ISSN 2225-5036 (Print), ISSN 2411-071X (Online)
http://infosecurity.nau.edu.ua; http://jrnl.nau.edu.ua/index.php/Infosecurity

DOI: 10.18372/2225-5036.30.18579

ЗАХИСТ КРИТИЧНИХ РЕСУРСІВ В ХМАРНИХ СЕРЕДОВИЩАХ З ВИКОРИСТАННЯМ ПІДХОДУ БЕЗПЕКА ЯК КОД: ВИРІШЕННЯ ПРОБЛЕМИ НЕПРАВИЛЬНИХ КОНФІГУРАЦІЙ

Опірський Іван, Вахула Олександр

Національний університет «Львівська політехніка»

ОПІРСЬКИЙ Іван Романович, д.т.н., професор
Рік та місце народження: 1987 рік, м. Сімферополь, АР Крим, Україна.
Освіта: Національний університет «Львівська Політехніка», 2008 рік.
Посада: завідувач кафедри захисту інформації з 2023 року.
Наукові інтереси: методи і засоби технічного захисту інформації, охорона державної таємниці, проектування комплексних систем захисту інформації, лазерні системи акустичної розвідки, математичні методи та моделі захисту інформації, технічні канали витоку інформації, спецвимірювання.
Публікації: більше 220 наукових публікацій, серед яких наукові статті, монографії, навчальні посібники, тези та матеріали доповідей на конференціях.
E-mail: iopirsky@gmail.com.
Orcid ID: 0000-0002-8461-8996.

ВАХУЛА Олександр Петрович, асистент
Рік та місце народження: 1986 рік, м. Яворів, Львівська обл., Україна.
Освіта: Національний університет «Львівська Політехніка», 2008 рік.
Посада: асистент кафедри захисту інформації, з 2023 року.
Наукові інтереси: Захист інформації в хмарних середовищах, Захист ІТ інфраструктури.
E-mail: oleksandr.p.vakhula@lpnu.ua.
Orcid ID: 0009-0008-5367-3344.

***Анотація**. Хмарні технології набувають все більшої популярності серед організацій, як засіб для забезпечення масштабування, еластичності та ефективності ІТ-інфраструктури. Однак, широке впровадження хмарних рішень також відкриває нові виклики в контексті забезпечення інформаційної безпеки, зокрема, пов'язані з неправильними конфігураціями критичних ресурсів, що може призводити до непередбачених витоків даних, порушень доступності сервісів та інших інцидентів безпеки. У цій статті розглядається проблематика захисту критичних ресурсів у хмарних середовищах із особливим акцентом на виклики, пов'язані з неправильними конфігураціями. Автори пропонують ефективне вирішення цієї проблеми за допомогою підходів Security as Code (SaC), які дозволяють інтегрувати безпекові вимоги безпосередньо у процес розробки та розгортання хмарних ресурсів, тим самим реалізуючи превентивний контроль і забезпечуючи "Shift left" підхід в галузі кібербезпеки. У статті детально аналізуються типові випадки неправильних конфігурацій хмарних ресурсів та їх потенційний вплив на безпеку інформаційних систем. Далі, на основі сучасних досліджень та практичного досвіду, автори висвітлюють, як застосування SaC може допомогти автоматизувати процес виявлення та усунення таких вразливостей на ранніх етапах життєвого циклу розробки. Окрема увага приділяється інструментам і технологіям, які можуть бути використані для імплементації SaC. Стаття закликає до подальших досліджень у сфері використання SaC для забезпечення безпеки хмарних середовищ і пропонує напрямки для майбутніх розробок в цій області, включаючи автоматизацію виявлення конфігураційних помилок, розробку універсальних безпекових політик та створення стандартів для інтеграції безпеки в процес розробки програмного забезпечення. Для підтримки дослідження було проведена широкий аналіз літератури та статей, які надають інформацію про методології DevOps, DevSecOps, Shift-left, які служать базою для підходу Безпека як код.*

***Ключові слова:** Безпека як код, Інфраструктура як код, DevSecOps, DevOps, Хмарні середовища, Безпечний цикл розробки програмного забезпечення, CI/CD.*

Постановка проблеми

Хмарні обчислення стають домінантною платформою для бізнесу на всіх рівнях, безпека критичних ресурсів в хмарних середовищах набуває особливої важливості. Завдяки швидкому розвитку хмарних технологій та їх широкому розгортанню, орга-

нізації стикаються з викликами, пов'язаними з конфігурацією та управлінням безпекою своїх хмарних ресурсів. Неправильно налаштовані хмарні сервіси можуть призводити до серйозних ризиків, що в свою чергу відкриває шлях для потенційних порушень даних та інших кіберзагроз. Підхід "Security as Code" (Безпека як Код) можна вважати стратегічним рішення для ефективного та послідовного управління безпекою в хмарних середовищах.

"Security as Code" передбачає інтеграцію контролів безпеки безпосередньо в процес розробки та розгортання інфраструктури, шляхом використання коду для автоматизації та забезпечення безпеки ресурсів. Цей підхід дозволяє командам забезпечити, що політики безпеки розробляються, перевіряються, тестуються та застосовуються систематично та ефективно на всіх етапах життєвого циклу розробки програмного забезпечення.

В контексті даної статті буде розглянуто інструмент Open Policy Agent (OPA), який дозволяє описувати та впроваджувати політики безпеки як код. Опишемо процес написання цих політик, їх роль у автоматизації перевірки конфігурацій і як це може допомогти в усуненні неправильних конфігурацій в хмарних середовищах перед тим як вони будуть розгорнуті. Також буде продемонстровано, як підхід “Security as Code” може бути інтегрований у процес GitOps через популярний інструмент автоматизації GitHub Actions, який спрощує впровадження безпеки та управління політиками на етапі розгортання інфраструктури.

Аналіз останніх досліджень і публікацій

В останні роки концепція "Security as Code" (Безпека як Код), інтеграція DevSecOps, Secure Software Development Life Cycle (SSDLC) та підхід "Shift Left" набувають все більшої актуальності у світі кібербезпеки, що пов'язано зі стрімким розвитком технологій, зростаючою кількістю кібератак та потребою в швидкій розробці безпечного програмного забезпечення. Актуальність даної тематики обумовлена вимогою бізнесу то швидких результатів, комплексністю сучасних програмних систем та еволюцією кіберзагроз. Недавні дослідження вказують на ефективність автоматизації інструментів безпеки, таких як SAST(Static Application Security Testing), DAST (Dynamic Application Security Testing) у процесах CI/CD (Continuous Integration /Continuous Deployment), а також на значення підходів Policy as Code для автоматизації валідації конфігурацій. Практика "Shift Left", що передбачає залучення розробників до процесу забезпечення безпеки на ранніх етапах, та фокус на безпеці контейнерних та безсерверних архітектур є ключовими аспектами сучасних досліджень у цій галузі. Враховуючи швидкі зміни у вимогах та умовах ринку, впровадження цих практик дозволяє не тільки підвищити рівень безпеки продуктів, а й зробити процес їх розробки більш ефективним.

У прагненні прискорити вихід хмарних додатків на ринок часто ігноруються вимоги безпеки. Це, здебільшого, пов'язано з усталеним переконанням, що створення можливостей безпеки вимагає багато часу і впливає на графік доставки продукту. Однак, ігнорування необхідних засобів безпеки в додатках має каскадний вплив на бізнес-цілі [1].

Коли питання безпеки включаються до процедур DevOps, люди відіграють ще більш важливу роль, що включає співпрацю між цими командами та командою безпеки. Більш того, безпека особливо важлива при розробці критичних систем, де нам потрібно керувати цілями, ризиками та доказами. Після впровадження безпеки в інструментальний ланцюг DevOps, робота тільки починається. Нам також потрібно почати зі змін у поведінці, аби створити культуру безпеки [2].

Інтеграція діяльності з безпеки в конвеєр розробки DevOps вимагає високого рівня автоматизації, яку можна досягти лише частково за допомогою численних інструментів розробки, розгортання та тестування, що є на ринку. Слабке впровадження таких інструментів та відсутність моделей безпеки були визначені як основні невирішені проблеми, які перешкоджають визначенню всеосяжної методології SecDevOps [3].

Мета та постановка завдання

Запровадження підходу "Security as Code" є особливо важливим для критичної інфраструктури, де наслідки вразливостей можуть мати далекосяжні та часто катастрофічні наслідки. Кодифікація політик безпеки та автоматизація їх впровадження дозволяють гарантувати, що вимоги безпеки втілені від самого початку і на кожному кроці експлуатації систем, від енергетики до телекомунікацій та фінансових служб. Автоматизовані інструменти постійно перевіряють відповідність інфраструктури до політик безпеки, що допомагає швидко ідентифікувати та усувати будь-які невідповідності або вразливості. Це створює більш стійку оборону від зловмисних атак та інших кіберзагроз, що є критично важливим для інфраструктури, від якої залежать основні життєві функції суспільства та економіки. Такий підхід не просто сприяє підвищенню безпеки, але й покращує гнучкість та спроможність систем швидко адаптуватися до нових викликів, що є необхідним у динамічному світі хмарних технологій.

Виклад основного матеріалу дослідження

Невідповідна конфігурація ресурсів, як одна з важливих проблем безпеки в хмарах

Велика кількість атак на сучасні інформаційні системи, в тому числі ті, що локалізуються в хмарних середовищах відбувається через помилки у налаштуваннях, так звані “misconfigurations”, які можуть бути використані зловмисниками для отримання несанкціонованого доступу, крадіжки даних, реалізації атаки відмови у сервісі чи інших ворожих дій. Важливість відповідної та безпечної конфігурації не може бути недооціненою, адже навіть найменша недбалість може призвести до серйозних наслідків.

Якщо проаналізувати дослідження за 2023 рік, кількісне визначення відсотку атак, які відбуваються через неправильні налаштування (misconfigurations), може варіюватися залежно від джерела і методології дослідження. Однак, згідно з доповідями і аналізами безпеки, можна зазначити, що значна частка інцидентів безпеки пов'язана саме з проблемами конфігурації. За даними різних досліджень, можна приблизно вказати, що:

- порушення безпеки даних, пов'язані з хмарними сервісами, часто наслідок неправильних конфі-

гурацій доступу до сховищ даних, що можуть становити до 70% - 80% усіх випадків;

- інциденти безпеки, які включають витік інформації через неправильно налаштовані хмарні сервіси, можуть становити понад 50% від загальної кількості витоків.

Ці цифри слід розглядати як індикатори загальної тенденції, а не як точні статистичні дані. Проблеми з конфігурацією становлять великий ризик, оскільки хмарні середовища є дуже динамічними та комплексними, що робить важким управління конфігураціями без помилок [4].

Серед поширених невідповідних конфігурацій можна виділити наступні.

Управління доступом та обліковими записами:

- неправильно налаштовані політики управління ідентичністю та доступом можуть надати надмірні дозволи, що виходять за рамки принципу найменших привілеїв;
- невикористання багатофакторної аутентифікації для облікових записів хмарних сервісів підвищує ризик компрометації облікового запису.

Невідповідні мережеві налаштування:

- групи безпеки або мережеві ACL, які дозволяють неконтрольований доступ до/з інтернету, можуть відкрити хмарні ресурси для потенційних атак;
- відсутність сегрегації або належного визначення мережевих зон може дозволити переміщення в межах системи у разі порушення безпеки.

Використання налаштувань за замовчуванням:

- використання стандартних налаштувань безпеки без коригування для конкретних потреб організації може залишити системи вразливими;
- стандартні облікові дані та відкриті порти можуть бути легко використані нападниками.

Відсутність шифрування даних або неналежне його налаштування:

- відсутність шифрування при зберіганні або в процесі передачі для чутливих даних може відкрити їх для перехоплення або несанкціонованого доступу;
- невідповідні практики управління ключами можуть призвести до компрометації ключів шифрування;

Відсутність моніторингу та логування:

- неактивування або неналежна конфігурація протоколювання та моніторингу можуть перешкоджати виявленню несанкціонованого доступу або аномальних дій.

Невідповідні налаштування безпека баз даних:

- бази даних, що відкриті для інтернету без належної аутентифікації та шифрування.

Неправильні конфігурації сховищ:

- служби хмарного сховища, що залишилися відкритими для загального доступу, можуть призвести до ненавмисного розкриття даних;
- невпровадження належних політик життєвого циклу даних може залишити застарілі дані вразливими до загроз.

Неправильне управління секретами:

- жорстке кодування секретів (таких як паролі та токени) у вихідному коді або конфігураційних файлах, особливо коли вони зберігаються у системах контролю версій, може призвести до порушення конфіденційності даних.

Неправильні конфігурації у хмарних середовищах залишають відкритим широкий спектр потенційних вразливостей, які можуть бути легко використані зловмисниками. Це підтверджується численними інцидентами безпеки, що сталися через такі неправильні конфігурації. Особливу увагу потрібно приділити безпеці сховищ, управлінню секретами, резервному копіюванню та відновленню після аварій, а також управлінню оновленнями.

Постановка завдання та огляд підходу Безпека як код

Впровадження контролів безпеки в конвеєри безперервної інтеграції та розгортання (CI/CD) є критично важливим для забезпечення безпеки процесів розробки та впровадження програмного забезпечення. Нижче представлено таблицю контролів безпеки, які можуть бути інтегровані в CI/CD конвеєри, з вказівкою на те, які з них можуть бути повністю або частково автоматизовані за допомогою підходу Security as Code (SaC) (табл.1). У нашому випадку ми розглянемо Політику або відповідність як код, але далі в тексті будемо користуватися загальним терміном "Безпека як Код".

Таблиця 1

Список основних контролів з описом та можливістю автоматизувати підходом SaC

Контроль безпеки для CI/CD	Автоматизація	Опис
Статичне тестування безпеки коду (SAST)	Так (SaC)	Автоматичні інструменти сканують вихідний код на наявність вразливостей на ранніх етапах розробки.
Динамічне тестування безпеки коду (DAST)	Так (SaC)	Автоматичні інструменти сканують запущені програми для виявлення вразливостей виконання.
Аналіз композиції програмного забезпечення (SCA)	Так (SaC)	Автоматично ідентифікує та оцінює компоненти третіх сторін відкритого коду у кодовій базі на наявність вразливостей.
Сканування контейнерів	Так (SaC)	Автоматичне сканування образів Docker на наявність вразливостей та неправильних конфігурацій.
Виявлення секретів	Так (SaC)	Автоматично сканує код та середовища на наявність жорстко закодованих секретів, таких як паролі, API ключі тощо.
Перевірки якості коду	Частково	Інструменти оцінюють якість коду для виявлення потенційних проблем безпеки, але для складних питань часто необхідний ручний огляд.

Сканування відповідності стандартам	Так (SaC)	Автоматичні інструменти забезпечують відповідність коду та процесів розгортання актуальним стандартам і регуляціям.
Захист у реальному часі	Частково	Впровадження та налаштування захисту запущених програм (RASP) або мережевого захисного екрана веб-додатків (WAF) можуть бути автоматизовані, але налаштування часто вимагає ручного втручання.
Тести на проникнення	Частково	Хоча існують автоматизовані інструменти для пенетраційного тестування, комплексне тестування часто вимагає ручних методик для ідентифікації складних вразливостей.
Політики безпеки як код	Так (SaC)	Визначення та примусове впровадження політик безпеки через код для автоматичного забезпечення безпечних налаштувань та конфігурацій інфраструктури як код
Контроль доступу та удитування	Так (SaC)	Автоматизований контроль доступу та аудитування активів для забезпечення відповідності безпеки та виявлення аномалій.
Харденінг середовища	Так (SaC)	Скрипти для автоматичного харденінгу середовищ згідно з кращими практиками та вимогами відповідності.
Тренування з реагування на інциденти	Частково	Автоматизовані симуляції можуть проводитися, але людське втручання є вирішальним для аналізу та формування стратегії реагування.

Зрозуміло, що Інфраструктура як Код (IaC) є важливою та зростаючою частиною сучасного управління IT-інфраструктурою. Вона все частіше використовується для автоматизації конфігурації, налаштування та управління системами, особливо в складних масштабних середовищах. Зростання IaC відповідає потребам ефективніших, масштабованих та знижуючих помилки підходів в IT-інфраструктурі, що вказує на його зростаючу важливість у галузі. Цей тренд свідчить про значний та постійний перехід до практик IaC в IT-операціях та розробці [5].

Згідно документу національної агенції з безпеки (NSA Top Ten Cloud Security Mitigation Strategies), є розділ який звучить як "Впровадження безпечних автоматизованих практик розгортання через інфраструктуру як код" (Enforce Secure Automated Deployment Practices through Infrastructure as Code). Де чітко артикулюється на важливості використання "Інфраструктури як код" і пояснюється ризиком ручного розгортання хмарних ресурсів, що не тільки забирає багато часу, але є причиною людських помилок, які в свою чергу можуть призвести до неправильних конфігурацій та прогалин у безпеці. З інфраструктурою як кодом (IaC), ресурси визначаються в одному місці та включаються, як частина процесу безперервної інтеграції/безперервної доставки (CI/CD) [6-7].

Більшість споживачів хмарних послуг згодні з тим, що "Інфраструктура як код" надає можливість автоматизувати оперативне розгортання послуг в хмарі, виключаючи потребу в ручному налаштуванні та пов'язаних з цим помилках. "Безпека як Код" продовжує розвивати цю концепцію, впроваджуючи автоматизоване визначення політик безпеки, стандартів та передових методів, які слід інтегрувати як базові правила у конфігураційні скрипти, що використовуються для налаштування хмарних сервісів та систем. IT-відділ може відмовитися від необхідності постійно балансувати між гнучкістю бізнес-процесів і безпекою, усвідомивши, що ці компоненти можливо ефективно поєднати, забезпечуючи належний рівень обох без компромісів (рис. 1) [8-9].

Рис. 1. Опис процесних складових реалізації "Безпека як код" [10]

Організаційні політики містять перелік необхідних контролів безпеки, які розбиваються на правила, трансформовані в код, зрозумілий сервісу перевірки відповідності політикам. Ці правила потім групуються в ієрархічно організовані політики зі структурою. Цей централізований сервіс діє як контрольний бар'єр, перевіряючи код інфраструктури на відповідність встановленим політикам перед розгортанням ресурсів. Наприклад, якщо організація приймає політику, що вимагає шифрування приватних або фінансових даних у базах даних, ця вимога інтегрується в процедуру, як одне з правил, що автоматично активується, коли команда DevSecOps буде намагатися розгорнути код на створення інфраструктури. Якщо код не відповідає політиці, його буде відхилено. Інші приклади політик можуть включати вимоги до використання образів контейнерів або віртуальних машин лише з перевірених джерел, необхідність резервного копіювання даних, дублювання ресурсів у різних зонах доступності, шифрування дисків віртуальних машин, а також вимоги до тегування ресурсів. Такі політики можуть бути отримані на основі вимог стандартів, регуляцій, кращих практик та рекомендацій зовнішніх організацій, таких як Альянс Безпеки Хмарних технологій (CSA, Cloud Security Alliance), Центр Інтернет-Безпеки (CIS Benchmark), NIST-800, GDPR, HIPAA, PCI DSS, SOC2, а також внутрішніх політик

безпеки, які прийняті в корпорації. Більшість цих вимог можна описати як код [11].

Інфраструктура як Код (IaC) служить основою для модулів, що виконують статичний аналіз відповідності політиці. IaC може бути розгорнуто за допомогою інструментів, таких як CloudFormation для AWS, Deployment Manager для GCP та Resource Manager для Azure, при цьому Terraform, або Pulumi краще підходять для гібридних середовищ без прив'язки до конкретного провайдера. Статичні перевірки відповідності політиці можуть бути інтегровані в процес CI/CD, дотримуючись принципів GitOps.

Компонент центрального сервісу перевірки відповідності політикам може бути реалізований за допомогою Open Policy Agent (OPA) або Regula, обидва рішення – є відкритим програмним забезпечення. OPA, прийняте до Cloud Native Computing Foundation (CNCF) 29 березня 2018 року та досягло рівня зрілості "Graduated" 29 січня 2021 року.

Рішення працює незалежно від будь-якого конкретного сервісу чи платформи, дозволяючи послідовне застосування політики у різних середовищах та платформах. OPA забезпечує контроль за виконанням політик і інтегрується з популярними хмарними провайдерами та Kubernetes, роблячи його універсальним вибором для різних гібридних інфраструктур. OPA політики є централізованими та управляються консистентно.

Як проект з відкритим кодом, OPA має сильну спільноту та екосистему, яка забезпечує надійну підтримку та постійні вдосконалення. Ці особливості роблять OPA потужним інструментом для організацій, які прагнуть впровадити єдиний, гнучкий та масштабований підхід Безпеки як код. [12]

Чому використання OPA може бути кращим у деяких випадках?

Гнучкість та універсальність, OPA може бути інтегрований з широким спектром технологій і сервісів, не обмежуючись одним провайдером хмарних послуг.

Гранулярний контроль, OPA дозволяє створювати детальні політики, які можуть бути застосовані безпосередньо до конкретних ресурсів або операцій.

Незалежність від платформи, оскільки OPA є відкритим інструментом, його можна використовувати в гібридних середовищах, всіх популярних провайдерів та наземних інфраструктурах.

Також одним із ключових аспектів, на які вартує наголосити, це підвищення безпеки інформаційних систем за допомогою сегрегації обов'язків, яка передбачає розділення ролей та відповідальності між різними командами або індивідами в процесі розробки та управління IT-інфраструктурою.

Суть сегрегації обов'язків у рамках SaC полягає у чіткому розділенні відповідальностей між командами, що займаються розробкою політик безпеки (зазвичай це відділ інформаційної безпеки) та командами, які відповідають за розгортання та управління інфраструктурою (наприклад, команди розробників програмного забезпечення).

Однією з практичних реалізацій сегрегації обов'язків в контексті SaC є створення окремих репозиторіїв для політик безпеки та для конфігурацій інфраструктури. Це дозволяє ізолювати політики, розроблені командою інформаційної безпеки, від безпосереднього розгортання інфраструктури, якою займаються розробники або DevOps.

Такий підхід не тільки підвищує безпеку за рахунок мінімізації ризику несанкціонованого доступу або зміни критичних безпекових налаштувань, але й сприяє більшій гнучкості та швидкості реалізації змін в інфраструктурі, зберігаючи при цьому високий рівень контролю та відповідності безпековим стандартам.

Реалізація сегрегації обов'язків через SaC вимагає чіткої координації між командами та розробки спільних процесів взаємодії, що включають регулярні перевірки та аудити політик , а також їх актуалізацію та оптимізацію відповідно до змінюваних умов та загроз. Такий інтегрований підхід дозволяє створити ефективну та безпечну інфраструктуру, яка відповідає вимогам та кращим практикам у сфері інформаційної безпеки.

Методологія розробки та перевірки політик

В нашому прикладі політики будуть написані на основі вимог Стандарту CIS Benchmark для Amazon Web Services версії 2.0.0. Усі стандарти CIS зосереджені на технічних налаштуваннях конфігурації для підтримки або покращення безпеки і повинні використовуватись разом з іншими важливими контролями. Документ "CIS Amazon Web Services Foundations Benchmark" є набором кращих практик конфігурації безпеки для Amazon Web Services (AWS).

Він зазвичай охоплює такі області, як управління ідентифікацією та доступом (IAM), логування та моніторингу, конфігурації мережевих ресурсів для захисту інформації та ресурсів, заходи безпеки для сервісів на кшталт EC2, S3, RDS тощо, а також загальні керівні принципи безпеки та відповідності.

Цей стандарт призначений для використання, як кращі практики для AWS з ціллю покращення безпеки середовищ цього хмарного провайдера.

Для дослідження ми виберемо вимоги щодо шифрування EBS диску та налаштування мережевої групи та опишемо їх мовою Rego для подальшого використання в OPA.[13] Також ми відразу перевіримо чи працює дана політика за допомогою простого інструменту - Open Policy Agent (OPA) Playground, що є інтерактивним середовищем, яке дозволяє писати і тестувати політики за допомогою мови запитів Rego (https://play.openpolicyagent.org/).

Вимога CIS Amazon Web Services Foundations Benchmark v2.0.0 - 06-28-2023 - 5.2. Ensure no security groups allow ingress from 0.0.0.0/0 to remote server administration ports.

В поле з лівого боку вставляємо нашу створену політику щодо вимоги вище (рис. 2). Для перевірки внесемо в поле Input приклад коду Terraform (в JSON форматі) який декларує створення AWS Security Group, із завідомо небезпечною конфігурацією де група безпеки AWS дозволяє вхідний трафік з 0.0.0.0/0 (всі IP-адреси) на порт 22 (SSH), що порушує вимогу CIS AWS Benchmarks та спостерігаємо результати в полі Output.

У цьому Rego скрипті, результат (output) `allow_ssh` стає `false` (що вказує на помилку політики) тому що знайдено правило, що дозволяє SSH з будь-якої IP-

адреси, відповідно `violation` створює повідомлення з описом невідповідності.

Посилання на репозиторій з кодом: https://gist.github.com/Alevak/5d608deaae16f99dd767485556d0ff19.

Рис. 2. Перевірка кодифікованої політики в пісочниці

Вимога CIS Amazon Web Services Foundations Benchmark v2.0.0 - 06-28-2023 - 2.2.1. Ensure EBS Volume Encryption is Enabled in all Regions)

В поле з лівого боку вставляємо нашу створену політику, для перевірки в поле Input описуємо приклад коду Terraform (в JSON форматі), який декларує створення AWS EBS, із шифруванням, що є конфігурацією згідно кращих практик вказаних у вимозі вище. Вже у цьому Rego скрипті, результат (output) `allow` є `true', результат перевірки позитивний і довзолить подальше виконання скрипта (рис. 3) Посилання на репозиторій з кодом [14]: https://gist.github.com / Alevak / 0db683fe7b82f10c3d0a363295d6d234.

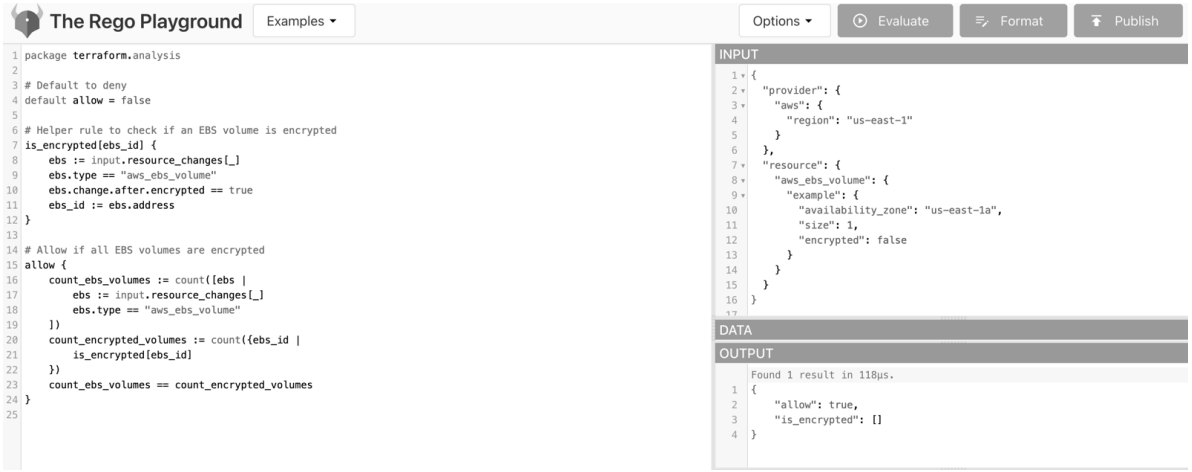

Рис. 3. Перевірка кодифікованої політики в пісочниці

Практичне впровадження безпосередньо в цикл розробки, зокрема в систему безперервної інтеграції та безперервного розгортання (CI/CD).

Повний потенціал Terraform та OPA реалізується, коли вони інтегровані в CI/CD конвеєр безпосередньо. У нашому прикладі, ми використовуємо GitHub Actions для запуску цих перевірок при кожному Pull request. Pull request у контексті систем управління версіями, таких як Git, - це запит на внесення змін до коду в головному репозиторії проекту. Коли розробник хоче додати свої зміни до основної гілки проекту, він створює pull request, щоб інші члени команди могли переглянути зміни, залишити коментарі та вирішити, чи можна ці зміни об'єднати з основною гілкою. GitHub Actions — це функція на платформі GitHub, яка дозволяє автоматизувати процеси розробки програмного забезпечення безпосередньо у GitHub репозиторії. Ця реалізація забезпечує перевірку будь-яких пропонованих змін інфраструктури на відповідність створеним нами раніше політикам перед їх злиттям і розгортанням у продуктивному середовищі.

Нижче ми надаємо приклад YAML коду, який створить процес перевірки відповідності, у нашому випадку, стосовно безпечної конфігурації середовища AWS відповідно до CIS Benchmark:

```
# This workflow validates Terraform code in a GitHub repository.
name: Validate Terraform
# Trigger the workflow on pull requests targeting the main branch.
on:
  pull_request:
    branches:
      - main
# Define jobs to run.
jobs:
```

```
    terraform:
      # Run this job in an Ubuntu environment.
      runs-on: ubuntu-latest
      # Define the sequence of tasks (steps) that will be executed as part of this job.
      steps:
      # Check out the repository code at the pull request commit.
      - name: Checkout repository
        uses: actions/checkout@v3
        with:
          fetch-depth: 0  # Fetch all history for all tags and branches
      # Set up Terraform CLI in the GitHub Actions runner.
      - name: Set up Terraform
        uses: hashicorp/setup-terraform@v2

      # Configure AWS credentials using secrets stored in the GitHub repository.
      - name: Configure AWS Credentials
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ secrets.AWS_REGION }}

      # Initialize Terraform in the GitHub Actions runner.
      - name: Terraform Init
        run: terraform init
      # Validate the Terraform files to ensure they are syntactically valid and internally consistent.
      - name: Terraform Validate
        run: terraform validate
      # Generate a Terraform plan and convert it to JSON format, excluding debug lines.
      - name: Generate Terraform Plan JSON Output
        run: |
          terraform plan -var="db_username=${{ secrets.DB_USERNAME }}" -var="db_password=${{ secrets.DB_PASSWORD }}" -out=tfplan.binary
          terraform show -json tfplan.binary | grep -v "::debug::" | tail -n +2 > plan.json

      # Install Open Policy Agent (OPA) to evaluate custom policy rules against the Terraform plan.
      - name: Install OPA
        run: |
          curl -L -o opa https://openpolicyagent.org/downloads/latest/opa_linux_amd64
          chmod 755 ./opa
          sudo mv opa /usr/local/bin/

      # Evaluate the Terraform plan against Rego policies defined in the repository and output results to a file.
      - name: Evaluate Rego Policies
        id: evaluate_policies
        run: |
          opa eval --data ./policies/aws_tags_policy.rego --input plan.json --format pretty "data.main.deny" > tags_policy_result.txt
          if [[ -s tags_policy_result.txt ]]; then
            exit 1  # Fail the step if there are policy violations
          fi
      # If policy violations are found, add the results to the job summary so they can be reviewed.
      - name: Add policy results to job summary
        if: failure()
        run: |
          if [[ -s tags_policy_result.txt ]]; then
            echo "Tags policy violations:" >> $GITHUB_STEP_SUMMARY
            cat tags_policy_result.txt >> $GITHUB_STEP_SUMMARY
          fi [15]
```

Давайте розглянемо кожен крок робочого процесу GitHub Actions:

1. Робочий процес запускається при створенні pull request до головної гілки (`main`);

2. Виконання роботи на останньому Ubuntu Runner: робота з назвою `terraform` виконується на останньому віртуальному середовищі Ubuntu, наданому GitHub Actions;

3. Витягування репозиторію: код репозиторію витягується для використання у робочому процесі;

4. Налаштування Terraform: встановлюється і налаштовується вказана версія Terraform;

5. Налаштування облікових даних AWS: налаштовуються облікові дані AWS за допомогою секретів, збережених у репозиторії GitHub, для безпечного доступу до сервісів AWS;

6. Terraform Init: ініціалізує Terraform, налаштовуючи робочий каталог для операцій Terraform;

7. Terraform Validate: перевіряє файли Terraform на синтаксичну коректність;

8. Генерація плану Terraform: створює план виконання Terraform і виводить його у бінарному форматі, потім конвертує його до JSON;

9. Оцінка політик Rego: оцінює користувацькі політики, написані на Rego, на основі плану Terraform і виводить результати у текстові файли;

10. Додавання результатів політик до підсумку роботи: якщо є порушення політик, вони додаються до підсумку роботи, який видно у користувацькому інтерфейсі GitHub Actions.

Кожен крок забезпечує, що код, злитий до головної гілки (`main`), відповідає визначеним політикам і безпечний для розгортання (рис.4).

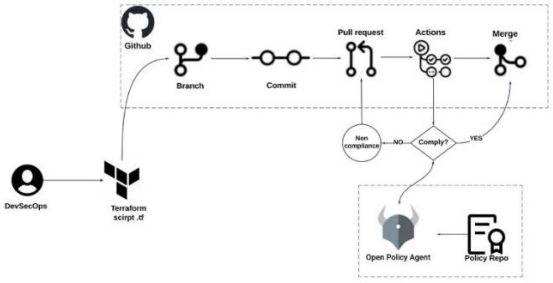

Рис. 4.Схема реалізації перевірок скриптів Інфраструктури як код за допомогою Github Actions та OPA

Може бути ситуація, коли зміни вже в існуючу інфраструктури було внесено вручну через консоль управління хмарою, в такому випадку ми маємо мати можливість виявити відхилення від наших вимог. Для цього так само можна використати Terraform або подібний інструмент в рамках workflow GitHub Actions, щоб створити актуальний знімок поточного стану інфраструктури у форматі JSON.

Файли IaC у репозиторії представляють бажаний стан інфраструктури. Далі OPA порівнює актуальний знімок із бажаним станом, визначеним у файлах IaC та політиках OPA, щоб виявити будь-які розбіжності або відхилення. Якщо виявляється відхилення, можна використати GitHub Actions для повідомлення про ці розбіжності, записуючи результати оцінки OPA у журналі роботи або створюючи проблему у репозиторії GitHub. Для відхилень, які можна буде безпечно автоматично виправити, можна додати кроки у workflow GitHub Actions для застосування необхідних змін до інфраструктури, щоб вирівняти її з бажаним станом, визначеним у файлах IaC. Зазвичай це включає виконання команди `terraform apply` або подібної.

Для складніших сценаріїв відхилень або в тих випадках, де автоматичне виправлення може бути небезпечним, потрібно додати кроки для повідомлення відповідального персоналу для ручного перегляду та вирішення розбіжностей. Повідомлення можна реалізувати через фукціонал проблеми GitHub, коментарями до запитів на злиття або інтегрувавшись з зовнішніми системами, такими як Slack або електронна пошта.

Інтеграція Terraform і Open Policy Agent (OPA) у конфеєр безперервної інтеграції/безперервного розгортання (CI/CD) через GitHub Actions є значним просуванням у практиці інфраструктури як код (IaC). Цей підхід не тільки автоматизовує процес розгортання інфраструктури, але й забезпечує високий стандарт відповідності та безпеки.

Проводячи ретельні оцінки політик на кожному pull request, система забезпечує, щоб до головної гілки (`main`) злито тільки код, який відповідає встановленим стандартам безпеки та операційної діяльності. Ця методологія ефективно мінімізує людську помилку та оптимізує процес розгортання, забезпечуючи послідовність, безпеку та відповідність всіх змін інфраструктури стандартам Центру Інтернет-Безпеки (CIS Benchmark) для AWS середовищ. Детальний розгляд кожного кроку в робочому процесі GitHub Actions підкреслює ретельність процесу, пропонуючи чіткий та ефективний шлях для підтримки цілісності інфраструктури. Ця практика не тільки підвищує безпеку, але й сприяє більш надійному циклу розробки.

Висновки. Завершуючи дану статтю про "Security as Code"(або "Policy as Code"), слід визнати, що ці підходи можуть бути визначальними для гарантування безпеки критичної інфраструктури в хмарних середовищах, або в будь яких середовищах які мають жорсткі вимоги до безпеки. Спроможність інтегрувати захисні механізми безпосередньо у код, що керує інфраструктурою, трансформує практику безпеки, перетворюючи її на невід'ємний компонент розробки програмного забезпечення. Зокрема, використання цих методів для розгортання кластерів Kubernetes дозволяє впроваджувати надійні та ефективні стратегії безпеки, що мінімізують ризики в широкомасштабних системах. Враховуючи це, подальші дослідження повинні зосередитись на розвитку інструментів, які спрощують інтеграцію безпеки, покращення методів тестування та автоматизації відповідності регулятивним вимогам для високодоступних та масштабованих системи, як Kubernetes кластери. Це допоможе розвивати підходи, які підтримують безперервну інноваційність у хмарному обчисленні, одночасно підтримуючи високий рівень безпеки, який важливий для підтримки суспільних та економічних функцій в сучасному світі.

Список літератури

[1]. Rakesh Kumar, Rinkaj Goyal (2021). When Security Meets Velocity: Modeling Continuous Security for Cloud Applications using DevSecOps (https: // link. springer.com/chapter/10.1007/978-981-15-9651-3_36).

[2]. Mary Sánchez-Gordón, Ricardo Colomo-Palacios Security as Culture: A Systematic Literature Review of DevSecOps (https: //dl.acm.org/doi/10.1145/ 33879-40.3392233).

[3]. Valentina Casola, Alessandra De Benedictis, Carlo Mazzocca, Vittorio Orbinato, Secure software development and testing: A model-based methodology, Computers & Security, Volume 137, 2024, 103639, ISSN 0167-4048, https: // doi.org / 10.1016/j.cose.2023.103639. (https: // www.sciencedirect.com / science /article/pii/ S0167404823005497).

[4]. Pal Alto Unit42 Cloud Threat Report volume7 (https: // www.paloaltonetworks.com / content / dam/ pan/en_US/assets/pdf/reports/unit42-cloud-threat-report-volume7.pdf).

[5]. K. Carter, "Francois Raynaud on DevSecOps," in IEEE Software, vol. 34, no. 5, pp. 93-96, 2017, doi: 10.1109/MS.2017.3571578 (https://ieeexplore.ieee.org/document/8048652).

[6]. FORT MEADE, Md. The National Security Agency (NSA) is releasing "Top Ten Cloud Security Mitigation Strategies", Cybersecurity Information Sheet, Enforce Secure Automated Deployment Practices through Infrastructure as Code (2024) (https://media.defense.gov/2024/Mar/07/2003407857/-1/-1/0/CSI-CloudTop10-Infrastructure-as-Code.PDF).

[7]. V. Gazdag. A Guide to Improving Security Through Infrastructure-as-Code. 2022. https://research.nccgroup.com/2022/09/19/a-guide-to-improving-security-through-infrastructureas-code/.

[8]. Chhavi Adtani, Aaron Bawcom, Jan Shelly Brown, Rich Cracknell, Rich Isenberg, Kaz Kazmier, Pablo Prieto-Munoz, and David Weinstein (2022). Security as code: The best (and maybe only) path to securing cloud applications and systems (https: // www.mckinsey.com/capabilities/mckinsey-digital/our-insights/security-as-code-the-best-and-maybe-only-path-to-securing-cloud-applications-and-systems).

[9]. Sarthak Das (2023). Security as Code 1st Edition.

[10]. Vakhula O., Opirskyy I., Mykhaylova O. Research on Security Challenges in Cloud Environments and Solutions based on the "security-as-Code" Approach (2023). CEUR Workshop Proceedings, 3550, pp. 55-69.

[11]. Xuejiao Zhang (2021). Cloud governance and compliance on AWS with policy as code (https://aws.amazon.com/ru/blogs/opensource/cloud-governance-and-compliance-on-aws-with-policy-as-code/).

[12]. Becki Lee (2022). Using Open Policy Agent (OPA) to Apply Policy-as-Code to Infrastructure-as-Code (https: // cloudsecurityalliance.org / blog /2020/04/02/ using-open-policy-agent-opa-to-apply-policy-as-code-to-infrastructure-as-code/).

[13]. CIS Amazon Web Services Foundations Benchmark v2.0.0 - 06-28-2023.

[14]. Policy language documentation https://www.openpolicyagent.org/docs/latest/policy-language/.

[15]. Guest Expert on GitGuardian (2022) What is Policy-as-Code? An Introduction to Open Policy Agent (https://blog.gitguardian.com/what-is-policy-as-code-an-introduction-to-open-policy-agent/).

УДК 004:378.091.31

Opirskyy I., Vakhula O. Protecting critical resources in cloud environments through Security as Code approaches: solving the problem of misconfigurations

Abstract. *Cloud technologies are becoming increasingly popular among organizations as a means of ensuring scalability, flexibility, and efficiency of IT infrastructure. However, the widespread adoption of cloud solutions also opens up new challenges in the context of information security, particularly those related to improper configurations of critical resources that can lead to unexpected data leaks, service availability breaches, and other security incidents. This article addresses the issue of protecting critical resources in cloud environments, with a special focus on challenges related to improper configurations. The authors offer an effective solution to this problem by using Security as Code (SaC) approaches, which allow security requirements to be integrated directly into the development and deployment processes of cloud resources, thus implementing preventative control and ensuring a "Shift left" approach in cybersecurity. The article thoroughly analyzes typical cases of improper configurations of cloud resources and their potential impact on the security of information systems. Furthermore, based on current research and practical experience, the authors illuminate how the application of SaC can help automate the process of detecting and remedying such vulnerabilities at early stages of the development lifecycle. Particular attention is given to the tools and technologies that can be used for implementing SaC. The article calls for further research in the use of SaC to ensure the security of cloud environments and proposes directions for future developments in this area, including the automation of configuration error detection, the development of universal security policies, and the creation of standards for integrating security into the software development process. To support the research, a broad analysis of literature and articles providing information about methodologies like DevOps, DevSecOps, Shift-left, which serve as the foundation for the Security as Code approach, was conducted.*

Keywords: *Security as code, Infrastructure as code, DevSecOps, DevOps, Cloud environments, secure software development cycle, CI/CD.*

Опірський Іван Романович, доктор технічних наук, професор, кафедра захисту інформації, Національного університету «Львівська політехніка».

Ivan Opirskyy, doctor of Technical Sciences, professor, Department of Information Security, Lviv Polytechnic National University.

Вахула Олександр Петрович, асистент, кафедра захисту інформації, Національного університету «Львівська політехніка».

Oleksandr Vakhula, assistant, Department of Information Security, Lviv Polytechnic National University.

Отримано *29 січня 2024 року*, затверджено редколегією *1 квітня 2024 року*