КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 1 (21), 2023
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2023.21.193210
УДК 004.4’2:004.56

Спасітєлєва Світлана Олексіївна
канд. ф.-м. наук, доцент, доцент кафедри комп’ютерних наук та математики
місце роботи: Київський університет імені Бориса Грінченка, м. Київ, Україна
ORCID ID 0000-0003-4993-6355
s.spasitielieva@kubg.edu.ua

Чичкань Іван Васильович
канд. ф.-м. наук, доцент, доцент кафедри інформаційних систем та технологій
місце роботи: Київський національний університет імені Тараса Шевченка, м. Київ, Україна
ORCID ID 0000-0002-0854-389X
Chychkan@knu.ua

Шевченко Світлана Миколаївна
канд. пед. наук, доцент, доцент кафедри комп’ютерних наук та математики
місце роботи: Київський університет імені Бориса Грінченка, м. Київ, Україна
ORCID ID 0000-0002-9736-8623
s.shevchenko@kubg.edu.ua

Жданова Юлія Дмитрівна
канд. ф.-м. наук, доцент, доцент кафедри комп’ютерних наук та математики
місце роботи: Київський університет імені Бориса Грінченка, м. Київ, Україна
ORCID ID 0000-0002-9277-4972
y.zhdanova@kubg.edu.ua

РОЗРОБКА БЕЗПЕЧНИХ КОНТЕЙНЕРНИХ ЗАСТОСУНКІВ З МІКРОСЕРВІСНОЮ АРХІТЕКТУРОЮ

Анотація. У статті розглянуто підходи до розробки програмного забезпечення, які дозволяють завдяки засобам автоматизації та методам гнучкої розробки створювати складні контейнерні застосунки з мікросервісною архітектурою. Розвиток хмарних технологій, глобальна стратегія контейнерізації, модернізація архітектури застосунків, підвищення вимог до безпеки призвели до зміни методології розробки програм. Метою дослідження є визначення підходів до підвищення швидкості розробки, безпеки та якості програмного коду контейнерних застосунків шляхом впровадження принципів безпеки та інструментів автоматизації на всіх етапах життєвого циклу. Розглянуто особливості та перспективи розвитку мікросервісних застосунків, розгорнутих у контейнерному середовищі. Визначені переваги контейнерної інфраструктури: мобільність, масштабованість, додатковий рівень безпеки сервісу. Використання контейнерів забезпечує ізольоване середовище для запуску мікросервісу що зменшує ризик поширення вразливостей безпеки, спрощує взаємодію мікросервісів. У роботі визначені проблеми безпеки мікросервісних застосунків, головні вразливості, пов’язані з використанням контейнерів. Показано, що методологія DevSecOps дозволяє реалізувати сучасну практику безперервної інтеграції, доставки, розгортання застосунку та інтегрувати засоби безпеки в усі етапи життєвого циклу. DevSecOps описує процеси розробки та процеси розгортання і експлуатації програм із застосуванням практик Security as Code та Infrastructure as Code відповідно. В роботі описана модель розробки та розгортання мікросервісних застосунків з контейнеризацією, визначені області безпеки, засоби контролю безпеки кожного етапу DevSecOps розробки. На базі цієї моделі визначено головні інструменти автоматизації контролю безпеки, які необхідно використовувати на всіх етапах конвеєру розробки та розгортання. У статті показано, що розглянута методика регламентує виконання визначених процедур безпеки на всіх етапах конвеєру, дозволяє скоротити час розробки і підвищити якість коду для контейнерних застосунків з мікросервісною архітектурою.

Ключові слова: мікросервісна архітектура; контейнер; DevSecOps; безпека додатків; вразливості безпеки.

ISSN 2663 - 4023

ВСТУП

Обсяг розроблюваного програмного забезпечення різного функціонального призначення та його складність зростають щороку. Наприклад, зростання ринку програмного забезпечення України в середньому складає 27% щороку (прогноз на 2023 складає 10%) [1]. Конкуренція на ринку висуває вимоги до підвищення швидкості та якості самого процесу розробки. Змінюється архітектура додатків, модернізується інфраструктура. Широке розповсюдження набуває сервіс-орієнтована архітектура програмних продуктів. Архітектура мікросервісів стає стандартом для постійно розгорнутих розподілених контейнерних середовищ. Сучасним трендом є принцип «security-first» – підвищені вимоги до безпеки таких програмних продуктів на всіх етапах життєвого циклу. Із зростанням складності та критичності програм, класичні методології розробки перестають задовольняти вимогам розробників і вимагають нових підходів до процесу розробки та застосування інструментальних засобів підтримки всіх етапів життєвого циклу програм. Відповідно змінюються методологія та підходи до етапів розробки, розгортання та вдосконалення сучасних програм.

Постановка проблеми. В роботі розглядаються методика, етапи, інструменти розробки додатків з мікросервісною архітектурою та розподіленою контейнерною інфраструктурою. Розглядається інтеграція принципів безпеки в процес неперервної розробки та вбудовування безпеки у застосунок на всіх етапах життєвого циклу для зведення до мінімуму вразливостей програмного коду. Визначається набір інструментів автоматизації всіх етапів життєвого циклу контейнерних застосунку з мікросервісною архітектурою.

Аналіз останніх досліджень і публікацій. Опису етапів життєвого циклу розробки програмного забезпечення присвячено багато публікацій закордонних та вітчизняних авторів [2]. Всебічному дослідженню стратегій, класичних моделей життєвого циклу (послідовна, ітераційна, спіральна), методів розробки програм присвячені роботи Юрдона Е., Лармана К., Хайсміта Дж., Фаулера М. Особливостям використання гнучкої розробки за принципом Agile присвячені роботи Кокберна А., Мартіна Р., Аппело Ю., Бека К., Кона М. [3]. У публікаціях закордонних і вітчизняних авторів велика увага приділяється питанням безпеки сучасних додатків, досліджуються підходи та методи інтеграції принципів безпеки в процес розробки. Методологія DevOps для швидкого створення та неперервного розвитку і розгортання програмного продукту досліджується та розвивається. Результатом є поява вдосконаленої методології DevSecOps, яка вбудовує безпеку в процес неперервної розробки та розгортання. Цим дослідженням присвячені роботи Вілсона Г., Шейна М., Кенднла Ж., Фарлі Д., Кіма Дж. та інших авторів [4], [5].

У наукових дослідженнях розглядаються питання створення гнучких, легко модифікованих, безпечних застосунків, на базі мікросервісної архітектури та контейнерів [6]. Дослідження з безпеки сучасних застосунків не встигають за динамічним розвитком технологій, модернізацією архітектури та моделями розвитку розподіленої інфраструктури застосунків. Тому необхідно розглянути цілісний підхід з визначенням методів, етапів, інструментів швидкої розробки та розгортання мікросервісних додатків у контейнерному середовищі з впровадженням усіх практик створення якісного та безпечного коду.

Метою статті є розробка методики впровадження принципів безпеки на всіх етапах розробки та розгортання застосунків з мікросервісною архітектурою в розподіленому контейнерному середовищі з метою скорочення часу розробки, підвищення безпеки, зменшення вразливостей та підвищення якості програмного коду.

ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

Розглянемо основні сучасні підходи, методології до організації процесів розробки та реалізації програмних рішень з впровадженням принципів безпеки на всіх стадіях життєвого циклу. Методології життєвого циклу розробки програмного забезпечення (SDLC – software development life cycle) еволюційно змінювалися з розвитком IT технологій та підвищенням складності програмних продуктів. Різноманітні ітераційні, спіральні методології SDLC дозволили розробникам швидше змінювати, вдосконалювати та адаптувати програмний код до вимог, хоча часто для окремих фаз застосовувалася загальна послідовна модель. Це дозволяло створювати мінімальний життєздатний продукт (MVP – minimum viable product), поступово додавати функціональні можливості а також переглядати вимоги користувачів з кожною ітерацією та адаптувати продукт за потреби. Сучасні гнучкі Agile [7] методології з безперервною інтеграцією та безперервним розгортанням (CI/CD – Continuous Integration/Continuous Delivery) [8] є розвитком ітераційної та спіральної методології. Розробка програмних продуктів розглядається як неперервний процес, який повторюється так часто, як це необхідно. Функціональність реалізується поступово, а не монолітним випуском для розгортання в обраному середовищі. Підхід DevOps є рушійною силою більшості сучасних проектів і використовується, щоб зробити розгортання та обслуговування невід'ємною частиною процесу розробки із застосуванням інструментів автоматизації всіх процесів [9].

Методи розробки розвивалися завдяки автоматизації конвеєрів побудови, тестування та розгортання, що дозволило швидко створювати та розгортати великі та складні програми. Фаза спеціального тестування безпеки стала головним вузьким місцем. При цьому важливість безпеки програмних продуктів зростала з розвитком IT-інфраструктури, появою веб-додатків та хмарних технологій. Виникла потреба вбудувати безпеку в кожен етап циклу, відповідно розвиваються підходи до створення життєвого циклу розробки безпечного програмного забезпечення (SSDLC – software development life cycle). На рисунку 1 представлена спрощена схема етапів SSDLC з врахуванням проблем безпеки.

Рис. 1. Етапи життєвого циклу розробки безпечного ПЗ (SSDLC)

Кожний етап SSDLC має вирішувати визначені задачі безпеки, які мають бути вбудовані безпосередньо в інструменти та робочі процеси цього етапу. Вимоги до безпеки визначаються на першому етапі разом із вимогами до функціональності та продуктивності. Аналіз та оцінка ризиків безпеки також мають проводитися на перших двох етапах. На етапі проектування проводиться аналіз поверхні атак, моделювання загроз, використовуються принципи Secure by Design які стають основним підходом до розробки для забезпечення безпеки. Код розробляється з дотриманням усіх вимог щодо безпечного кодування та підходу до безпеки як важливого аспекту якості коду. Обов'язковим етапом є тестування безпеки застосунку із застосуванням інструментів безпеки для виконання тестів, інтерпретації та координації виправлень. На етапі розгортання необхідно визначити та підтримувати безпечну конфігурацію додатку та інфраструктуру розгортання, здійснювати пошук вразливостей та моніторинг загроз безпеці.

Абстрактна високорівнева модель процесу розробки SSDLC має практичне втілення для гнучкої CI/CD розробки в середовищі з обмеженими ресурсами у вигляді методології DevSecOps [4], [10]. Безперервна інтеграція та безперервна доставка (CI/CD) – це сучасна практика розробки ПЗ, яка використовує автоматизовані етапи розробки та тестування для надійного та ефективного внесення змін до застосунку. Розробники використовують інструменти CI/CD для випуску нових версій програми та швидкого реагування на проблеми на етапі експлуатації. DevSecOps дозволяє скоротити цикл розробки ПЗ та застосувати CI/CD практики в командах «неперервної» розробки та доставки, що дозволяє їм швидко адаптуватися до вимог користувачів. Цей підхід дозволяє інтегрувати безпеку в усі операційні задачі, автоматизувати процеси розробки та розгортання, оптимізувати зусилля і витрати на розробку та зміну проекту.

Ключові історичні зміни у підходах до розробки, розгортання та безпеки програмних продуктів представлено на рисунку 2.

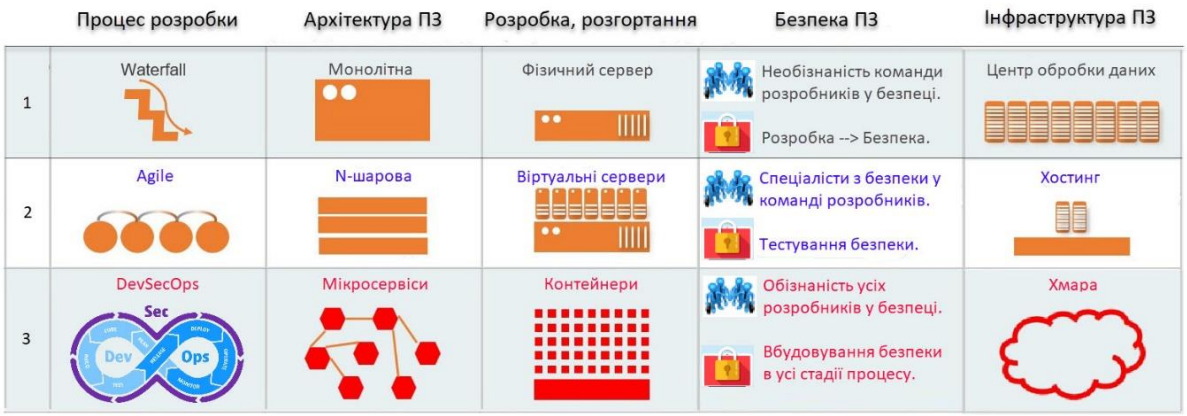

Рис. 2. Підходи до розробки та розгортання застосунків

Коротка характеристика трьох підходів до розробки програм:

1. Центри обробки даних із фізичними серверами, на яких працюють монолітні додатки, розроблені з використанням традиційних каскадних методологій. Погана обізнаність команд розробників у питаннях безпеки. Впровадження практик безпеки після завершення розробки. Розповсюджена в 1980-1990 роках.

2. Центри обробки даних і хостинг-провайдери, що працюють із серверами Unix/Linux, поява віртуалізації (VMWare, KVM), запуск багаторівневих додатків, розроблених за допомогою гнучкої методології. Залучення спеціаліста з безпеки до

команди розробників. Впровадження фази спеціального тестування безпеки. Розповсюджена в 2000-х роках.

3. Хмара все частіше замінює центри обробки даних і хостинг, декомпозиція додатків на мікросервіси, що працюють на контейнерній інфраструктурі, керовані та організовані Kubernetes, гнучка CI/CD розробки за допомогою методології DevSecOps. Навчання DevOps інженерів і розробників методам та засобам розробки безпечного ПЗ, розгортання ПЗ у захищеному середовищі. Адаптація та вбудовування практик безпеки в усі стадії процесу розробки та розгортання. Розвивається та використовується з 2010-го року.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Розвиток мікросервісної архітектури застосунків

Перехід від IT-інфраструктури on-premise до інфраструктури on-cloud призвів до модернізації архітектури застосунків і розвитку мікросервісів. Мікросервіси є кроком вперед у розвитку сервіс-орієнтованої архітектури (Service Oriented Architecture – SOA). Мікросервіси є архітектурним стилем створення застосунків, які будуються як сукупність окремих незалежних сервісів, кожен з яких відповідає за виконання конкретного функціонала, працює у власному процесі та взаємодіє з іншими за допомогою стандартних протоколів (рис. 3).

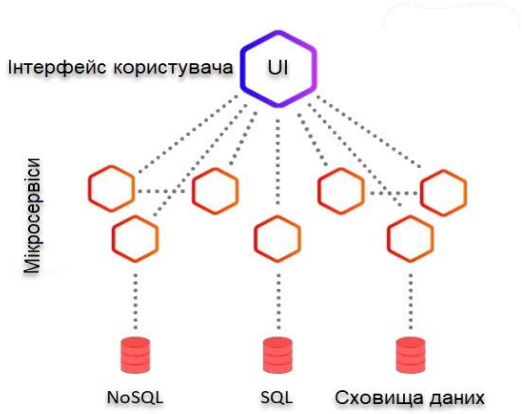

Рис. 3. Мікросервісна архітектура

Мікросервісна архітектура забезпечує гнучкість, швидкість, незалежність розробки та розгортання застосунку, легке масштабування та модифікацію коду за рахунок додавання нових сервісів або заміни сервісів [11], [12]. Мікросервіси можна розгортати незалежно, що дозволяє покращувати код застосунку, додавати нові функції та масштабувати кожну службу набагато легше, ніж у монолітній архітектурі. За допомогою мікросервісів можна оновити наявну службу без перебудови та повторного розгортання всієї програми. Кожна служба представляє окрему кодову базу, тому нею може керувати невелика команда розробників. Архітектура мікросервісів спрощує процес створення та підтримки складних програм, але не підходить для невеликих програм. Мікросервіси слабко пов’язані, тому, якщо одна служба виходить з ладу, решта продовжують працювати, що підвищує відмовостійкість усього застосунку. Крім того,

вони підтримують «поліглотне» програмування, тому сервісам не потрібно спільно використовувати той самий технологічний стек, бібліотеки чи фреймворки.

Переваги контейнерної інфраструктури

Для спрощення моделі розгортання та швидкого масштабування мікросервісів розробники все частіше їх розгортають у контейнерах. У такому вигляді мікросервіси стануть ключовим компонентом високо масштабованої та гнучкої інфраструктури, яку легко використовувати, розгортати та підтримувати [11]. Контейнери є однією із сучасних технологій, що швидко розвиваються та суттєво змінює середовище розробки застосунків. Можна розглядати контейнери як віртуалізацію операційної системи, в якій робочі процеси спільно використовують ресурси операційної системи, тобто контейнер інкапсулює «полегшене» середовище виконання програми. На рисунку 4 представлено спрощена структура застосунку на базі мікросервісів з контейнерізацією.

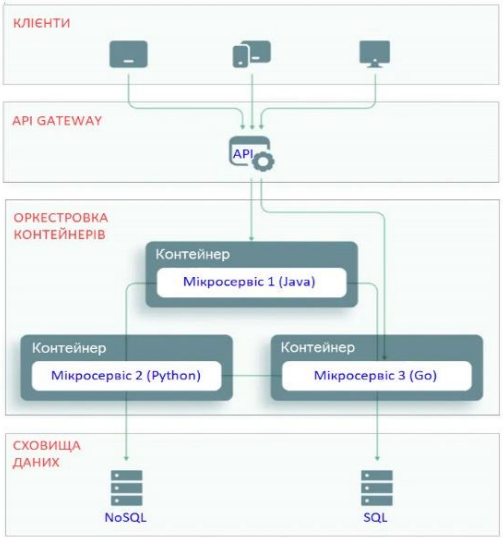

Рис. 4. Загальна структура застосунку на базі мікросервісів з контейнерізацією

Контейнери забезпечують усе, що потрібно для роботи певної служби, але відокремлюють програмне забезпечення від інших частин операційної системи чи інших служб, що працюють на тій же машині. Це дозволяє точно розподіляти ресурси обробки на рівні операційної системи, щоб можна було ефективно керувати процесами. Контейнери дозволяють упакувати програму разом із бібліотеками та іншими залежностями, забезпечуючи ізольоване середовище для запуску мікросервісу що зменшує ризик поширення вразливостей безпеки. Використання контейнерів дає змогу усунути проблеми конфігурації в різних середовищах, створювати легкі, автономні сервіси для розгортання, які містять усе, що необхідно застосунку: код, час виконання, системні інструменти, системні бібліотеки та налаштування. Контейнери забезпечують переносимість, масштабованість, додатковий рівень безпеки сервісу. Контейнеризація спрощує взаємодію мікросервісів, оскільки всі вони працюють у контейнерах, розташованих на одній платформі. З тієї ж причини розробникам легше оркеструвати мікросервіси. Docker, Kubernetes – це популярні платформа автоматизації управління контейнерами (оркестрації) програмного забезпечення, які призначають ресурси та управляють плануванням контейнерної інфраструктури [13].

Проблеми безпека контейнерних застосунків на базі мікросервісів

Як будь-який інший підхід, контейнери і мікросервіси можуть створити певні проблеми безпеки [14]. Щоб забезпечити безпеку програмного продукту, важливо знати про найпоширеніші загрози безпеці та планувати як запобігти або пом'якшити можливі негативні наслідки.

Головні вразливості контейнерних застосунків.

1. Вразливості на рівні оркестровки контейнерів. Застосунки на основі мікросервісів зазвичай є складними, оскільки вони складаються з багатьох частин. Один застосунок може містити сотні мікросервісів, які розгорнуті у тисячах контейнерів. Це робить додаток досить уразливим до кібератак, оскільки важко забезпечити належну безпеку при взаємодії такої кількості контейнерів. Загалом проблеми безпеки, пов'язані із мікросервісами та контейнерами, значною мірою вирішуються на рівні безпеки платформ оркестрування [15]. Інструменти оркестровки контейнера (Google Kubernetes, Docker Swarm, Apache Mesos) дозволяють користувачам управляти розгортанням контейнерів та автоматизувати оновлення, здійснювати моніторинг працездатності та процедури відновлення після відмови. Однак не з усіма ризиками безпеки можна впоратися на рівні оркестровки.

2. Вразливості образу контейнера (container image) – це загроза безпеці, вбудована в образ контейнера, яка є найпоширенішою загрозою безпеці в застосунках на базі мікросервісів і контейнерів [15]. Образ є проектом, інструкцією контейнера, за допомогою образу можна запускати контейнер. Контейнер являє собою завантажений образ в якому працює програма мікросервісу з описом залежностей відповідно до інструкції. Зазвичай такі вразливості виникають через незахищені бібліотеки або інші залежності. Хоча вразливі образи самі по собі не становлять активної загрози але якщо контейнер створено на основі вразливого образу, то контейнери можуть створити уразливість всьому контейнерному середовищу. Коли сервіс упаковано у контейнер, то «секрети» сервісу (ім'я користувача, пароль, закриті ключи тощо) можна вставити в образ. При цьому з'являється можливість проаналізувати образ і отримати доступ до «секретів», що створює ризик для безпеки. Створювати, налаштовувати і запускати контейнери можна за допомогою програми Docker.

3. Вразливості коду мікросервісів. Уразливості мікросервісу можуть виникати через недоліки у вихідному коді [16]. Наприклад, якщо один сервіс має вразливість переповнення буфера, можна використати її для виконання небезпечного коду та захоплення відповідного контейнеру.

API доступ до мікросервісів

Одним із найважливіших принципів безпеки для мікросервісу є гарантія того, що будь-який мікросервіс є чітко визначений, добре задокументований і стандартизований. Без створення API для забезпечення стандартизованого та чітко визначеного доступу до мікросервісів неможливо безпечне функціонування застосунків з мікросервісною архітектурою [17]. API має забезпечувати безпечну та добре визначену точку доступу до мікросервісу. Важливо, щоб кожен мікросервіс і його точка доступу API мали вбудовані механізми видимості та безпеки. Захищений API – це той, який може гарантувати конфіденційність інформації, яку він обробляє, роблячи її видимою лише для користувачів, сервісів і серверів, які мають право використовувати її. API має гарантувати цілісність інформації, яку він отримує від клієнтів і серверів, з якими він співпрацює, і обробляти таку інформацію, якщо вона не була змінена третьою стороною. API має бути завжди доступним для надійної обробки запитів, реалізації викликів до сторонніх серверів.

Ризики використання зловмисного програмного забезпечення.

Проблема зі зловмисним ПЗ полягає в тому, що якщо не виявити його перед запуском контейнера, то таке ПЗ може вразити мікросервіси цього контейнера та можливо усе середовище [15]. Наприклад, можна збирати конфіденційні дані, блокувати процеси або порушувати роботу інших контейнерів. Можна навести приклади типових атак:

- отримати доступ до контейнера та ввести у нього код, який може атакувати мікросервіс у цьому контейнері, інші контейнери або операційну систему хоста;
- компрометація середовища CI/CD і впровадження зловмисного програмного забезпечення у сховище вихідного коду, який використовуються для створення образів контейнерів;
- злам реєстру контейнерів і заміна образу на той, що містить шкідливе ПЗ;
- завантаження образів пошкоджених контейнерів із зовнішніх джерел.

Ризики, пов'язані з доступом до коду.

Чим більше людей можуть отримати доступ до коду та змінити його, тим більше виникає ризиків для безпеки [15]. Найпоширенішими є таки:

1. Занадто широкі права доступу. Багато компаній-розробників обирають підхід DevSecOps для створення застосунків за допомогою мікросервісів і контейнерів, оскільки він усуває бар’єри між командами та забезпечує постійну інтеграцію та безперервне розгортання (CI/CD). Однак DevSecOps може призвести до надання занадто широких прав доступу, збільшуючи ризик того, що хтось може змінити код у розподіленому контейнерному робочому середовищі.

2. Слабке управління секретами. Багато людей можуть отримати доступ до контейнерів у разі порушення правил безпеки. Наприклад, розробники можуть розміщувати закодовані облікові дані в сценаріях у контейнери або зберігати секрети у недостатньо захищеній системі управління ключами.

Необмежений зв'язок між контейнерами

Зазвичай контейнери не можуть отримати доступ до будь-яких ресурсів за межами середовища, яким вони безпосередньо керують — це називається непривілейованим режимом. Необхідно дозволяти лише ті зв’язку між контейнерами, які необхідні для правильної роботи програми [14]. Наприклад, контейнер сервісу обробки даних може встановити з’єднання з контейнером бази даних. Якщо контейнер має більше привілеїв, ніж потрібно, це може спричинити додаткові ризики безпеці. Контейнер може отримати надмірні привілеї в результаті неправильної конфігурації, спричиненої браком досвіду або неправильним управлінням оркестровкою.

Безпечне управління даними

Розподілена структура архітектури мікросервісів ускладнює захист даних, оскільки важко контролювати доступ і безпечну авторизацію до окремих служб [14]. Тому потрібно приділяти більше уваги забезпеченню конфіденційності, доступності і цілісності даних у кожній службі. Інша проблема полягає в тому, що дані в мікросервісах постійно переміщуються, змінюються та використовуються в різних службах для різних цілей, що створює більше точок входу до даних для зловмисників.

Вибір і налаштування інструментів розробки.

Під час розробки та підтримки архітектури мікросервісів команди DevSecOps використовують багато інструментів, у тому числі з відкритим вихідним кодом та сторонніх розробників, які не завжди забезпечують необхідну безпеку. Необхідно ретельно налаштовувати параметри інструментів для безпечного використання.

DevSecOps розробка та доставка контейнерних застосунків на базі мікросервісів

В умовах жорсткої конкуренції швидкість розробки має вирішальне значення для розробників ПЗ. Розробники мають працювати з потребами користувачів і оперативно усувати проблеми, які з'являються у ПЗ. Регулярні релізи та можливість швидко випускати версії після оперативного виправлення помилок стали новим стандартом для багатьох розробників, при цьому ключовою частиною їх процесів є CI/CD-конвейєрі.

DevSecOps – це методика інтеграції принципів безпеки в конвеєр безперервної інтеграції, безперервного постачання та безперервного розгортання (див. рис. 5). Розгортання та експлуатація стають продовженням процесу розробки із застосуванням практик Infrastructure as Code (IaC). Розробникам надається можливість автоматизувати налаштування, розгортання, управління IT-інфраструктурою з використанням загальних практик розробки ПЗ [18].

Рис. 5. Життєвий цикл ПЗ за методологією DevSecOps

Реалізація DevSecOps передбачає, що перевірка безпеки стає активною, невід'ємною частиною процесу розробки із застосуванням практик Security as Code (SaC). Безпека як код – це практика інтеграції безпеки в інструменти та процеси розробки для включення перевірок безпеки, тестів та шлюзів у процес внесення змін і код та інфраструктуру без додаткових витрат та затримок. Розробники можуть визначити інфраструктурні платформи та конфігурацію та створювати призначений для цього код. Базове розгортання SaC можна досягти шляхом включення в конвеєр CI/CD та код застосунку правил та політик безпеки, інструментів та агентів, тестів та сканерів безпеки. DevSecOps принцип зсуву ліворуч (Shift Left Security) означає, що проблеми безпеки, пошук вразливостей потрібно розглядати на початку життєвого циклу розробки, тестування та оцінку безпеки необхідно здійснювати на кожному етапі розробки і це є «обов'язком кожного». При кожній фіксації фрагменту коду, тести повинні виконуватися автоматично а результати мають бути доступні розробникам для виправлення. Це дозволяє одночасно тестувати та реалізувати можливості безпеки та функціональні можливості коду. При цьому з початку життєвого циклу розробки над вирішенням проблем безпеки застосунку тісно співпрацюють команди безпеки, розробки, тестування та розгортання. Зсув праворуч (Shift Right Security) вказує також на важливість приділяти увагу безпеці після розгортання програмного продукту. Деякі проблеми безпеки, вразливості можуть залишитися непоміченими під час попередніх сканувань безпеки та стати очевидними при використанні застосунку. Тому важливо, щоб команда експлуатації продовжувала відстежувати потенційні проблеми, вносити виправлення та працювати над випуском оновлених версій застосунку разом з командами із забезпечення безпеки та розробників.

Мікросервіси дозволяють створювати архітектуру, яка добре працює з автоматичним тестуванням і автоматичним розгортанням, оскільки кожен мікросервіс можна модифікувати окремо, що забезпечує оновлення та розгортання практично без простоїв. Мікросервіси дозволяють командам розробників працювати відносно незалежно та в різних місцях. Команди можуть бути організовані незалежно одна від іншої з використанням різних мов та середовищ розробки, завдання та обов'язки можуть бути розподілені між ними ефективніше. Контейнерізація мікросервісів – це справді революційний механізм доставки застосунків, який нарешті дозволяє видавцям контенту реалізувати мрію про економічну та ефективну віртуалізацію інфраструктури розгортання.

На рисунку 6 представлена DevSecOps модель розробки та розгортання мікросервісного застосунку з контейнеризацією, представлено головні засоби контролю безпеки на кожній стадії конвеєра [19].

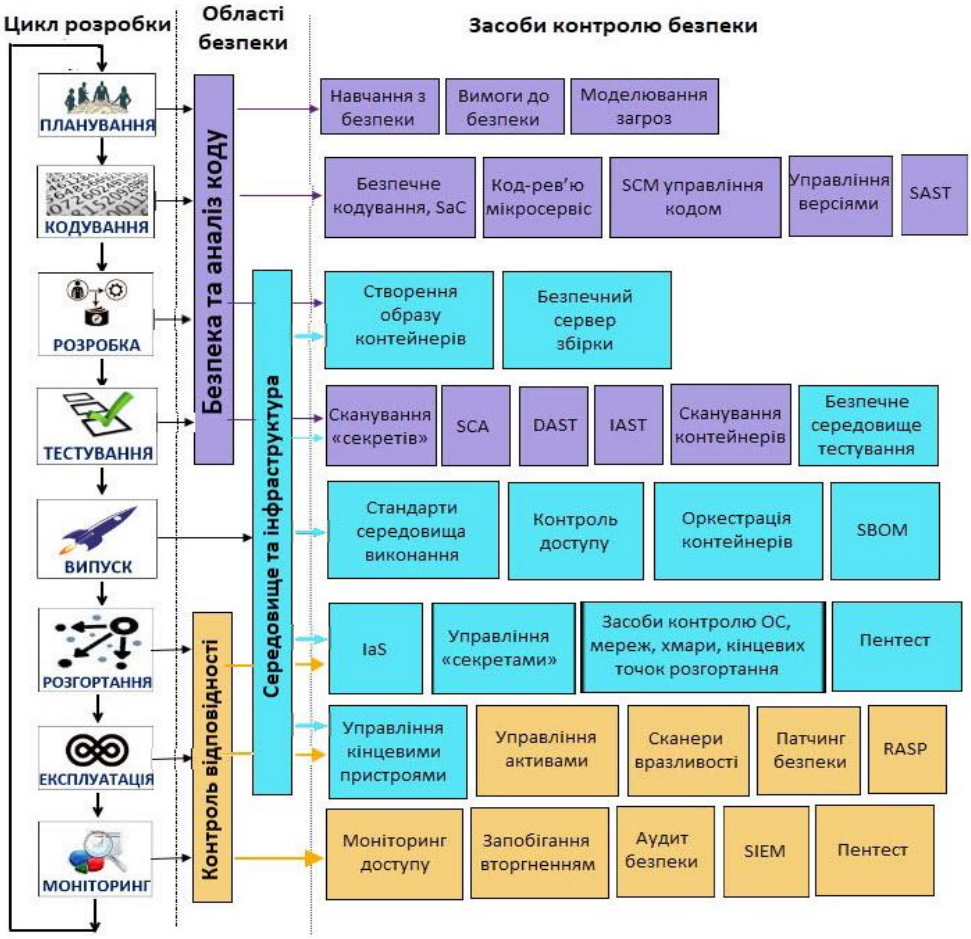

Рис. 6. Модель розробки та розгортання мікросервісного застосунку з контейнеризацією

Етап планування (Plan) є найменш автоматизованим і передбачає навчання практикам безпечної розробки, проведення аналізу вимог до безпеки розроблюваного застосунку, моделювання загроз безпеки [19]. Визначення вимог дозволяє створити «рецепт» розробки програми з нуля, який враховує ризики та вимоги до відповідності, конфіденційність даних, зв'язки з іншими системами, а також її технічні компоненти. Моделювання загроз дозволяє зрозуміти ландшафт загроз для розроблюваної програми.

Етап кодування (Code). Для створення безпечного коду використовуються принципи безпечного програмування та практики SaC [19]. Для підвищення безпеки сервісів необхідно проводити огляд коду (код – рев'ю) мікросервісів. Для кожного мікросервісу і його API необхідно забезпечити вбудовані механізми видимості та безпеки. В архітектурі застосунків, що базується на мікросервісах, лише API доступні зовні, що означає, що процеси, які надходять до API, можуть бути організовані та узгоджені незалежно. Огляд коду є важливим аспектом підходу до безпечного програмування і дозволяє мінімізувати кількість помилок, підвищити якість та безпеку коду. Огляд передбачає ретельну перевірку версій коду веб-сайту для усунення недоліків. Системи контролю версій запроваджують процеси рецензування, історію створення версій, яку можна перевірити, і шаблони розробки коду. Для управління вихідним кодом використовується системи централізованого управління кодом (SCM). Аналіз локальних конфіденційних даних дозволяє перевіряти код на наявність «секретів», облікових даних, ключів API тощо. Секрети, що зберігаються у вихідному коді, видимі іншим розробникам. Статичне тестування безпеки програм (SAST – Static Application Security Testing) дозволяє сканувати вихідний код на наявність вразливостей [20].

Етап розробки (Build). На цьому етапі відбувається збірка («складання») застосунку і доставка його у середовище тестування. Основним завданням є підготовка до автоматизованого аналізу безпеки вихідного варіанту збірки. У контейнерному застосунку необхідно створити безпечні образи контейнерів. Необхідно забезпечити перевірку стану кожного образу контейнера, запобігти запуску невідповідних образів, забезпечити дотримання практик безпечної конфігурації образів контейнера [11]. Образи контейнерів є пакетами, які містять всі необхідні файли для запуску контейнерів, вони мають переноситися в різні середовища без внесення змін. Образ, створений в лабораторії розробки, має легко переміщуватися до тестової лабораторії для оцінки, а потім у виробниче середовище для роботи.

Етап тестування (Test). Динамічне тестування безпеки додатків (DAST – Dynamic Application Security Testing), інтерактивне тестування безпеки додатків (IAST – Interactive Application Security Testing), аналіз складу ПЗ (SCA – Software Composition Analysis) є найважливішими процедурами безпеки [19]. DAST є методом тестування «чорної скриньки» для виявлення вразливостей у працюючому ПЗ без доступу до коду. IAST є методом тестування безпеки «сірої скриньки» для виявлення вразливостей ПЗ шляхом взаємодії з програмою в робочому середовищі. IAST складається із спеціальних моніторів безпеки, які запускаються із застосунку. SCA дозволяє сканувати вихідний код на наявність уразливих бібліотек і програмного забезпечення з відкритим кодом. SCA автоматизує процес виявлення потенційно небезпечного ПЗ з відкритим кодом. Сканування облікових даних та секретів дозволяє виявити небезпечні сховища даних. На цьому етапі необхідно підтвердити та перевірити вміст, функціональність образів контейнерів, підписати образи та надсилати образи до реєстру [11]. Реєстри зберігають образи та доставляють зображення оркестранту за запитом. Реєстри надають API, які дозволяють автоматизувати типові завдання, пов'язані із образом. Служби реєстру забезпечують централізований обмін образами і пошук сервісів для забезпечення повторного використання створеного ПЗ. Також необхідно сканувати образи контейнерів на наявність вразливостей образів, дефектів конфігурації образу. Управління вразливостями, включаючи виправлення та налаштування конфігурації образу є, як правило, обов'язком розробників під час створення нової версії образу.

Етап випуску (Release). На цьому етапі головною задачею є захист архітектури середовища виконання. Необхідно перевірити конфігурацію середовища, включаючи

контроль доступу користувачів з дотриманням принципу мінімальних привілеїв, перевірку ключів API для обмеження доступу, керування персональними даними [19]. Генерація актуального списку специфікацій всіх сторонніх компонент ПЗ, які використовуються у кодовій базі застосунку (SBOM – Software Bill of Materials) дозволяють виявити застарілі програмні бібліотеки з відомими вразливостями. Звіт 2021 з безпеки та аналізу ризиків ПЗ з відкритим вихідним кодом (OSSRA) виявив вразливості та конфлікти ліцензій у 1500 кодових базах, тому, крім аспекту безпеки, перелік специфікацій програмного забезпечення також може допомогти виявити невідповідність ліцензій з відкритим вихідним кодом. Необхідно використовувати найновіші версії компонентів застосунку, мов, фреймворків і операційних систем. На цьому етапі відбувається безпечне використання контейнерів та їх оркестровка. Оркестратори дозволяють отримати образи з реєстру, розгорнути їх у контейнерах та управляти запуском контейнерів на хостах. Хости запускають або зупиняють контейнери з виділеними ресурсами відповідно до інструкцій оркестратора [11]. Оркестратори мають засоби для безпечного створення, тестування та розгортання контейнерів у різних середовищах.

Етап розгортання (Deploy). На цьому етапі відбувається розгортання релізу застосунку в робочому середовищі. Необхідно використовувати практики IaC для створення прикладних середовищ та автоматизувати налаштування, розгортання, управління IT-інфраструктурою [19]. При контейнерному розгортанні сторонніх служб всі вони за своєю конструкцією здатні працювати практично будь-де – на звичайному апаратному забезпеченні, на гетерогенних кластерах, на однорідних кластерах, на високопродуктивних або на слабших екземплярах. Рекомендовано використовувати мережу доставки вмісту (CDN – Content Delivery Network), якщо це можливо, щоб підвищити доступність і захист програм та увімкнути політику безпеки вмісту (CSP – Content Security Policy) на веб-сервері або CDN. Рекомендовано шифрувати загальнодоступний трафік застосунку. Рекомендовано створити і використовувати дійсний сертифікат SSL для кожної URL-адреси програми або застосовувати wildcard-сертифікат відкритого ключа, налаштувати веб-службу для перенаправлення всіх вхідних запитів до безпечної кінцевої точки HTTPS. Запровадження брандмауера веб-програми (WAF – Web Application Firewall) дозволяє відстежувати, блокувати HTTP-трафік до служби та навпаки, захищати програму від відомих атак. Ведення журналів подій (Logging) у реальному часі дозволить в подальшому надсилати їх у централізоване сховище або SIEM для аналізу подій безпеки. Якщо програма розгорнута у хмарі або використовує власні хмарні служби, потрібно перевірити стан безпеки у хмарі для впровадження безпечних хмарних ресурсів, які відповідають найкращим практикам. Управління секретами передбачає створення безпечного зашифрованого сховища для зберігання конфіденційних даних, облікових даних, паролів, ключів API, токенів тощо. Для оцінювання захищеності та виявлення слабких місць стратегії захисту виконується тестування на проникнення (Penetration test – pentest).

Етап експлуатації (Operation). На цьому етапі проводиться технічне обслуговування, пошук функціональних недоліків, сканування вразливостей програмного продукту. Рекомендується використовувати інструменти IaC для ефективного захисту IT-інфраструктури, оновлювати ПЗ, яке використовується кінцевими пристроями. На цьому етапі потрібно регулярно отримувати зворотний зв'язок з безпеки застосунку, додавати необхідні поліпшення так само, як додавати будь-яку функціональність [19]. У процесі експлуатації в програмах постійно виявляються нові вразливості, тому потрібно швидко реагувати на виявлення «дірок» у застосунку та робити виправлення. Патч безпеки – це невелика програма, яка виправляє та «закриває»

будь-які вразливості, помилки чи інші проблеми застосунку, які можуть становити загрозу безпеці. Інструменти самозахисту програми під час виконання (RASP – Runtime Application Self Protection) також забезпечують додатковий захист та дозволяють автоматично виявляти та блокувати підозрілу поведінку. Засоби RASP працюють за принципом зворотного проксі-сервера і відстежують вхідні атаки, дозволяючи застосунку автоматично змінювати конфігурацію в залежності від умов.

Етап моніторингу (Monitor). Потрібно здійснювати постійний нагляд за функціонуванням застосунку для виявлення будь-яких порушень та відхилень від нормальної роботи. Важливим є впровадження засобів постійного моніторингу для визначення головних показників функціонування, виявлення вразливостей програмного продукту [19]. До таких засобів відносяться системи моніторингу доступу, аудиту безпеки, запобігання вторгненням, управління безпекою, тестування на проникнення. Система запобігання вторгненням (IPS – Intrusion Prevention System) дозволяє виявляти вторгнення або порушення безпеки та надає захист в автоматичному режимі. SIEM (Security information and event management) система управління безпекою та управління подіями безпеки забезпечує аналіз подій безпеки у реальному часі.

Інструменти автоматизації конвеєру DevSecOps

Розвиток технологій DevSecOps призвів до появи великої кількості інструментів для автоматизації конвеєрів CI/CD. Можна навести приклади інструментів, які дозволяють автоматизувати кожен етап конвеєру.

1. Етап планування. Slack – інструмент для співпраці та спілкування, Jira, Asana – управління проектом та відстеження проблем. IriusRisk – інструмент моделювання загроз.
2. Етап кодування. SonarQube, Codacy – інструменти аналізу якості коду. DeepSource, CodeScene – інструменти статичного аналізу. PMD, Gerrit, SpotBugs, CheckStyle, Phabricator і Find Security Bugs – інструменти безпеки коду.
3. Етап розробки. Checkmarx, SourceClear, Retire.js, SonarQube, OWASP Dependency-Check, Snyk – інструменти аналізу збірки. Anchore, Docker Scout – інструменти сканування образу контейнера.
4. Етап тестування. BDD Automated Security Tests, Boofuzz, JBro Fuzz, OWASP ZAP, SecApp suite, GAUNTLET, IBM AppScan, Arachi – інструменти тестування.
5. Етап випуску. HashiCorp Terraform, Docker, Ansible, Chef і Puppet – інструменти управління конфігурацією. Kubernetes, Mesos, Docker Swam – інструменти оркестрування контейнерів.
6. Етап розгортання. Osquery, Falco, Tripwire – інструменти верифікації застосунку. Chaos Monkey – пошук вразливостей інфраструктури. HashiCorp Vault, AWS Secrets Manager, Kubernetes Secrets – інструменти управління секретами.
7. Етап експлуатації. AWS CloudFormation, Prometheus, Nagios − інструменти управління інфраструктурою. Imperva RASP, Alert Logic, Halo, Raygun – інструмент моніторингу безпеки, захисту кінцевих точок (користувачів, баз даних).
8. Етап моніторингу. Vulnerability Manager Plus, Qualys Vulnerability Management, Tenable.io Vulnerability Management, Intruder.io – інструменти моніторингу вразливостей.

Разом з цим розвиваються платформи підтримки розробки та розгортання ПЗ. Такі платформи пропонують нові продукти контролю якості ПЗ (Quality Assurance, QA). Платформи підтримки конвеєрів розробки та розгортання пропонують засоби вистежування, тестування коду, забезпечення неперервної інтеграції версій, модулі комплексної безпеки коду, інструменти розгортання застосунків. У таблиці 1 наведені деякі популярні рішення розробки ПЗ [21].

Таблиця 1

Порівняння платформ підтримки конвеєру CI/CD

Платформи/ Властивості	Jenkins	CircleCi	TeamCity	Bamboo	GitLab	Azure DevOps
Відкритий код	Так	Так	Ні	Ні	Так	Ні
Інфраструктура	On-premise, on-cloud	On-cloud	On-premise, on-cloud	On-premise, Bitbucket cloud	On-premise, on-cloud	On-cloud
Безкоштовна версія	Так	Так	Так	30-днів	Так	30-днів
ОС, які підтримуються	Windows, Linux, macOS, Unix-подібні ОS	Windows, Linux, macOS, Solaris, FreeBSD	Window, Linux, macOS	Windows, Linux, macOS, Solaris	Windows, Linux, macOS,	Windows, Linux, macOS,
Інтеграція з іншими продуктами	AWS, Google Cloud, Azure, Digital, Jira, Pipeline, PostgreSQL, Slack, Trello, тощо	Anchore, Atlassian, AWS, Docker, GitHub, GitLab, Pulumi, Terraform тощо	Google Cloud, AWS, Docker, Visual Studio, GitLab, Bitbucket тощо	Jira, Bitbucket, AWS CodeDeplo, Opsgenie тощо	Asana, ClickUp, Discord, Google Sheets, Notion, Slack, Todoist, Trello тощо	AWS, Docker, GitHub, Google Cloud, Kubernetes, MS Azure, Slack, Testiny тощо
Ціна ліцензії	-	Від $15 на місяць.	Від $45 на місяць.	Від $10/агент/ місяць	Від $19/користувач/місяць	від $15 на місяць.

Традиційні інструменти сканування безпеки не дуже добре підходять для автоматизованого CI/CD-конвеєра але платформи підтримки DevSecOps містять інструменти, які спеціально розроблені для автоматизації та інтегруються в конвеєр, а результати відображаються на панелі моніторингу або безпосередньо передаються в системи відстеження помилок (багтрекер). Сканування безпеки необхідно проводити після кожної зміни коду. В таблиці 2 наведені сканери GitLab та DevSecOps on AWS.

Таблиця 2

Сканери безпеки платформ підтримки операцій DevSecOps

Назва	Опис	Платформа
SAST	Статичне тестування безпеки програми - сканує вихідний код програми та двійкові файли на наявність слабких і вразливих місць.	GitLab, DevSecOps on AWS
Dependency Scanning	Аналіз зовнішніх залежностей (наприклад, бібліотек) на наявність відомих вразливостей у коді за допомогою CI/CD. Перевірка відповідності ліцензії. Пошук схвалених і занесених у чорний список ліцензій, визначених спеціальними політиками проекту.	GitLab, DevSecOps on AWS
DAST	Динамічне тестування безпеки програми - аналізує працюючу веб-програму на наявність відомих вразливостей під час виконання.	GitLab, DevSecOps on AWS
(IAST)	Інтерактивне тестування безпеки додатків — аналізує працюючу програму в реальному середовище для пошуку відомих вразливостей. Може розпізнавати зашифровані дані, файлові системи та доступ до веб-сайтів.	DevSecOps on AWS

Container Scanning	Аналіз контейнерів на відомі вразливості безпеки в середовищі розгортання програми, використовуючи загальнодоступні бази даних вразливостей.	GitLab, DevSecOps on AWS
Secret Detection	Перевірка наявність облікових даних і секретів у версіях коду та проекту для завчасного виявлення неприпустимого розкриття конфіденційних даних.	GitLab, DevSecOps on AWS
Vulnerability Management	Перегляд, сортування, відстеження та усунення вразливостей, виявлених у програмах, оцінка ризиків.	GitLab
Fuzz Testing	Введення несподіваних та неправильно сформованих даних у програму, щоб виміряти наслідки та стабільність роботи програми з метою пошуку невідомих вразливостей.	GitLab

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У статті зроблено аналіз використання різних підходів до розробки програмних продуктів. Показано, що застосування організаційно-технічної методології DevSecOps дозволяє поєднувати робочі процеси управління проектами з автоматизованими IT-інструментами. За допомогою методології DevSecOps розробники застосунків впроваджують професійні методи та технології безпеки, які відповідають нормативним вимогам. Це дозволяє активно виявляти потенційні проблеми безпеки в коді, модулях та технологіях, які використовуються для створення контейнерних застосунків з мікросервісною архітектурою. Реалізація контейнерних застосунків на базі мікросервісів дозволяє децентралізувати та демократизувати процес розробки застосунків та доступ до даних. Доступ команді розробників до результатів сканування безпеки на ранніх стадіях написання коду дозволяє оптимізувати ресурси та заощадити час і кошти на розробку програмних продуктів.

Зазначимо що, швидке впровадження DevSecOps методології може бути ускладненим для розробників програм та вимагатиме навчання для впровадження «культури безпеки». Усі команди розробників конвеєру CI/CD мають бути однаково переконані у важливості впровадження методів та засобів забезпечення безпеки застосунків. Тому потрібно продовжувати роботи з вдосконалення методики створення безпечного програмного забезпечення з урахуванням особливостей архітектури застосунків та IT-інфраструктури їх розгортання із застосуванням сучасних інструментів автоматизації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Куліков, Є. (2022). Галузь розробки програмного забезпечення очима IT Ukraine. https://ko.com.ua/vitchiznyana_it-galuz_ochima_it_ukraine_140263
2 Conger, S. (2010). Software Development Life Cycles and Methodologies: Fixing the old and adopting the new. Sprouts: Working Papers on Information Systems, 10(172). https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0cc3ebc9b5490d4d6769f005dbc0d50d4a8ad8722
3 Martin, R. (2002). Agile Software Development, Principles, Patterns, and Practices. Pearson.
4 Wilson, G. (2020). DevSecOps: A leader's guide to producing secure software without compromising flow, feedback and continuous improvement. Rethink Press.
5 Mack, S. (2023). The DevSecOps Playbook: Deliver Continuous Security at Speed. Wiley.
6 Newman, S. (2021). Building Microservices, 2d Edition: Designing Fine-Grained Systems. O'Reilly Media. https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/
7 TeamCity (2020). Agile Continuous Integration. https://www.jetbrains.com/teamcity/ci-cd-guide/agile-continuous-integration/

8 TeamCity (2020). What are CI/CD Tools and how do they work. https://www.jetbrains.com/teamcity/ci-cd-guide/ci-cd-tools/
9 DORA (2022). Accelerate State of DevOps Report 2022. https://services.google.com/fh/files/misc/2022_state_of_devops_report.pdf
10 TeamCity (2020). What is DevSecOps and its role in CD. https://www.jetbrains.com/en-us/teamcity/ci-cd-guide/what-is-devsecops/
11 Scott, James A. (2017). A Practical Guide to Microservices and Containers. Mastering the Cloud, Data, and Digital Transformation. https://www.academia.edu/41522528/A_Practical_Guide_to_Microservices_and_Containers_Mastering_the_Cloud_Data_and_Digital_Transformation
12 Kocher, P. (2018). Microservices and Containers. Addison-Wesley Professional. 304. https://res.infoq.com/articles/microservices-and-containers-book-review/en/resources/Kocher_InfoQ_Sample-1525845801075.pdf
13 Ortega, J. (2022). Implementing DevSecOps with Docker and Kubernetes**.** An Experiential Guide to Operate in the DevOps Environment for Securing and Monitoring Container Applications (English Edition)
14 Souppaya, M., Morello, J., Scarfone, K. (2017). NIST Special Publication 800-190. Application Container Security Guide. https://doi.org/10.6028/NIST.SP.800-190
15 Chandramouli, R. (2019). NIST Special Publication 800-204. Security Strategies for Microservices-based Application Systems. https://doi.org/10.6028/NIST.SP.800-204
16 Production Reedy (2022). Мікросервісна архітектура у практиці DevOps. https://production-ready.dev/2022/11/mikroservisna-arkhitektura/
17 MuleSoft (2023). Microservices and Security: Increasing security by increasing surface area https://www.mulesoft.com/resources/api/microservices-security
18 Cloudfresh (2022). DevSecOps: Інтеграція безпеки продукту на кожному етапі SDLC. https://cloudfresh.com/ua/cloud-blog/devsecops-intehratsiya-produktu-bezpeky-na-kozhnomu-etapi-sdlc/.
19 McCarty P. (2022). DevSecOps Playbook - Version 1.3 https://github.com/6mile/DevSecOps-Playbook
20 Бурячок, В., Спасітєлєва, С., Складанний, П. (2018). Організація розробки безпечних .Net прикладних програм у закладах вищої освіти. *Сучасна спеціальна техніка*, 1(52), 13-23.
21 BrowserStack (2022). Top 14 CI CD Tools for your DevOps project https://www.browserstack.com/guide/top-ci-cd-tools

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 1 (21), 2023
ISSN 2663 - 4023

Svitlana O. Spasiteleva
PhD, Associate Professor, Associate Professor of the Department of Computer Science and Mathematics
Borys Grinchenko Kyiv University, Kyiv, Ukraine
ORCID ID: 0000-0003-4993-6355
s.spasitielieva@kubg.edu.ua

Ivan V. Chychkan
PhD, Associate Professor, Associate Professor of the Department of Information Systems and Technologies
Taras Shevchenko National University, Kyiv, Ukraine
ORCID ID: 0000-0002-0854-389X
Chychkan@knu.ua

Svitlana M. Shevchenko
PhD, Associate Professor, Associate Professor of the Department of Information and Cyber Security
Borys Grinchenko Kyiv University, Kyiv, Ukraine
ORCID ID: 0000-0002-9736-8623
s.shevchenko@kubg.edu.ua

Yulia D. Zhdanova
PhD, Associate Professor, Associate Professor of the Department of Computer Science and Mathematics
Borys Grinchenko Kyiv University, Kyiv, Ukraine
ORCID ID: 0000-0002-9277-4972
y.zhdanova@kubg.edu.ua

DEVELOPMENT OF SECURE CONTAINERIZED APPLICATIONS WITH A MICROSERVICES ARCHITECTURE

Abstract. The article analyzes approaches to software development that allow creating complex container applications with a microservice architecture based on automation tools and flexible development methods. The development of cloud technologies, the global strategy of containerization, the modernization of the application architecture, and the increase in security requirements have led to a change in the application development methodology. The study aims to determine approaches to increase the speed of development, security and quality of software code of containerized applications by implementing security principles and automation tools at all stages of the life cycle. Features and development prospects of microservice applications deployed in a container environment are considered. The advantages of the container infrastructure are defined: mobility, scalability, an additional level of microservice security. Containers provide an isolated environment for running a microservices, this reduces the risk of security vulnerabilities and simplifies interaction between microservices. The article identifies the security problems of microservice applications and the main vulnerabilities associated with the use of containers. It is determined that DevSecOps methodology allows implementing modern practice of continuous integration, continuous delivery, continuous application deployment and integration of security tools at all life cycle stages. DevSecOps describes development processes, deployment and operation processes using Security as Code and Infrastructure as Code practices. The research describes the model for developing and deploying microservice applications with containerization, defines the security domains, and the security controls for DevSecOps development pipeline. Based on this model, the main security control tools that must be used at all development and deployment pipeline stages are defined. The article proves that the considered technique regulates the implementation of given security procedures at all stages of the pipeline, allows to reduce development time and improve code quality for container applications with a microservices architecture.

Keywords: microservice architecture; container; DevSecOps; application security, security vulnerabilities.

REFERENCES

1 Kulikov, E. (2022). IT Ukraine about the of software development field. https://ko.com.ua/vitchiznyana_it-galuz_ochima_it_ukraine_140263

2 Conger, S. (2010). Software Development Life Cycles and Methodologies: Fixing the old and adopting the new. Sprouts: Working Papers on Information Systems, 10(172). https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0cc3ebc9b5490d4d6769f005dbc0d50d4a8ad8722
3 Martin, R. (2002). Agile Software Development, Principles, Patterns, and Practices. Pearson.
4 Wilson, G. (2020). DevSecOps: A leader's guide to producing secure software without compromising flow, feedback and continuous improvement. Rethink Press.
5 Mack, S. (2023). The DevSecOps Playbook: Deliver Continuous Security at Speed. Wiley.
6 Newman, S. (2021). Building Microservices, 2d Edition: Designing Fine-Grained Systems. O'Reilly Media. https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/
7 TeamCity (2020). Agile Continuous Integration. https://www.jetbrains.com/teamcity/ci-cd-guide/agile-continuous-integration/
8 TeamCity (2020). What are CI/CD Tools and how do they work. https://www.jetbrains.com/teamcity/ci-cd-guide/ci-cd-tools/
9 DORA (2022). Accelerate: State of DevOps Report 2022. https://services.google.com/fh/files/misc/2022_state_of_devops_report.pdf
10 TeamCity (2020). What is DevSecOps and its role in CD. https://www.jetbrains.com/en-us/teamcity/ci-cd-guide/what-is-devsecops/
11 Scott, James A. (2017). A Practical Guide to Microservices and Containers. Mastering the Cloud, Data, and Digital Transformation. https://www.academia.edu/41522528/A_Practical_Guide_to_Microservices_and_Containers_Mastering_the_Cloud_Data_and_Digital_Transformation
12 Kocher, P. (2018). Microservices and Containers. Addison-Wesley Professional. 304. https://res.infoq.com/articles/microservices-and-containers-book-review/en/resources/Kocher_InfoQ_Sample-1525845801075.pdf
13 Ortega, J. (2022). Implementing DevSecOps with Docker and Kubernetes**.** An Experiential Guide to Operate in the DevOps Environment for Securing and Monitoring Container Applications (English Edition)
14 Souppaya, M., Morello, J., Scarfone, K. (2017). NIST Special Publication 800-190. Application Container Security Guide. https://doi.org/10.6028/NIST.SP.800-190
15 Chandramouli, R. (2019). NIST Special Publication 800-204. Security Strategies for Microservices-based Application Systems. https://doi.org/10.6028/NIST.SP.800-204
16 Production Reedy (2022). Microservice architecture in DevOps practice. https://production-ready.dev/2022/11/mikroservisna-arkhitektura/
17 MuleSoft (2023). Microservices and Security: Increasing security by increasing surface area https://www.mulesoft.com/resources/api/microservices-security
18 Cloudfresh (2022). DevSecOps: The Integrate product security at each stage of the SDLC. https://cloudfresh.com/ua/cloud-blog/devsecops-intehratsiya-produktu-bezpeky-na-kozhnomu-etapi-sdlc/.
19 McCarty P. (2022). DevSecOps Playbook - Version 1.3 https://github.com/6mile/DevSecOps-Playbook
20 Buriachok, V., Spasiteleva, S., Skladannyi, P. (2018). Organization of development of safe .Net applications in higher education institutions. *Modern special technics*, 1(52), 13-23.
21 BrowserStack (2022). Top 14 CI CD Tools for your DevOps project https://www.browserstack.com/guide/top-ci-cd-tools