. Наприклад, при кожному запуску CI/CD пайплайна можна автоматично запускати інструменти SAST для сканування нових або змінених фрагментів коду. Такий підхід мінімізує ризик впровадження вразливостей ще до того, як код дійде до етапу тестування або розгортання.

Динамічний аналіз застосунків (DAST) проводиться на пізніших етапах SDLC, коли застосунок вже працює в середовищі виконання або в тестовому середовищі, що максимально наближене до фінальної версії. DAST дає можливість тестувати безпеку функціонуючого застосунка, його взаємодію з іншими системами, зокрема перевіряти реакцію на зовнішні атаки. Цей метод надзвичайно важливий для тестування веб-застосунків та сервісів, оскільки він симулює атаки, які можуть бути використані в реальних сценаріях. DAST часто застосовується на стадії випуску версії або під час предфінальних тестувань. Інтерактивне тестування безпеки застосунків (IAST) поєднує елементи SAST і DAST і застосовується переважно на етапах тестування в інтеграційному або QA-середовищі. IAST впроваджується шляхом інтеграції агентів безпосередньо в застосунок, що дає йому можливість працювати в режимі реального часу в процесі виконання. Завдяки цьому інструмент може виявляти вразливості з високою точністю, надаючи негайно інформацію про конкретні проблемні ділянки коду. Інтеграція IAST у CI/CD пайплайни дає змогу проводити більш точне та контекстне тестування без значного впливу на продуктивність, що робить його ефективним доповненням до традиційних методів SAST і DAST.

Захист під час виконання (RASP) застосовується на етапі продакшн або у середовищі попереднього розгортання. На відміну від SAST, DAST чи IAST, RASP працює безпосередньо під час виконання застосунка, інтегруючись у його середовище через агента або бібліотеку. Це дає змогу виявляти й блокувати атаки в режимі реального часу, ґрунтуючись на фактичній поведінці застосунка, а не лише на сигнатурах або відомих патернах вразливостей. Основна перевага RASP полягає в тому, що він надає рівень самозахисту застосунку, даючи йому можливість автономно реагувати на загрози навіть у разі, коли вразливість не була виявлена під час тестування. Завдяки цьому підхід RASP є особливо корисним у продакшн-середовищах, де важлива як точність, так і здатність до негайного реагування без втручання людини.

Продуктивність інструментів AST залежить від їхнього застосування на різних етапах життєвого циклу та специфіки проєкту. Важливо зрозуміти, як саме інструменти SAST, DAST, IAST та RASP масштабуються залежно від розміру коду або складності системи. SAST зазвичай працює швидше, коли використовується для невеликих кодових баз, однак його продуктивність може знижуватися в міру збільшення розміру проєкту. У великих проєктах статичний аналіз може займати значний час і вимагати додаткових обчислювальних ресурсів. Для оптимізації цього процесу інструменти SAST часто налаштовуються на інкрементальне сканування, тобто скануються тільки нові або змінені фрагменти коду, що дає змогу уникати повторного сканування всього проєкту. DAST тестує працюючий застосунок, його продуктивність залежить від середовища, в якому застосунок розгорнуто, а також від складності тестованої системи. Чим більше взаємодій між компонентами застосунка, тим більше часу займає динамічний аналіз. Важливим аспектом DAST є можливість пара-

лельного тестування, що може значно прискорити процес у випадку великих програмних систем. IAST є потужним інструментом у сфері тестування безпеки застосунків (AST). Завдяки своїй динамічній роботі IAST пропонує ряд переваг у процесі створення безпечного програмного забезпечення.

У порівнянні з методами SAST та DAST, він зазвичай демонструє кращу швидкість і точність результатів, а також легко вбудовується в CI/CD-процеси. Втім, IAST має і певні обмеження. Він не завжди підтримує всі мови програмування та технологічні стеки, які можуть використовуватися в установі чи організації. На відміну від SAST, цей інструмент не аналізує кожен рядок коду, а зосереджується лише на тих частинах, які активуються під час тестування. IAST найбільш ефективний при використанні в комбінації з іншими типами AST – як от SAST і DAST.

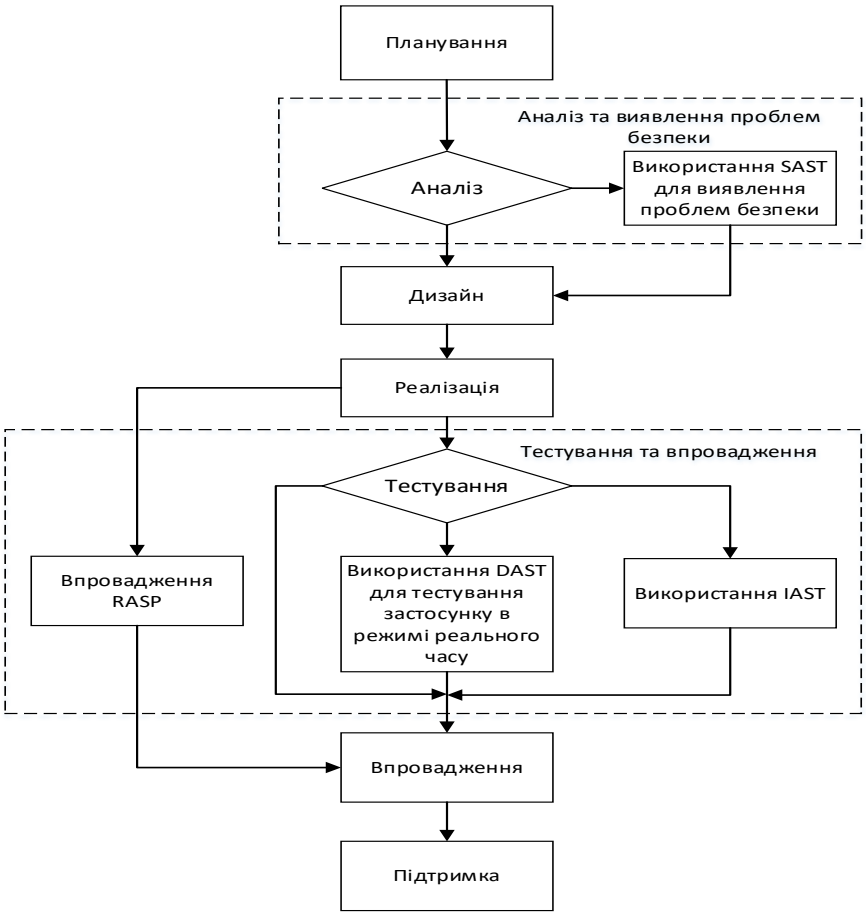

Рис 5. Позиція SAST, DAST, IAST та RASP у часовій шкалі SDLC.

RASP працює безпосередньо в середовищі виконання застосунка, що дає змогу виявляти та блокувати атаки в режимі реального часу. Такий підхід забезпечує високий рівень захисту на етапі експлуатації, однак супроводжується певним впливом на продуктивність. Хоча більшість сучасних RASP-рішень оптимізовані для мінімізації накладних витрат, у застосунках із великою кількістю транзакцій або за умов високих навантажень цей вплив може бути відчутним. Саме тому важливо попередньо проводити навантажувальне тестування перед розгортанням у продуктивному середовищі.

Масштабованість RASP значною мірою визначається архітектурою програмного забезпечення. У монолітних системах інтеграція RASP зазвичай не викликає складнощів, тоді як у мікросервісній або контейнеризованій архітектурі (наприклад, у Kubernetes) кожен сервіс або модуль потребує окремої конфігурації агента безпеки. Це може збільшити складність керування, але водночас забезпечує гнучкість і точковий контроль безпеки на рівні окремих компонентів. Для ефективного використання RASP у рамках DevSecOps-підходу доцільно інтегрувати його розгортання, оновлення та налаштування у CI/CD-процеси. Така автоматизація дає змогу оперативно впроваджувати зміни кон-

фігурацій і забезпечувати актуальність засобу безпеки навіть у динамічних середовищах. Попри те, що основне призначення RASP полягає в захисті вже розгорнутих застосунків, його CI/CD-інтеграція перетворює його на важливий компонент загальної безпекової інфраструктури.

Типи виявлених вразливостей

Методи SAST, DAST, IAST та RASP виявляють різні типи вразливостей, що робить їх взаємодоповнюючими інструментами у контексті забезпечення незалежного рівня безпеки програмних продуктів.

SAST, як інструменти статичного аналізу здатні виявляти вразливості у вихідному коді, як от помилки, пов'язані з неправильним управлінням пам'яттю (буферні переповнення, подвійне звільнення пам'яті), проблеми із захистом даних (некоректне або недостатнє шифрування), недоліки в управлінні ресурсами та помилки логіки програми. SAST особливо ефективний для виявлення вразливостей, які можуть бути непомітні при тестуванні на етапі виконання програми. DAST як інструменти динамічного аналізу фокусуються на вразливостях, які проявляються під час роботи застосунку в реальних умовах, зокрема SQL-ін'єкції, XSS (міжсайтовий скриптинг), неправильне опрацювання HTTP-запитів або інші веб-вразливості, проблеми з автентифікацією та управлінням сесіями. DAST дає можливості перевірити, як застосунок реагує на зовнішні атаки, і виявити вразливості, пов'язані із взаємодією з іншими компонентами або базами даних. IAST поєднує можливості SAST і DAST, аналізуючи як вихідний код, так і його поведінку під час виконання. Завдяки впровадженню агента безпосередньо в застосунок, IAST дає змогу виявляти складні вразливості в режимі реального часу в процесі виконання тестових сценаріїв або інтеграційних перевірок. Цей метод особливо цінний тим, що визначає точний контекст коду, де виникає проблема, і надає негайне повідомлення розробникам. IAST підходить для середовищ, де важлива швидкість реагування на виявлені загрози без шкоди для точності результатів. RASP функціонує безпосередньо в середовищі виконання застосунка, забезпечуючи виявлення та активне блокування атак у режимі реального часу на основі аналізу поведінки та контексту виконання. На відміну від інших методів, RASP не лише сигналізує про потенційні загрози, а й самостійно реагує на них, знижуючи ризик експлуатації вразливостей без потреби зовнішнього втручання. Цей підхід дає змогу ефективно протидіяти загрозам, як от ін’єкції, зловмисне використання API, несанкціонований доступ і спроби обійти механізми автентифікації, навіть якщо вразливість ще не була ідентифікована іншими засобами. RASP корисний для захисту складних, розподілених або мікросервісних архітектур, де традиційні методи можуть бути недостатніми. Завдяки інтеграції в сам застосунок, RASP забезпечує високий рівень точності виявлення загроз, мінімізує кількість хибнопозитивних спрацювань та виступає як важливий елемент багаторівневої стратегії кібербезпеки в DevSecOps.

Одним із найважливіших аспектів для організацій є порівняння витрат на виявлення та виправлення вразливостей на різних етапах розробки. SAST використовується на ранніх етапах SDLC, даючи змогу виявляти і виправляти вразливості ще на стадії написання коду, що значно знижує вартість їх виправлення. Виправлення вразливостей на етапі розробки менш витратне, оскільки зміни можуть бути швидко впроваджені без необхідності модифікації архітектури або повторного розгортання застосунка. AST проводить виявлення вразливостей за допомогою динамічного аналізу може відбуватися на пізніших етапах, коли застосунок вже розгорнуто або готовий до експлуатації. У таких випадках виправлення вразливостей може вимагати більших витрат, оскільки можуть бути потрібні зміни в архітектурі застосунка, а також повторне тестування та розгортання. IAST забезпечує більш збалансований підхід, даючи можливість виявляти вразливості як у коді, так і під час його виконання в процесі тестування. Інтеграція IAST у середовище CI/CD дає змогу отримувати детальну інформацію про вразливості в момент, коли код ще перебуває в тестовому середовищі, але вже виконується. Завдяки цьому установи та організації можуть ідентифікувати проблеми точніше та раніше, ніж при використанні лише DAST, але з вищою ефективністю, ніж SAST у складних динамічних сценаріях. Це дає можливість скоротити витрати на виправлення завдяки точному визначенню проблемних ділянок коду без затримок у процесі розгортання. RASP демонструє високу

ефективність виявлення вразливостей на етапі експлуатації, коли застосунок вже працює в реальному середовищі, забезпечуючи оперативне реагування на загрози. Завдяки своїй здатності до самозахисту, RASP не лише фіксує аномальну поведінку, а й здатен автоматично блокувати потенційні атаки без необхідності зупиняти роботу системи чи втручання з боку розробників. Це значно зменшує ризик експлуатації вразливостей, які не були виявлені на попередніх етапах тестування, і дає можливість уникнути критичних збоїв у роботі застосунка. Хоча витрати на впровадження RASP можуть бути вищими через потребу в інтеграції з кожним екземпляром застосунка, вони часто компенсуються зменшенням втрат, пов'язаних із потенційними інцидентами безпеки. Крім того, завдяки контекстному аналізу в режимі реального часу, RASP забезпечує можливість точно і швидко локалізувати проблему, що знижує витрати на її усунення порівняно з традиційними методами, які потребують ретроспективного аналізу або переробки коду.

Рис. 6. ілюструє вищу ефективність раннього виявлення проблем з використанням методу SAST. Поєднання методів SAST, DAST, IAST та RASP є основою комплексного підходу до забезпечення безпеки в інноваційній методології DevSecOps. Метод SAST ефективний на ранніх етапах розробки, забезпечуючи швидке виявлення вразливостей у коді, тоді як DAST дає можливість перевірити застосунок на предмет реальних загроз після його розгортання. Обираючи оптимальне рішення для безпеки застосунків, важливо враховувати співвідношення між точністю, швидкістю, охопленням і витратами. IAST та RASP є цінними доповненнями до арсеналу засобів захисту, однак самі по собі вони не забезпечить повного контролю над безпекою. Розуміння сильних і слабких сторін кожного з цих підходів сприяє коректному обранню правильної стратегії захисту на кожному етапі SDLC.

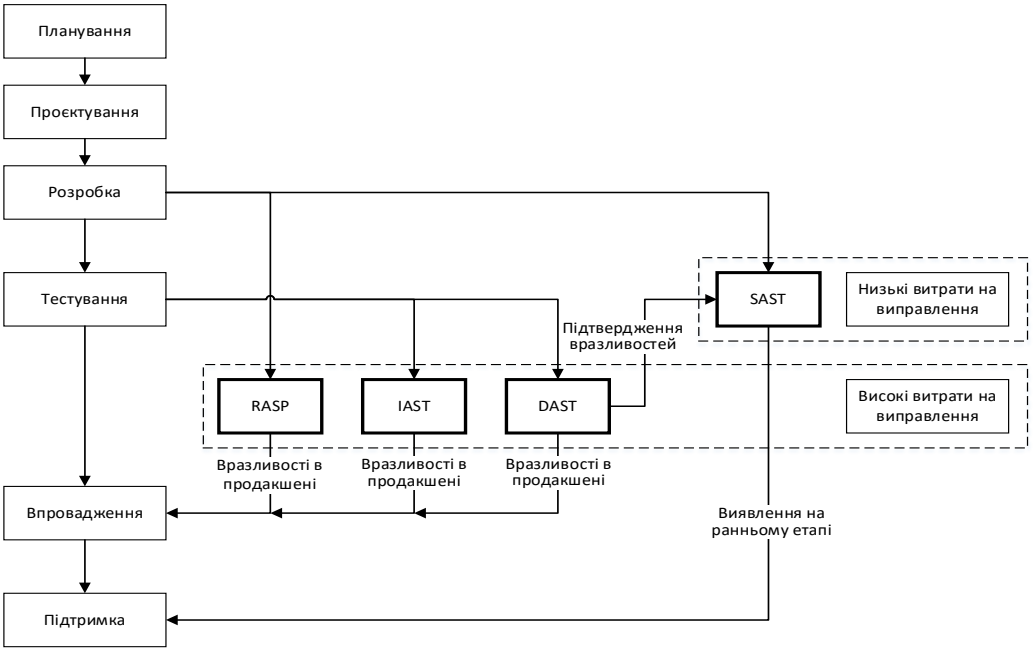

Рис 6. Збільшення витрат на виправлення в залежності від етапу виявлення вразливості.

Інструменти SAST, DAST та IAST в методології DevSecOps

Впровадження засобів статичного (SAST), динамічного (DAST) та інтерактивного (IAST) аналізу безпеки вихідного коду і самозахисту застосунків (RASP) у методології DevSecOps вимагає ретельного вибору інструментів, здатних інтегруватися у поточний життєвий цикл розробки, CI/CD процеси та відповідати вимогам проєктів. Ці інструменти повинні забезпечувати автоматичний моніторинг, зручність у використанні та бути здатними знаходити критичні вразливості на різних етапах.

SAST (Static Application Security Testing) фокусується на виявленні вразливостей шляхом аналізу вихідного коду на ранніх етапах розробки. Інструменти SAST допомагають ідентифікувати слабкі місця в коді до того, як застосунок буде зібрано або розгорнуто, тим самим надаючи можли-

вість знизити ризик виникнення критичних помилок безпеки. Проаналізуємо найпоширеніші інструменти SAST. SonarQube – інструмент з відкритим вихідним кодом для статичного аналізу коду, який підтримує понад 25 мов програмування. SonarQube інтегрується в процес CI/CD, даючи змогу постійно стежити за якістю коду та виявляти потенційні вразливості. Переваги включають багатий інтерфейс для аналізу якості коду, легку інтеграцію з системами на кшталт Jenkins, Azure DevOps та GitHub Actions. Checkmarx – професійний інструмент, який дає можливість проводити статичний аналіз для виявлення вразливостей безпеки на ранніх етапах SDLC. Він підтримує багато мов і може інтегруватися з різними системами CI/CD. Checkmarx також пропонує рекомендації щодо виправлення знайдених проблем, що полегшує роботу розробникам. Veracode Static Analysis вважається одним із лідерів на ринку SAST, який забезпечує глибокий аналіз і підтримує широкий спектр мов програмування та технологій. Veracode надає розробникам можливості швидко знаходити і виправляти вразливості завдяки автоматизованим звітам, легкої інтеграції та високій точності виявлення. Fortify Static Code Analyzer – комплексний інструмент для статичного аналізу коду, який підтримує понад 20 мов програмування і широко використовується в корпоративному середовищі. Fortify дає змогу аналізувати великі кодові бази на наявність вразливостей та легко інтегрується в існуючі CI/CD процеси.

Інтеграція з CI/CD забезпечується завдяки використанню інструментів SAST, зокрема, SonarQube або Checkmarx, що можуть бути легко інтегровані в CI/CD пайплайни для автоматичного аналізу коду на кожному етапі. Це даєможливість виявляти вразливості ще до розгортання застосунка, що значно знижує витрати на їх виправлення. Зазвичай вони запускаються на етапі побудови (build) або перевірки коду (pull request), аналізуючи зміни в кодовій базі і порівнюючи їх з раніше зафіксованими помилками. Автоматизація процесів досягається за допомогою сучасних інструментів, які забезпечують автоматизацію процесу виявлення та виправлення помилок безпеки. Це дає змогу знизити кількість ручної роботи та підвищити швидкість реакції на потенційні загрози.

Масштабованість забезпечується завдяки здатності більшості SAST інструментів працювати з великими проєктами та масштабуватися відповідно до потреб організацій. Це важливо для компаній з великим кодовим базисом і постійним потоком змін. DAST (Dynamic Application Security Testing) аналізує програмне забезпечення в процесі виконання для виявлення вразливостей, які можуть виникати через взаємодію компонентів програми. На відміну від SAST, DAST виявляє проблеми, пов'язані з поведінкою застосунка під час його виконання, що робить його важливим інструментом для кінцевої перевірки перед розгортанням.

Проаналізуємо найпоширеніші інструменти методу DAST. OWASP ZAP – інструмент з відкритим кодом, розроблений для автоматизованого динамічного аналізу безпеки веб-застосунків. ZAP дає змогу виявляти вразливості на різних рівнях застосунків, наприклад, SQL-ін’єкції, XSS (Cross-Site Scripting) та інші. ZAP підтримує інтеграцію з CI/CD пайплайнами, що робить його корисним інструментом у процесі DevSecOps. Burp Suite – один з найпопулярніших інструментів для динамічного тестування веб-застосунків, який включає сканування вразливостей та ручне тестування. Burp Suite також підтримує автоматизацію через API та добре інтегрується в CI/CD пайплайни, що робить його надійним вибором для проєктів, орієнтованих на безпеку. AppSpider – професійний інструмент для динамічного аналізу, який фокусується на тестуванні веб-застосунків і забезпечує глибокий аналіз взаємодії застосунку з користувачем. AppSpider дає можливість автоматизувати тестування та надавати докладні звіти про виявлені вразливості. Netsparker – це комерційний інструмент для DAST, який спеціалізується на виявленні вразливостей у веб-застосунках. Netsparker має високу точність виявлення та низький рівень хибнопозитивів, що допомагає швидко знаходити реальні загрози.

Інтеграція методів DAST у методологію DevSecOps має свої особливості. Інтеграція з пайплайном розгортання досягається завдяки тому, що інструменти DAST можуть бути легко інтегровані в етапи кінцевого тестування застосунків, що дає змогу виявити вразливості перед розгортанням. Зазвичай DAST запускається після побудови програми, коли застосунок розгорнутий на тестовому середовищі або в середовищі інтеграції. Автоматизація та сценарії тестування реалізується інстру-

ментами DAST, які можуть автоматизувати багато аспектів динамічного тестування, створюючи скрипти для тестування веб-застосунків та API. Це дає можливість повторно використовувати тестові сценарії для різних версій застосунка та виявляти вразливості під час кожного розгортання. Вибір інструментів для методів SAST і DAST доцільно проводити, використовуючи як багатокритеріальні методи прийняття рішень (МПР), так і інші підходи, які надають можливості оцінювання інструментів за різними ознаками та параметрами.

Інструменти для IAST (Інтерактивний аналіз безпеки)

IAST (Interactive Application Security Testing) поєднує елементи статичного та динамічного аналізу, виконуючи тестування безпеки застосунка в режимі його реального функціонування. Інструменти IAST працюють із середини програми, активно спостерігаючи за її поведінкою під час виконання та відслідковуючи потоки даних, виклики функцій, а також взаємодію з компонентами середовища. Це дає змогу точніше виявляти вразливості та зменшувати кількість хибнопозитивних спрацювань, характерних для інших типів аналізу. Проаналізуємо найпоширеніші інструменти IAST. Contrast Assess – один із провідних інструментів IAST, який інтегрується безпосередньо в застосунок і надає можливість автоматично виявляти вразливості в процесі тестування або роботи програми. Він забезпечує високу точність за рахунок аналізу реальних шляхів виконання коду та відображає виявлені проблеми безпеки з докладним контекстом і порадами щодо їх усунення. Підтримує широкий спектр мов програмування та сумісний із CI/CD-пайплайнами. Seeker (Synopsys) – корпоративний IAST-інструмент, що забезпечує глибокий моніторинг динамічної поведінки застосунків. Він автоматично виявляє критичні вразливості, як от SQL-ін’єкції або міжсайтовий скриптинг (XSS), з урахуванням контексту виконання. Seeker також інтегрується з DevOps-інструментарієм, включаючи Jenkins, GitLab та Azure DevOps, і дає змогу розробникам отримувати звіти в режимі реального часу. HCL AppScan IAST – інтерактивний модуль, що є частиною платформи AppScan, спеціалізується на виявленні вразливостей у застосунках під час автоматизованих або ручних тестів. AppScan IAST має гнучке розгортання та детальну аналітику щодо джерела, проходження та кінцевої точки небезпечних даних. Інструмент ефективно працює з Java, .NET та іншими популярними платформами, що робить його зручним у використанні для команд, орієнтованих на CI/CD та shift-left практики. RIPS IAST (SonarSource) – інструмент, що орієнтований на точне виявлення вразливостей у PHP, Java та інших популярних середовищах програмуання. Завдяки інтеграції з фреймворками тестування та середовищами CI, RIPS забезпечує ефективне відстеження уразливостей в режимі реального часу, пропонуючи деталізовані рекомендації для розробників та забезпечуючи швидке усунення проблем ще до релізу застосунка.

Інтеграції методів IAST у методологію DevSecOps має свої особливості. Вбудовування в середовище виконання є однією з ключових особливостей IAST. Інструменти інтерактивного аналізу безпеки інтегруються безпосередньо в застосунок або його середовище виконання, що дає змогу виявляти вразливості в режимі реального часу під час автоматизованих тестів, інтеграційного тестування або ручного використання. Завдяки цьому забезпечується висока точність виявлення з урахуванням реального контексту виконання коду, що застосовує перевірку шляхів передачі даних та їхню обробку в застосунку. Сумісність з CI/CD процесами дає IAST можливість органічно вбудовуватись у DevSecOps-підхід, забезпечуючи безперервний моніторинг безпеки на всіх етапах розробки та тестування. Більшість сучасних інструментів IAST мають підтримку популярних CI/CD-середовищ (таких як Jenkins, GitLab CI, Azure DevOps), що забезпечує гнучку інтеграцію та спрощує автоматизацію аналізу безпеки на кожному етапі розгортання. Вибір IAST-інструментів доцільно здійснювати на основі сумісності з інструментами та програмно-алгоритмічними засобами для методів SAST та DAST.

Інструменти для RASP (Самозахист застосунків у режимі виконання)

RASP (Runtime Application Self-Protection) – це технологія, що забезпечує активний захист застосунка безпосередньо під час його виконання. Інструменти RASP працюють інтегровані в програмне середовище, аналізуючи контекст виконання, поведінку застосунку та вхідні дані в режимі реального часу. Це дає змогу виявляти та блокувати атаки миттєво, без потреби зовнішнього

втручання. Завдяки тісній інтеграції з кодом застосунка та використанню сенсорів у середовищі виконання, RASP мінімізує кількість хибнопозитивних спрацювань і забезпечує високу точність реагування. Проаналізуємо найпоширеніші інструменти RASP. Contrast Protect – один із лідерів у сфері RASP, який інтегрується в застосунок на рівні середовища виконання та забезпечує проактивний захист від загроз на кшталт SQL-ін'єкцій, XSS, RCE тощо. Підтримує популярні мови програмування (Java, .NET, Node.js) та легко вбудовується у DevSecOps-процеси. Imperva RASP – корпоративне рішення, орієнтоване на захист веб-застосунків без зміни коду. Він забезпечує повну видимість виконання застосунка та зупиняє атаки ще до того, як вони призведуть до експлуатації вразливостей. Signal Sciences (Fastly) – RASP-компонент, що поєднується з WAF-рішенням, забезпечуючи поведінковий аналіз трафіку та захист у середовищах із мікросервісною архітектурою. Добре підходить для хмарних і контейнеризованих інфраструктур. Hdiv Protection – RASP-рішення з підтримкою Java та .NET, яке забезпечує глибоку інспекцію внутрішніх процесів застосунка, захищаючи від OWASP Top 10 вразливостей у режимі реального часу. Інтегрується з DevOps-процесами та підтримує автоматизоване оновлення політик безпеки.

Інтеграція RASP-інструментів у DevSecOps забезпечує безперервний захист застосунків у режимі реального часу, даючи змогу виявляти та блокувати атаки ще під час виконання, без потреби змінювати код або затримувати розгортання. Завдяки сумісності з CI/CD-процесами та автоматизованим оновленням політик безпеки, RASP-інструменти ефективно вписуються в DevSecOps-архітектуру, підвищуючи рівень захисту без порушення швидкості та гнучкості розробки.

Використання методу аналізу ієрархій для визначення необхідного інструменту реалізації проєкту

Метод аналізу ієрархій – це багатокритеріальний підхід, який дає можливість обрати найкращий варіант з множини альтернатив, враховуючи ряд параметрів. Використання методу передбачає побудову ієрархії критеріїв, в якій кожен критерій послідовно порівнюється з іншими за важливістю. Формула для обчислення ваги критерію:

$$\lambda_{\max} = \frac{\sum_{i=1}^{n} \left(\sum_{j=1}^{n} \frac{a_{ij}}{w_j} \right)}{n}$$

де a_{ij} – елементи матриці парних порівнянь; w_j – вага кожного критерію. Остаточна оцінка кожного інструменту визначається шляхом множення ваг критеріїв на значення їх оцінок для кожного інструменту.

Метод дає змогу враховувати як кількісні, так і якісні параметри інструментів, такі як продуктивність, зручність використання, вартість і підтримка мов програмування.

Кроки методу аналізу ієрархій

Крок 1. Визначення критеріїв

Крок 2. Побудова ієрархії критеріїв:

Визначаються основні критерії, за якими будуть оцінюватися інструменти SAST чи DAST.

Кожен критерій отримує вагу на основі оцінки важливості.

Крок 3. Порівняння альтернатив:

Проводиться парне порівняння кожного інструмента за кожним критерієм.

Оцінки робляться на основі шкали (1-9), де 1 – однакове значення, 9 – надзвичайна перевага одного інструмента над іншим.

Крок 4. Обчислення ваг для кожного інструмента та вибір найкращого.

Розглянемо приклад.

Крок 1. Визначення критеріїв. Припустимо, що оцінюємо три SAST інструменти: SonarQube, Checkmarx, і Veracode. Основні критерії, за якими вони будуть оцінюватися: вартість (Cost) – вартість ліцензії та підтримки; продуктивність (Performance) – швидкість аналізу коду і глибина перевірки; зручність інтеграції (Ease of Integration) – легкість інтеграції в CI/CD конвеєри; підтримка мов (Language Support) – кількість підтримуваних мов програмування; звітність та аналіз (Reporting & Analytics) – якість звітів і метрики вразливостей.

Крок 2: Побудова матриці парних порівнянь.

Таблиця 2

Оцінювання критеріїв за їх важливістю на основі проведених досліджень або попереднього досвіду

	Вартість	Продуктивність	Інтеграція	Підтримка мов	Звітність
Вартість	1	3	5	7	6
Продуктивність	1/3	1	3	5	4
Інтеграція	1/5	1/3	1	3	2
Підтримка мов	1/7	1/5	1/3	1	1.5
Звітність	1/6	1/4	1/2	1/1.5	1

Оцінюючи критерії таким чином, створюємо матрицю для кожного з критеріїв для порівняння інструментів.

Крок 3: Оцінка інструментів по кожному з критеріїв.

Таблиця 3

Парні порівняння інструментів за критеріями (наприклад, за критерієм «Продуктивність»)

	SonarQube	Checkmarx	Veracode
SonarQube	1	1/3	1/5
Checkmarx	3	1	1/2
Veracode	5	2	1

Після цього підраховуються відносні ваги для кожного інструмента.

Крок 4: Обчислення загальних ваг та вибір інструмента. Після того, як всі оцінки зведені у матриці і обчислені ваги для кожного інструмента по кожному критерію, обчислюється загальна вага для кожного інструмента. Вага інструмента по кожному критерію – це відношення суми його оцінок до суми оцінок всіх інструментів.

Таблиця 4

Загальні оцінки

Інструмент	Загальна вага
SonarQube	0.35
Checkmarx	0.45
Veracode	0.20

Checkmarx має найвищу вагу і, відповідно, є найкращим вибором за цим контекстом.

Виклики та обмеження інтеграції методів SAST і DAST у методології DevSecOps

Інтеграція інструментів статичного (SAST) і динамічного (DAST) аналізу вихідного коду в DevSecOps – є критично важливим процесом забезпечення належного рівня безпеки програмних продуктів. Однак, як і будь-який інший підхід, впровадження зазначених інструментів має свої виклики і обмеження. Детальніше розглянемо ключові аспекти, з якими можуть зіткнутися організації під час інтеграції методів SAST та DAST у методології DevSecOps.

Однією з основних проблем при інтеграції методів SAST та DAST є збільшення тривалості розроблення. Інструменти SAST, що аналізують код на ранніх етапах SDLC (Software Development Life Cycle), можуть збільшувати час збирання проєкту, особливо для великих кодових баз. DAST, у свою

чергу, потребує часу на тестування вже розгорнутих застосунків, що може сповільнити цикл розробки. Чим більше програмного забезпечення, тим довше триває перевірка. Інтеграція SAST/DAST може збільшувати час проходження тестів, що у свою чергу впливає на швидкість деплою та на CI/CD конвеєри.

В багатьох компаніях використовуються мультиплатформенні середовища, що означає присутність цілого ряду мов програмування в одному проєкті. Не всі інструменти SAST/DAST підтримують широкі спектри мов прогамування, що стає серйозним технологічним викликом. Деякі інструменти SAST та DAST можуть не мати повної підтримки мов програмування, що використовуються. Якість виявлення вразливостей може відрізнятися залежно від мови програмування, що використовується.

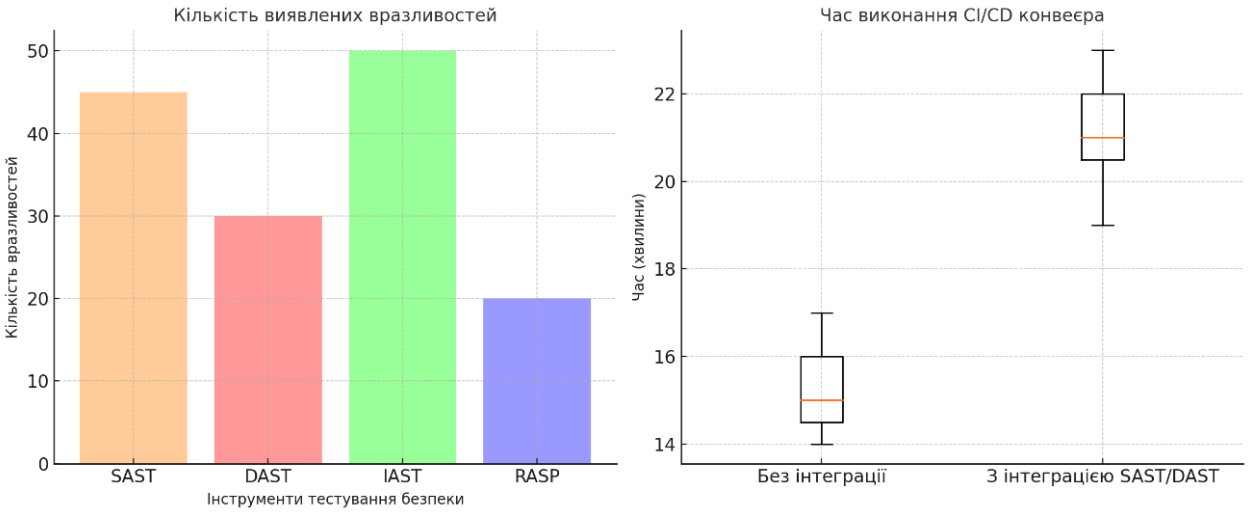

Рис. 7. Результати експерименту.
Ліва частина – кількість виявлених вразливостей різними інструментами.
Права частина – час виконання CI/CD-конвеєра без інтеграції засобів безпеки та з інтеграцією SAST/DAST.

Методологія DevSecOps передбачає постійне тестування безпеки на кожному етапі розробки, що вимагає глибокої інтеграції інструментів SAST і DAST з CI/CD конвеєрами. Проблема полягає в тому, що не всі інструменти легко інтегруються з існуючими системами автоматизації. Для правильної інтеграції можуть знадобитися значні ресурси на налаштування і адаптацію. Можливі конфлікти між різними засобами безпеки та автоматизації.

Висновки

Інтеграція інструментів статичного (SAST) і динамічного (DAST) аналізу коду у процеси методології DevSecOps є одним із ключових кроків на шляху до забезпечення належного рівня безпеки програмного забезпечення на всіх етапах життєвого циклу його розроблення. Ці інструменти забезпечують можливість виявляти та усувати вразливості ще на ранніх стадіях, знижуючи ризики їх потрапляння у фінальний продукт і запобігаючи серйозним наслідкам, пов'язаним із кіберзагрозами. Інструменти SAST дають можливість виявляти вразливості на ранніх етапах розробки, аналізуючи вихідний код до моменту його реального виконання, тоді як DAST здійснює тестування під час виконання застосунку, ідентифікуючи потенційні вразливості в реальному середовищі. У сукупності ці два підходи забезпечують вищий рівень безпеки і дають компаніям можливість швидше реагувати на загрози.

Незважаючи на переваги, інтеграція SAST і DAST може супроводжуватися низкою викликів: зниження продуктивності, підвищення вартості проекту, необхідність навчання розробників, а також труднощі у налаштуванні та підтримці інструментів у різних середовищах розробки. Ці виклики потребують значних ресурсів і стратегічного планування для їх подолання. Навіть найсучасніші інстру-

менти SAST і DAST мають певні обмеження, такі як велика кількість хибних спрацьовувань або недостатня підтримка специфічних мов програмування. Крім того, їх впровадження в контейнери та мікросервісну архітектуру також вимагає особливого підходу. Вибір конкретних інструментів SAST і DAST повинен базуватися на чітких критеріях: відповідність середовищу розробки, продуктивність, здатність інтеграції в існуючі CI/CD конвеєри. Метод аналізу ієрархій (AHP) є одним із науково–обґрунтованих підходів, який допомагає формалізувати процес вибору інструментів і прийняття зваженого рішення. У міру розвитку технологій безпеки, інструменти SAST і DAST також удосконалюються. Майбутнє може включати більшу автоматизацію процесів безпеки, покращення інтеграції з хмарними платформами і глибшу підтримку сучасних архітектур розроблення. Інвестиції в безпеку програмного забезпечення та активне впровадження інноваційної методології DevSecOps продовжуватимуть бути важливими аспектами у забезпеченні кіберстійкості фірм, компаній та організацій. Таким чином, інтеграція інструментів безпеки у DevSecOps є важливим кроком для досягнення балансу між швидкістю розроблення та надійною безпекою, які є надзвичайно важливими у медичній галузі. Для ефективного впровадження потрібно не тільки технічне налаштування, але й врахування організаційних аспектів, що дасть змогу зробити безпеку невід'ємною частиною розробницьких процесів.

REFERENCES

Angermeir, F., Fischbach, J., Moyón, F., & Méndez Fernández, D. (2024). Towards automated continuous security compliance. *Proceedings of the 2024 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 440–446. https://doi.org/10.1145/3674805.3690748.

Banik, S., & Kothamali, P. R. (2019). Developing an End-to-End QA Strategy for Secure Software: Insights from SQA Management. *International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence*, 10(1), 125–155

Basu, A., Sengupta, S., Bhattacharya, A., Srivastava, S., Mishra, A., & Daniher, S. (2023). Enhancing DevSecOps: Securing CI/CD pipelines on cloud platforms. https://www.researchgate.net/publication/387694577_Enhancing_DevSecOps_Securing_CICD_Pipelines_on_Cloud_Platforms

Campbell, Larry. (2021). DevSecOps: Integrating Security into DevOps. https://www.researchgate.net/ publication/ 392104352_DevSecOps_Integrating_Security_into_DevOps

Charoenwet, W., Thongtanunam, P., Pham, V.-T., & Treude, C. (2024). An empirical study of static analysis tools for secure code review. *arXiv*. https://doi.org/10.48550/arXiv.2407. 12241

Li, J., & Li, H. (2025). Evolution of application security based on OWASP Top 10 and CWE/SANS Top 25 with predictions for the 2025 OWASP Top 10. *Proceedings of 2025 IEEE 5th International Conference on Information Communication and Technology (ICICT)*, 1178–1183. https://doi.org/10.1109/ICICT 64420.2025.11004742

Pochu, S., & Kathram, S. (2024). Integrating security requirements into software development: A comprehensive approach to secure software design Bulletin of Engineering Science and Technology (BESTEC). Vol. 01, No. 03, 2024: 60–76

Publication, Research. (2019). Integrating Security Into the DevOps Process (DevSecOps). *SSRN Electronic Journal*. 4. 269–281

Rangnau, T., Buijtenen, R., Fransen, F., & Turkmen, F. (2020). Continuous security testing: A case study on integrating dynamic security testing tools in CI/CD pipelines. *Enterprise Distributed Object Computing Conference (EDOC)*, 145–154. https://doi.org/10.1109/EDOC49727.2020.00026

Sharma, S., & Chatterjee, S. (2021). DevSecOps: Integrating security practices into DevOps https://www.researchgate.net/publication/383334897_Integrating_Security_Into_the_DevOps_Process_DevSecOps

Yadati, N. S. P. K. (2021). Integrating dynamic security testing tools into CI/CD pipelines: A continuous security testing case study. *International Journal of Science and Research (IJSR), 10*, 1403–1405. https://doi.org/10.21275/SR24615152732

INTEGRATION OF SOURCE CODE ANALYSIS TOOLS INTO THE INNOVATIVE DEVSECOPS METHODOLOGY

Oleksii Duda[2], Mykola Orlov[1], Ihor Pavliv[3]

[1]Ternopil Ivan Puluj National Technical University,
Department of Computer Science, Ternopil, Ukraine
[2, 3] Lviv Polytechnic National University,
Department of Information Systems and Networks, Lviv, Ukraine
[1] Email: oleksij.duda@gmail.com, ORCID: 0000-0003-2007-1271
[2] Email: orlovnv86@gmail.com, ORCID: 0009-0007-9835-6177
[3] Email: micky.ukr@gmail.com, ORCID: 0009-0003-0957-0843

The article examines the relevance of integrating source code analysis tools, specifically Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST), into modern secure software development processes based on the innovative DevSecOps methodology. A review of scientific approaches and current practices for integrating security tools into CI/CD pipelines is provided, analyzing the advantages and limitations of SAST and DAST, as well as outlining trends in the development of combined security methods. The article presents a generalized model for integrating these tools into a DevSecOps environment and experimental research results on the effectiveness of using SAST and DAST individually and in combination. Practical recommendations are formulated for optimizing the selection of tools and improving software security while maintaining the performance of CI/CD processes.

Keywords: DevSecOps, CI/CD, SAST, DAST, software security, automated testing, integration of security tools.