COMPUTER SYSTEMS AND NETWORKS

Vol. 6, No. 2, 2024

DOI: https://doi.org/10.23939/csn2024.02.131

УДК 004.02; 004.05; 004.09

РОЛЬ, ПРОБЛЕМИ ТА МЕТОДИ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ БЕЗПЕКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

М. В. Максимович, Л. З. Мичуда

Національний університет "Львівська політехніка",
кафедра безпеки інформаційних технологій
E-mail: maksym.v.maksymovych@lpnu.ua, lesia.z.mychuda@lpnu.ua

У сучасному світі, де інформаційна безпека стає ключовим елементом діяльності будь-якої організації, автоматизація тестування безпеки програмного забезпечення стає як ніколи важливою. Успіх додатку напряму залежить від його стабільності, надійності та безпеки, тому належне впровадження механізмів контролю додатку є вкрай необхідним. Збільшення кількості кіберзагроз та зростання складності програмних систем надають цій темі ще більшої актуальності.

Основна роль автоматизації полягає в забезпеченні швидкого та ефективного виявлення потенційних вразливостей додатку на етапі його розробки. Попри очевидні переваги, існує низка проблем, що ускладнюють впровадження автоматизованих рішень, серед яких – складність налаштування, високі витрати на впровадження та підтримку, відсутність експертизи в області, відсутність пріоритетності.

У статті розглянуто роль автоматизації тестування безпеки програмного забезпечення, проблематику, методи та засоби тестування, способи їх поєднання для більшої ефективності. На основі аналізу запропоновано архітектурне рішення, яке забезпечує швидке, надійне та регулярне тестування безпеки додатку на різних етапах його життєвого циклу, що значно збільшує імовірність його стійкості до різних вразливостей.

Ключові слова: програмне забезпечення, кібербезпека, автоматизація, тестування, контейнеризація, захист.

Вступ

У сучасному світі програмне забезпечення все більше стає невід'ємним інструментом, що легко інтегрується в наші повсякденні справи. Від комунікацій і розваг до бізнесу й освіти цифрові додатки стають невід'ємним елементом в будь-якій сфері. Вони дають користувачам можливість отримувати доступ до інформації, спілкуватися та виконувати різні завдання зручно та ефективно. Незалежно від того, чи йдеться про управління фінансами, покупки в мережі «Інтернет» чи віддалену співпрацю, програмне забезпечення докорінно змінює методи взаємодії з навколишнім світом. Щоб задовольнити глобальні потреби користувачів щодо зберігання, пошуку та обробки інформації, програмне забезпечення повинно функціонувати правильно та надійно, бути простим, безпечним і придатним для використання [1]. Це створює значні виклики для розробників, оскільки з активним розвитком програмного забезпечення так само активно розвиваються нові методи його злому, відповідно розробники повинні інвестувати ресурси не тільки у розробку основного функціоналу, але й у забезпечення належного рівня безпеки додатку.

Огляд літературних джерел

Зі стрімким розвитком інтернет-технологій з кожним днем все більш популярними стають додатки, які надають різні сервіси, такі як онлайн-магазини, інтернет-банкінг та інші, які значно спрощують життя кожного з нас [2]. Однак ці технології створюють не тільки зручності у нашому повсякденному житті чи навчанні, але й певні безпекові ризики. Відсутність належного рівня безпеки додатку може призвести до значних проблем для користувача, таких як втрати фінансових ресурсів, втрати конфіденційної інформації, репутаційних збитків тощо.

Враховуючи зростаючу складність систем програмного забезпечення, нові сфери застосування, динамічні та часто критичні умови роботи, розподілену природу багатьох систем програмного забезпечення, а також конкурентний тиск на постачальників програмного забезпечення, створення безпечних систем стає дуже складним завданням [3].

У статті [4] розглядаються основні виклики впровадження забезпечення безпеки у процеси розробки та операційної діяльності. Автори підкреслюють важливість автоматизації безпеки та безперервної оцінки безпеки. Як зазначається в статті, однією з ключових проблем є досягнення балансу між швидкістю доставки програмного забезпечення та забезпеченням його безпеки.

В статті [5] зазначається, що мінімізація вразливостей додатку на етапі розробки є важливим кроком під час проєктування та випуску програмного забезпечення в умовах постійної присутності загроз безпеці. Мінімізація загроз безпеці додатку може бути виконана шляхом використання різних інструментів сканування коду на вразливості й впровадження їх на різних етапах життєвого циклу розробки.

Будь-яке програмне забезпечення може піддаватися атакам зі сторони зловмисників, що можуть включати в себе атаки на різні компоненти системи, такі як бази даних, інфраструктура системи чи безпосередньо додаток, з яким працює кінцевий користувач.

Згідно з даними опитування Gartner, 75 % атак на програмне забезпечення відбуваються на рівні додатку, а не на інфраструктурному рівні [6], тим не менш, багато компаній продовжують витрачати багато зусиль та фінансових ресурсів на безпеку інфраструктури, в той же час нехтуючи безпекою на рівні додатків, що створює відкритий простір для атак, яким можуть скористатись зловмисники [7]. Як наслідок, відсутність достатнього рівня підтримки безпеки на рівні додатку може призвести до відсутності доступу до сервісу, некоректної роботи або навіть втрати чи пошкодження інформації, що в свою чергу створить суттєві репутаційні та фінансові збитки для компанії, зменшення кількості користувачів тощо.

Постановка задачі

Ця стаття присвячена огляду важливості, проблематики та методів забезпечення автоматизованого тестування безпеки програмного забезпечення.

Метою цього дослідження є визначення ролі та аналіз проблематики автоматизації тестування програмного забезпечення, аналіз сучасних типів тестування, методів автоматизації та способів їх поєднання для забезпечення ефективного тестування безпеки додатку в контексті мінімізації витрат на впровадження та підтримку.

Актуальність

Одним з методів забезпечення надійності роботи додатку є проведення регулярного автоматизованого тестування, яке може включати в себе тестування безпеки. В сучасному світі технологій існує багато методів забезпечення безпеки додатків, таких як проведення аудиту безпеки сторонніми компаніями на певному етапі розробки продукту або проведення регулярних тестів на проникнення спеціалістами з безпеки. Тим не менш, кожне таке тестування безпеки чи проведення аудиту тягне за собою залучення значних ресурсів, як людських так і фінансових, тому питання автоматизації цього процесу та простої інтеграції його в життєвий цикл розробки додатку є дуже актуальними.

Автоматизація дозволить виконувати тестування безпеки на регулярній основі при кожній зміні додатку, що дозволить виявляти вразливості на ранніх стадіях розробки та максимально зменшить можливість випуску нової версії додатку з вразливостями. Також регулярне автоматизоване тестування не допустить появи тих самих вразливостей після того, як вони були виправлені розробниками, оскільки при коректному налаштуванні випуск нової версії продукту неможливий без проведення тестування безпеки, яке повинно виявити вразливість.

Сьогодні існує безліч інструментів розробки програмного забезпечення, з кожним днем появляються все нові і нові бібліотеки, які активно використовують розробники, які також можуть включати в себе вразливості безпеки, що ще більше підкреслює необхідність комплексного автоматизованого та регулярного процесу тестування безпеки.

Проблематика тестування безпеки програмного забезпечення

Налаштування коректного тестування безпеки є непростим завданням, оскільки потребує від розробника певних знань в області захисту інформації, наявних інструментів тестування та можливості їхньої легкої інтеграції безпосередньо в проєкт, над яким працює розробник. Зазвичай розробники не володіють великим обсягом навиків в області автоматизації тестування безпеки програмного забезпечення, більше зосереджуючись на доменній експертизі чи експертизі в контексті вибраної мови програмування. Також, як зазначається в статті [8], ідея використання інструментів автоматизованого тестування безпеки звучить ефективно та багатообіцяюче, проте багато інструментів мають певні недоліки, такі як орієнтація на певні мови програмування чи часті хибні позитивні результати, що змушує розробників скептично ставитись до цих рішень та в кінцевому результаті відмовлятись від них.

Отже, коли перед розробником постає завдання налаштування автоматизованого тестування безпеки, він повинен провести певні дослідження в цій області, такі як визначення основних методологій тестування безпеки, аналіз наявних інструментів та можливості їхньої інтеграції в проєкт, підготовка технічної документації на основі проведених досліджень, реалізація визначеного підходу тощо. Всі ці дії потребують значних затрат часу, що уповільнюють розробку основного функціоналу продукту, тягнуть за собою додаткові фінансові витрати та зменшують конкурентність продукту на ринку. Альтернативним методом для компанії може бути використання комерційних інструментів чи співпраця з експертом в області кіберзахисту, однак це також потребує додаткових фінансових витрат, які компанія може застосувати більш ефективно.

Іншою проблемою є пріоритетність цього завдання. Зазвичай основними пріоритетами під час роботи над продуктом є розробка функціоналу, якого вимагає ринок, швидке отримання результатів, перевірка гіпотез та отримання доходів, що відводить питання безпеки на інший план та позбавляє його пріоритетності. Однак зі стрімким розвитком інтернет-технологій та інтеграцією в наше повсякденне життя питання безпеки також стає все більш актуальним, швидке налаштування автоматизованого тестування безпеки стає пріоритетним завданням, тому дослідження в області підвищення ефективності цього процесу є актуальними так, як ніколи.

Потенційні атаки

Як зазначалось раніше, на цей момент вебдодатки є основною метою зловмисників, оскільки є найбільш популярним вибором при розробці різних сервісів. Розрізняють два основних типи атак на вебдодатки:

– атаки на рівні клієнта – як ми можемо зрозуміти з назви, це атаки, які можуть бути виконані відносно клієнтської частини вебдодатку. Атаки на рівні клієнта виконуються шляхом виконання певного роду злоякісних вебскриптів відносно вебсторінки з метою пошкодження інформації або викрадення чутливої інформації, такої як приватні ключі сесії тощо;

– атаки на рівні сервера – це атаки, метою яких є серверна частина застосунку. Під час атаки на сервер метою зловмисника можуть бути точки доступу до сервера (API), які зловмисник може викликати з різними даними, відповідно аналізуючи відповідь сервера під час кожного виклику. Зокрема, сервери додатку, які мають певні вразливості, можуть надавати відповідь з певними

конфіденційними даними залежно від методу виклику точки доступу або ж також модифікувати чи навіть видалити певну інформацію, критичну для роботи застосунку [9].

Одним з основних проєктів, який спеціалізується на покращенні безпеки вебдодатків, є OWASP (Open Web Application security project). OWASP – це організація, яка працює над підвищенням обізнаності про веббезпеку та, зрештою, покращує її. Топ-10 OWASP – це популярний документ, що представляє широкий консенсус щодо найбільш критичних ризиків для безпеки вебдодатків [10]. Цей документ оновлюється щороку та включає в себе такі основні типи атак, як атаки ін'єкції, атаки пошкодженої аутентифікації та менеджменту сесії, атака міжсайтового скриптингу та інші (рис. 1).

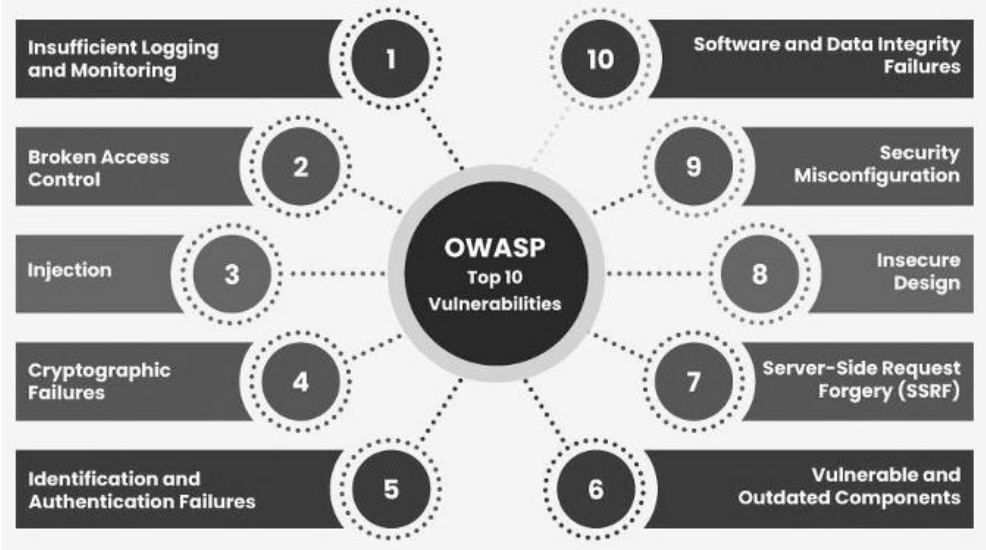

Рис. 1. Основні типи атак вебдодатків за версією OWASP top 10

Розглянемо детальніше найбільш популярні атаки, зазначені вище [11].

Ін'єкції – це атаки, які полягають у надсиланні певного злоякісного коду до виконання. Припустимо, на вебсторінці є поле для вхідного тексту, де можна ввести будь-яку інформацію, яка буде збережена в базу даних. Якщо це поле вразливе для атаки типу ін'єкція, то зловмисник може видалити всю інформацію з бази даних шляхом відправки простого скрипта відповідно до бази даних.

Порушення аутентифікації чи менеджменту сесій – у випадку атаки в області аутентифікації чи менеджменту сесії зловмисник намагається втрутитись в механізми керування доступу до вебдодатків, щоб отримати доступ до нього в ролі користувача для зловмисних дій.

Атаки міжсайтового скриптингу полягають в інтеграції злоякісного коду у відповіді сервера, яка може виконуватись браузером, відповідно спричиняючи злоякісні дії.

Також нерідко причиною успішної атаки може стати зберігання чутливої інформації розробником у відкритому коді. Як відомо, клієнтський код вебдодатку є публічним, оскільки браузер отримує як результат відповіді сервера, і відповідно в результаті його виконання користувач бачить вебсторінку, з якою може працювати. Отже, якщо у браузері користувача встановлений злоякісний додаток, то він з легкістю може проаналізувати код додатку, отримати ключ, який розробник зберіг у відкритому коді, та використати його в подальших атаках [12].

Типи тестування безпеки

Для боротьби зі згаданими вище атаками існують різні типи тестування, основними серед яких є такі, як методи аналізу статичного коду, динамічний, інтерактивний та композитний аналіз (рис. 2). Методи статичного аналізу можуть бути корисні для усунення проблеми зберігання

чутливої інформації, яка згадується вище, шляхом статичного аналізу коду. Методи динамічного та інтерактивного аналізу можуть бути корисні для виявлення атак зі списку OWASP топ-10.

Рис. 2. Типи тестування безпеки додатку

Розглянемо детальніше кожен з вищезгаданих типів тестування.

Статичний аналіз (SAST) сканує вихідний код програми, інструмент тестування білої скриньки, він визначає першопричину вразливостей і допомагає усунути основні недоліки безпеки [13]. Рішення SAST аналізують програму "зсередини" і не використовують запущену систему для виконання сканування [14]. Серед основних інструментів SAST можна згадати такі, як SonarQube, Veracode, Brakeman [15]. Однак інструменти SAST не здатні ідентифікувати вразливості поза кодом. Наприклад, вразливості, виявлені в сторонньому API, не будуть виявлені SAST і вимагатимуть динамічного тестування безпеки додатків (DAST).

Динамічний аналіз (DAST) – це інструменти «чорної скриньки», які дозволяють аналізувати запущену програму, атакуючи всі вхідні дані зовнішнього джерела вебпрограми [15]. На першому етапі вони намагаються виявити (сканувати) інтерфейс вебпрограми, тобто всі можливі вихідні дані програми. Після фази сканування інструменти виконують рекурсивну атаку зі зловмисним корисним навантаженням на всі виявлені вихідні дані вебпрограми. Одним з основних безкоштовних інструментів динамічного тестування є ZED Attack Proxy – інструмент відкритого коду, запропонований компанією OWASP. Серед комерційних можна виділити такі, як Acunetix, Netsparker, InsightAppSec [16].

Інтерактивний аналіз (IAST) аналізує код на наявність вразливостей, коли додаток запускається за допомогою автоматичного тестування, спеціаліста з підтримки якості або будь-якої дії, що «взаємодіє» з функціями додатка [17]. Як інструменти інтерактивного аналізу можна згадати такі інструменти, як Checkmarx чи Seeker Interactive Application Security Testing.

Композитний аналіз (SCA) використовується для перевірки на вразливості сторонніх компонентів системи, таких як допоміжні бібліотеки, бібліотеки з відкритим кодом тощо. Серед основних інструментів композитного аналізу можна згадати такі, як OWASP Dependency-check, RetireJS чи Safety [18].

Враховуючи різноманітність типів тестування безпеки та велику кількість потенційних загроз для вебдодатків, ми можемо вважати необхідним застосування комплексного підходу тестування безпеки, який базується на поєднанні згаданих вище підходів [19], що повноцінно інтегрується в цикл розробки програмного забезпечення та виконує тестування безпеки при кожній зміні додатку, на певному етапі, залежно від типу тестування та архітектури додатку.

Оцінка ефективності комбінованого тестування безпеки

Як зазначалось раніше, для ефективного тестування необхідне поєднання декількох інструментів тестування безпеки. В статті [19] проводилось дослідження, в рамках якого визначалась ефективність поєднання інструментів. В рамках дослідження використовували такі інструменти:

- SAST: Fortify SCA (Microfocus), FindSecurityBugs (OWASP);
- DAST: OWASP ZAP (OWASP), Arachni (Tasos Lassos);
- IAST: Contrast Community Edition (Contrast Security), CIAs (Checkmarx).

Для дослідження було заздалегідь підготовлено проєкт, який включав в себе спеціально підготовлені вразливості (рис. 3)

CWE	Vulnerability Types	Number of Test Cases
78	Command Injection	40
643	Xpath Injection	20
79	Cross Site Scripting (XSS)	40
90	LDAP Injection	20
22	Path Traversal	40
89	SQL Injection	40
614	Secure Cookie flag	20
501	Trust Boundary Violation	20
330	Weak Randomness	40
327	Weak Cryptographic	20
328	Weak Hashing	20
	Total	320

Рис. 3. Вразливості, включені в проєкт

Як основні метрики, використані в дослідженні, можна виділити такі:

TPR (true positive rate) (1) – відношення виявлених вразливостей до кількості, яка реально існує в коді.

$$TP / (TP + FN), \quad (1)$$

де TP (true positives) – це кількість справжніх вразливостей, виявлених у коді, а FN (false negatives) – це загальна кількість існуючих вразливостей, не виявлених у коді.

FPR (false positive rate) (2) – коефіцієнт помилкових тривог для вразливостей, яких насправді немає в коді.

$$TN / (TN + FP), \quad (2)$$

де TN (true negatives) – це кількість невиявлених вразливостей, яких немає в коді, а FP (false positives) – загальна кількість виявлених вразливостей, яких немає в коді.

В рамках дослідження було проаналізовано ефективність застосування методів поодинці та в комбінації 2 та більше інструментів (рис. 4).

На рис. 4 можна побачити середню ефективність інструментів у комбінації. Коефіцієнт TPR зростає зі збільшенням кількості інструментів у комбінації, досягаючи практично значення 1 у всіх вразливостях. Коефіцієнт FPR збільшується в основному в кількох типах вразливостей.

Також дослідження показало середній час, необхідний для тестування обраного проєкту з використанням кожного метода (див. таблицю).

Як видно з таблиці, найбільш часозатратним методом тестування є динамічне тестування, оскільки час тестування займає більше 10 годин, в той час як тестування інтрументами статичного/інтерактивного аналізу займає декілька хвилин чи десятків хвилин.

Отже, на основі цього аналізу можна зробити висновок, що ефективне поєднанання методів тестування можливе при реалізації регулярного тестування методами SAST/IAST при кожному оновленні ПЗ, оскільки дані типи тестування займають відносно небагато часу, в той час як інструменти DAST можливо застосовувати з періодичністю раз в тиждень чи навіть раз в місяць, оскільки є досить часозатратним та відповідно ресурсозатратним методом тестування. Також аналіз

показує, що поєднання інструментів збільшує ефективність тестування. Проте, як саме їх поєднувати, залежить від ресурсів та потреб конкретного проєкту.

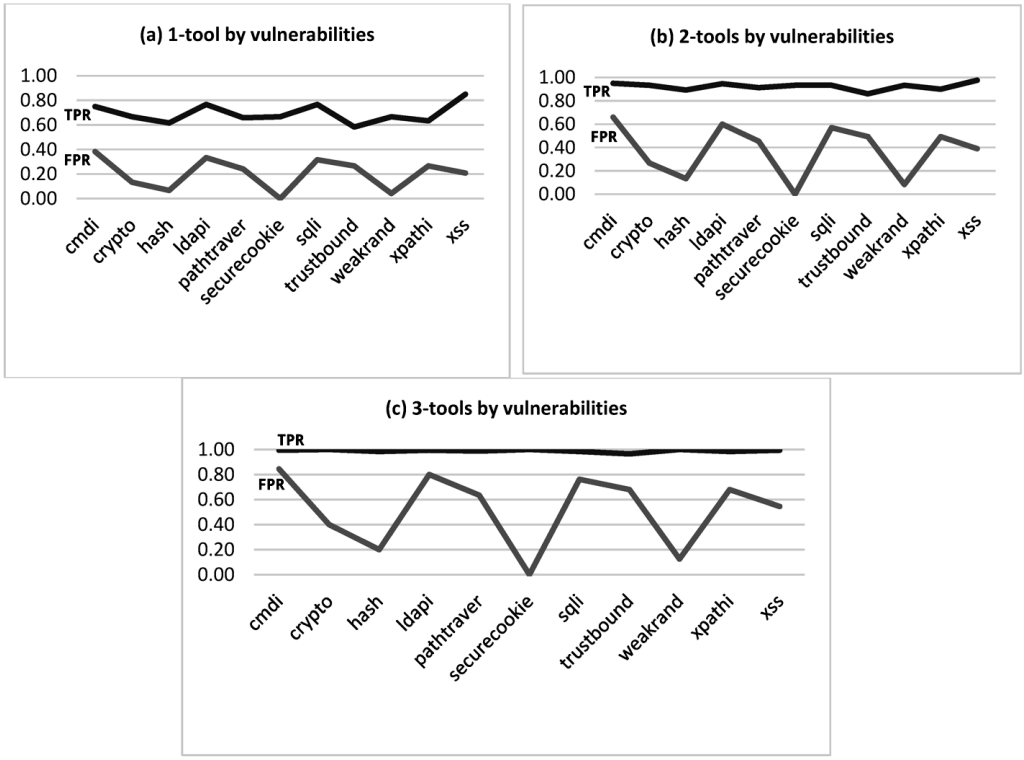

Рис. 4. Результат поодинокого та комбінованого застосування методів тестування

Час тестування проєкту з використанням вказаних інструментів

Інструмент	Час на тестування
FindSecurityBugs	22 хв
Fortify SCA	19 хв
OWASP ZAP	36 год
Arachni	13 год
Contrast Community Edition	2 хв
CIAs	4 хв

Методи тестування безпеки

Одним з методів ефективного поєднання цих методологій для проведення комплексного тестування безпеки та легкої інтеграції його в механізм безперервної інтеграції додатку є використання інструментів контейнеризації.

Контейнеризація [20] – це упакування програми разом з її залежностями, такими як бібліотеки та інші бінарні файли, в один єдиний блок, що називають контейнером. Контейнери забезпечують стабільні середовища розробки та розгортання програм, вони ізольовані один від одного та можуть працювати на будь-якій платформі, що підтримує технологію контейнерів.

Контейнеризація має багато переваг, включаючи портативність, підвищену безпеку та покращене використання ресурсів. Контейнери є легкими і можуть бути швидко розгорнуті, а також легко масштабуватись за потребою. Контейнеризація дозволяє програмі швидко та надійно працювати від одного середовища до іншого, без необхідності окремо встановлювати та налаштовувати залежності. Одним з провідних інструментів контейнеризації на цей момент є Docker – відкрита платформа, яка дозволяє розробникам автоматизувати розгортання, масштабування та управління

застосунками в легких і портативних контейнерах. Його використання можливе для забезпечення контейнеризації будь-якого елемента системи, чи то бази даних, брокера повідомлень, серверів бізнес логіки, чи власне контейнерів, які забезпечуватимуть автоматизоване тестування безпеки.

Отже, при використанні методів контейнеризації для побудови інфраструктури додатку ми також можемо використовувати контейнеризацію під час використання різних інструментів статичного, динамічного та інтерактивного тестування, що дозволить розгортати проєкт шляхом виконання єдиної команди та проводити повноцінне, комплексне тестування безпеки з використанням вищезгаданих методологій також з виконанням єдиної команди. Приклад архітектури додатку з відповідними контейнерами для забезпечення тестування зображено на рис. 5.

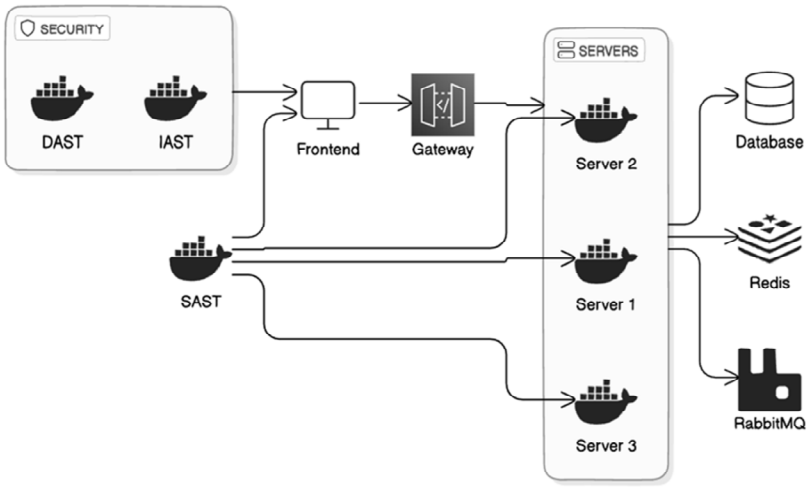

Рис. 5. Приклад архітектури вебдодатку з використанням композиції типів тестування безпеки програмного забезпечення

Реалізація цієї архітектури забезпечить автоматизоване, швидке та надійне тестування безпеки програмного забезпечення при кожній ітерації, з мінімальними затратами ресурсів на підтримку, що мінімізує ризики виникнення вразливостей, і тим самим збільшить надійність продукту в цілому.

Результати дослідження

В рамках цього дослідження було виявлено, що автоматизація тестування безпеки програмного забезпечення є критичним інструментом для підвищення надійності та стійкості програмних систем до кіберзагроз. Зі стрімким розвитком цифрових технологій відповідно появляються все нові методи та засоби злому, тому дослідження в області покращення методів протидії цим загрозам є актуальними.

Дослідження також показало труднощі впровадження автоматизованих рішень для підвищення безпеки додатків, серед яких складність налаштування, недостатня експертиза розробників в області кіберзахисту, відсутність пріоритетності, вартість застосування, що в свою чергу доводить необхідність покращення методів інтеграції в контексті мінімізації часу та витрат на впровадження та підтримку.

Також було розглянуто основні методи тестування безпеки програмного забезпечення, серед яких динамічний, інтерактивний, статичний та композитний аналіз, та потенційні шляхи інтеграції цих методів тестування в проєкт. На основі аналізу було запропоновано архітектурне рішення, реалізація якого допоможе суттєво знизити витрати часу на впровадження та підтримку регулярного автоматизованого тестування безпеки програмного забезпечення, що матиме позитивний вплив на стійкість програмного продукту до можливих кібератак.

Отже, результати досліджень підтверджують потребу подальших удосконалень в області автоматизованого тестування безпеки програмного забезпечення для підвищення стійкості програмних продуктів до можливих кіберзагроз.

Висновки

Програмне забезпечення є важливою складовою сучасного життя, забезпечуючи ефективність та інновації у різних галузях, таких як медицина, транспорт, освіта, логістика, бізнес, військова справа та багато інших. Завдяки автоматизації рутинних завдань і оптимізації робочих процесів програмне забезпечення дозволяє компаніям підвищити свою продуктивність, конкурентоспроможність і, відповідно, фінансові показники.

Разом з перевагами, які програмне забезпечення несе для кожного користувача, його використання передбачає також можливі безпекові ризики, які можуть призвести до суттєвих репутаційних та фінансових втрат, якщо питання безпеки не буде враховано на етапі розробки. На жаль, безпека часто залишається поза увагою через обмеженість ресурсів або недостатню пріоритетність цього питання.

Інтеграція автоматизованих методів тестування безпеки у розробку програмного забезпечення сприяє створенню більш надійних продуктів, що відповідають вимогам сучасного світу, де кіберзагрози є постійним викликом. Це підвищує довіру користувачів, зберігає репутацію компаній і захищає їхні інвестиції.

У цій статті висвітлюється критична роль та проблеми автоматизації тестування безпеки програмного забезпечення як ефективного засобу запобігання загрозам. Розглянуто основні типи тестування безпеки, які можуть бути інтегровані у процес розробки з мінімальними витратами на впровадження та підтримку. Пропонований підхід дозволяє забезпечити базовий рівень безпеки для додатків, що значно знижує ймовірність виникнення вразливостей, тим самим збільшуючи імовірність успіху продукту на ринку в цілому.

Список літератури

1. Kokol P. (2022). Software Quality: How Much Does It Matter? Electronics, 11(16), Article 16. https://doi.org/10.3390/electronics11162485

2. Web Application Software Engineering Technology and Process | IEEE Conference Publication | IEEE Xplore. (n.d.). Retrieved May 3, 2024, from https://ieeexplore.ieee.org/document/9421250

3. Mirakhorli M., Galster M. & Williams L. (2020). Understanding Software Security from Design to Deployment. ACM SIGSOFT Software Engineering Notes, 45(2), 25–26. https://doi.org/10.1145/3385678.3385687

4. Rajapakse R. N., Zahedi M., Babar M. A. & Shen H. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. Information and Software Technology, 141, 106700. https://doi.org/10.1016/j.infsof.2021.106700

5. Concea-Prisăcaru A.-I., Niţescu T.-A. & Sgârciu V. (2023). SDLC AND THE IMPORTANCE OF SOFTWARE SECURITY. UPB Scientific Bulletin, Series C: Electrical Engineering and Computer Science, 85(1), 117–130. Scopus.

6. Song B., Sun L. & Qin Z. (2022). Design of Web Security Penetration Test System Based on Attack and Defense Game. Scientific Programming, 2022. Scopus. https://doi.org/10.1155/2022/8645969

7. Dalai A. K. & Jena S. K. (2017). Neutralizing SQL Injection Attack Using Server Side Code Modification in Web Applications. Security and Communication Networks, 2017, e3825373. https://doi.org/10.1155/2017/3825373

8. Wu Y., Su J., Moran D. D. & Near C. D. (2023). Automated Software Testing Starting from Static Analysis: Current State of the Art (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2301.06215

9. Nagendran K., Adithyan A., Chethana R., Camillus P. & Bala Sri Varshini, K. B. (2019). Web application penetration testing. International Journal of Innovative Technology and Exploring Engineering, 8(10), 1029–1035. Scopus. https://doi.org/10.35940/ijitee.J9173.0881019

10. Brito T., Ferreira M., Monteiro M., Lopes P., Barros M., Santos J. F. & Santos N. (2023). Study of JavaScript Static Analysis Tools for Vulnerability Detection in Node.js Packages. IEEE Transactions on Reliability, 72(4), 1324–1339. Scopus. https://doi.org/10.1109/TR.2023.3286301

11. Fredj O. B., Cheikhrouhou O., Krichen M., Hamam H. & Derhab A. (2021). An OWASP Top Ten Driven Survey on Web Application Protection Methods. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 12528 LNCS, 235–252. Scopus. https://doi.org/10.1007/978-3-030-68887-5_14

12. Higuera J.-R., Bermejo J., Montalvo J. A., Villalba J., & Pez J. (2020). Benchmarking Approach to Compare Web Applications Static Analysis Tools Detecting OWASP Top Ten Security Vulnerabilities. Computers, Materials and Continua, 64, 1555–1577. https://doi.org/10.32604/cmc.2020.010885

13. Li J. (2020). Vulnerabilities mapping based on OWASP-SANS: A survey for static application security testing (SAST). Annals of Emerging Technologies in Computing, 4(3), 1–8. Scopus. https://doi.org/10.33166/AETiC.2020.03.001

14. Yang J., Tan L., Peyton J., & A Duer K. (2019). Towards Better Utilizing Static Application Security Testing. 51–60. Scopus. https://doi.org/10.1109/ICSE-SEIP.2019.00014

15. Singh R., Kumar Gupta M., Patil D. R., & Maruti Patil S. (2024). Analysis of Web Application Vulnerabilities using Dynamic Application Security Testing. 2024 IEEE 9th International Conference for Convergence in Technology, I2CT 2024. Scopus. https://doi.org/10.1109/I2CT61223.2024.10543484

16. OWASP DevSecOps Guideline—V-0.2 | OWASP Foundation. (n.d.). Retrieved October 14, 2024, from https://owasp.org/www-project-devsecops-guideline/latest/02b-Dynamic-Application-Security-Testing.html

17. Pan Y. (2019). Interactive Application Security Testing. 2019 International Conference on Smart Grid and Electrical Automation (ICSGEA), 558–561. 2019 International Conference on Smart Grid and Electrical Automation (ICSGEA). https://doi.org/10.1109/ICSGEA.2019.00131

18. OWASP DevSecOps Guideline—V-0.2 | OWASP Foundation. (n.d.). Retrieved October 14, 2024, from https://owasp.org/www-project-devsecops-guideline/latest/02d-Software-Composition-Analysis.html

19. Tudela F. M., Higuera J.-R. B., Higuera J. B., Montalvo J.-A. S. & Argyros M. I. (2020). On combining static, dynamic and interactive analysis security testing tools to improve owasp top ten security vulnerability detection in web applications. Applied Sciences (Switzerland), 10(24), 1–26. Scopus. https://doi.org/10.3390/app10249119

20. Aparo C., Bernardeschi C., Lettieri G., Lucattini F. & Montanarella S. (2023). An Analysis System to Test Security of Software on Continuous Integration-Continuous Delivery Pipeline. 58–67. Scopus. https://doi.org/10.1109/EuroSPW59978.2023.00012

ROLE, PROBLEMS, AND METHODS OF SOFTWARE SECURITY TESTING AUTOMATION

M. Maksymovych, L. Mychuda

Lviv Polytechnic National University,
Department of Information Technologies Security
E-mail: maksym.v.maksymovych@lpnu.ua, lesia.z.mychuda@lpnu.ua

In the modern world, where information security becomes a key element of any organization's operations, software security testing automation is more important than ever. The success of an application directly depends on its stability, reliability, and security, which makes the proper implementation of control mechanisms critical. The increase in cyber threats and the growing complexity of software systems make this topic even more relevant.

The main role of automation is to provide quick and efficient detection of potential application vulnerabilities during its development phase. Despite its obvious advantages, several issues complicate the implementation of automated solutions, such as the complexity of configuration, high implementation and maintenance costs, lack of expertise in the field, lack of prioritization.

This article examines the role of software security testing automation, challenges, methods, tools for testing, and ways to combine them for greater efficiency. Based on the analysis, an architectural solution is proposed that ensures quick, reliable, and regular security testing of the application at different stages of its lifecycle, significantly increasing the likelihood of its resistance to various vulnerabilities.

Keywords: software, cybersecurity, automation, testing, containerization, protection.