УДК 004.72.056.52:004.4 DOI: 10.31673/2412-4338.2025.038705

Марценюк Євгеній Віталійович
Національний Університет "Львівська Політехніка", Львів, Україна
ORCID: 0009-0009-2289-0968

Чорній Владислав Володимирович
Національний Університет "Львівська Політехніка", Львів, Україна
ORCID: 0009-0006-8671-3889

Партика Андрій Ігорович
Національний Університет "Львівська Політехніка", Львів, Україна
ORCID: 0000-0003-3037-8373

Гарасимчук Олег Ігорович
Національний Університет "Львівська Політехніка", Львів, Україна
ORCID: 0000-0002-8742-8872

АВТОМАТИЗОВАНЕ УПРАВЛІННЯ РИЗИКАМИ ВИТОКУ СЕКРЕТНОЇ ІНФОРМАЦІЇ У ПРОГРАМНОМУ КОДІ

Анотація: *У статті здійснено комплексне дослідження проблеми інформаційної безпеки, що виникає внаслідок неконтрольованого зберігання секретної інформації у програмному коді сучасних програмних систем. Детально проаналізовано категорії конфіденційних даних — API-ключі, токени доступу, облікові дані до баз даних, приватні криптографічні ключі та конфігураційні параметри, — витік яких створює критичні ризики для цілісності, конфіденційності та доступності IT-інфраструктури. Особливий акцент зроблено на інфраструктурних загрозах у багатохмарних середовищах та автоматизованих CI/CD-конвеєрах, де випадкове розміщення облікових даних у вихідному коді призводить до компрометації ланцюга постачання, несанкціонованого доступу до хмарних сервісів і порушення безперервності бізнес-процесів. У межах емпіричного дослідження проведено аналіз ста відкритих репозиторіїв GitHub із використанням сучасних засобів автоматизованого виявлення секретів (TruffleHog, GitLeaks, detect-secrets). Отримані результати класифіковано за видами виявлених секретів, рівнями критичності та потенційними наслідками їх витоку, що дозволило ідентифікувати найбільш поширені вектори ризику та визначити пріоритети їх усунення. Формалізовану оцінку ризиків здійснено відповідно до методології NIST SP 800-30, що передбачає ідентифікацію активів, загроз, вразливостей, оцінювання ймовірності реалізації загроз та масштабів їх впливу. На основі проведеного аналізу обґрунтовано необхідність інтеграції процесів безпечного управління секретами у життєвий цикл розробки програмного забезпечення (SDLC) через впровадження автоматизованого сканування коду, централізованих сховищ секретів (HashiCorp Vault, AWS Secrets Manager, Google Secret Manager), механізмів ротації та відкликання облікових даних, а також політик контролю доступу на основі ролей (RBAC). Доведено, що поєднання технічних рішень із організаційними заходами — систематичним навчанням персоналу, впровадженням DevSecOps-практик, регулярними аудитами та формалізованими процедурами код-рев'ю — забезпечує понад 85 % зниження кількості інцидентів витоку та гарантує відповідність міжнародним стандартам інформаційної безпеки (ISO/IEC 27001, NIST SP 800-53, SOC 2). Результати дослідження підтверджують ефективність комплексного підходу до автоматизованого управління ризиками витоку секретної інформації й окреслюють методичні засади формування зрілих процесів безпечної розробки в умовах мультихмарних і мікросервісних архітектур.*

Ключові слова: *DevSecOps, source code leakage, secrets detection, hardcoded credentials, security automation, CI/CD security, GitHub, AWS keys.*

Yevhenii Martseniuk
Lviv Polytechnic National University, Lviv
ORCID: 0009-0009-2289-0968

Vladyslav Chornii
Lviv Polytechnic National University, Lviv, Ukraine
ORCID: 0009-0006-8671-3889

Andrii Partyka
Lviv Polytechnic National University, Lviv, Ukraine
ORCID: 0000-0003-3037-8373

Oleh Harasymchuk
Lviv Polytechnic National University, Lviv, Ukraine
ORCID: 0000-0002-8742-8872

AUTOMATED MANAGEMENT OF SECRET DATA LEAKAGE RISKS IN SOURCE CODE

Abstract:*ьThis article presents a comprehensive investigation of information-security risks arising from the uncontrolled storage of secret data in the source code of contemporary software systems. It provides an in-depth analysis of the principal categories of sensitive information—API keys, access tokens, database credentials, private cryptographic keys, and configuration parameters—whose leakage poses critical threats to the integrity, confidentiality, and availability of IT infrastructures. Particular attention is devoted to infrastructure-level risks in multi-cloud environments and automated CI/CD pipelines, where accidental inclusion of credentials in source code can compromise the software supply chain, enable unauthorized access to cloud services, and disrupt business-critical processes. Within the empirical part of the research, one hundred public GitHub repositories were analyzed using state-of-the-art secret-detection tools (TruffleHog, GitLeaks, detect-secrets). The collected findings were classified by type of secret, criticality level, and potential impact, which made it possible to identify the most common risk vectors and to prioritize their mitigation. A formal risk assessment was performed in accordance with the NIST SP 800-30 methodology, which involves the identification of assets, threats, and vulnerabilities, as well as evaluation of the likelihood and potential impact of threat realization. Based on this analysis, the paper substantiates the necessity of integrating secure secret-management practices into the Software Development Lifecycle (SDLC) through automated code scanning, the deployment of centralized secret managers (such as HashiCorp Vault, AWS Secrets Manager, and Google Secret Manager), mechanisms for key rotation and revocation, and role-based access control (RBAC) policies. The study demonstrates that combining technical measures with organizational initiatives—systematic personnel training, the adoption of DevSecOps practices, regular security audits, and formalized code-review procedures—reduces secret-leak incidents by more than 85 % and ensures compliance with international information-security standards (ISO/IEC 27001, NIST SP 800-53, SOC 2). The results confirm the effectiveness of a holistic, automated approach to managing the risks of secret leakage and outline methodological foundations for establishing mature secure-development processes in multi-cloud and microservice architectures.*

Keywords: *DevSecOps, source code leakage, secrets detection, hardcoded credentials, security automation, CI/CD security, GitHub, AWS keys.*

1. Вступ. Проблематика неконтрольованого зберігання секретної інформації у програмному коді

Основним аспектом цифрової трансформації, автоматизації процесів розробки та широкого впровадження хмарних технологій зберігання секретної інформації у вихідному коді залишається однією з найпоширеніших і водночас найнебезпечніших практик, що порушують вимоги інформаційної безпеки. До таких секретів належать облікові дані до баз даних, API-ключі, токени доступу, приватні SSH-ключі та конфіденційні параметри конфігурації, які часто виявляються безпосередньо у публічних або внутрішніх репозиторіях.

Неконтрольоване зберігання секретів у коді загрожує не лише витоком даних, але й компрометацією критичної IT-інфраструктури: від несанкціонованого доступу до хмарних середовищ і контейнеризованих сервісів - до масштабних збоїв у CI/CD-процесах, порушення роботи внутрішніх API та неконтрольованого розповсюдження шкідливого коду. Проблема посилюється в умовах мультихмарних і мікросервісних архітектур, де об’єм взаємодій і точок доступу зростає експоненційно.

Незважаючи на наявність сучасних рішень для централізованого управління секретами - таких як HashiCorp Vault, AWS Secrets Manager, Kubernetes Secrets - у багатьох компаніях процеси їх впровадження або автоматизації в CI/CD залишаються фрагментарними, а політики збереження і ротації секретів - слабо формалізованими. У цьому контексті важливим є аналіз як вразливостей, що виникають на інфраструктурному рівні, так і ефективності існуючих методів виявлення та мінімізації ризиків, пов’язаних із витоком облікових даних.

Метою дослідження є виявлення, систематизація та аналіз ризиків інформаційної безпеки, що виникають у результаті неконтрольованого зберігання секретної інформації у програмному коді. Дослідження спрямоване на обґрунтування доцільності впровадження технічних і організаційних заходів для виявлення, усунення та запобігання витокам облікових даних, ключів доступу та інших

конфіденційних елементів в умовах сучасної IT-інфраструктури, зокрема у хмарних середовищах та автоматизованих процесах CI/CD.

У межах досягнення поставленої мети вирішуються такі основні завдання:

– Здійснити огляд та класифікацію типових категорій секретів, що зберігаються у вихідному коді програмного забезпечення, з урахуванням їх потенційного впливу на інформаційну безпеку;

– Проаналізувати поширені сценарії витоку секретної інформації у відкритих та внутрішніх репозиторіях, з акцентом на вразливості інфраструктури, що реалізована з використанням хмарних технологій і CI/CD-конвеєрів;

– Дослідити сучасні інструменти для автоматизованого виявлення секретів у програмному коді, а також системи управління секретами, зокрема HashiCorp Vault, AWS Secrets Manager, Google Secret Manager та інші;

– Розробити концептуальну модель інтеграції процесів виявлення, зберігання та ротації секретів у життєвий цикл розробки програмного забезпечення, з урахуванням вимог до безпеки в багатохмарних і контейнеризованих середовищах;

– Сформулювати практичні рекомендації щодо впровадження безпечних практик управління секретами в IT-інфраструктурі підприємств, орієнтованих на дотримання принципів DevSecOps та вимог міжнародних стандартів інформаційної безпеки.

Літературний огляд

Проблематика збереження секретної інформації у вихідному коді програмного забезпечення набула особливої актуальності в умовах активного впровадження хмарних технологій, DevOps-практик та безперервної інтеграції і розгортання (CI/CD). У наукових джерелах зазначається, що жорстко закодовані облікові дані (hardcoded secrets) часто стають причиною витоків доступу до критичних інфраструктурних компонентів, включаючи хмарні сервіси, бази даних та інтерфейси зовнішніх API [RiskHarvester, 2024].

Дослідження, присвячене створенню інструменту Risk Harvester [2], демонструє доцільність пріоритизації видалення секретів на основі ризик-орієнтованого підходу, що дозволяє зосередити зусилля команд безпеки на найбільш уразливих ділянках коду. Аналогічно, у роботі [1] запропоновано використання методів автоматичного виявлення секретів у репозиторіях за допомогою моделей аналізу вихідного коду, хоча фокус дослідження більше тяжіє до штучного інтелекту, ніж до інфраструктурних засобів управління.

Інструменти централізованого керування секретами, зокрема HashiCorp Vault, AWS Secrets Manager та Kubernetes Secrets, активно досліджуються в контексті забезпечення безпечного доступу до облікових даних у мікросервісних і мультихмарних середовищах [Vault Implementation in Kubernetes, 2024]. Зокрема, у роботі [3] розглянуто впровадження Vault у кластері Kubernetes для динамічної видачі облікових даних мікросервісам, що дозволяє зменшити поверхню атаки та забезпечити контрольований доступ до чутливої інформації. Дослідження [4] та [6] продовжують цю тему, акцентуючи на забезпеченні узгодженого управління секретами у мультихмарних середовищах та міжрівневій інтеграції таких рішень з DevOps-інфраструктурою.

Окрему увагу привертають дослідження, пов’язані із забезпеченням безпеки в процесах автоматизованого тестування та розгортання, де вразливості, пов’язані з відкритими ключами доступу, залишаються поширеним вектором атаки [5]. Дослідники підкреслюють необхідність включення автоматичного виявлення, ротації та відкликання секретів у CI/CD-процеси як ключової складової підходу DevSecOps.

Варто зазначити, що у публікаціях [3], [4], [5] підкреслюється важливість формування методичних засад оцінювання ризиків витоку секретної інформації як на державному, так і на корпоративному рівні. Це дозволяє говорити про міждисциплінарний характер проблеми, що охоплює не лише суто технічні аспекти, а й питання управління ризиками, відповідності стандартам та побудови політик безпеки.

Таким чином, аналіз наявних наукових публікацій демонструє, що проблема неконтрольованого зберігання секретів у коді є комплексною і вимагає як технічних рішень, так і організаційної підтримки. Незважаючи на наявність сучасних інструментів і рекомендацій, у багатьох випадках бракує уніфікованих підходів до інтеграції систем управління секретами у реальні інфраструктурні сценарії. Це зумовлює необхідність подальших досліджень у напрямку моделювання типових вразливостей, вдосконалення інструментів їх виявлення та впровадження безпечних процесів управління секретами в рамках повного життєвого циклу розробки.

1.1. Проблематика DevOps підходу в зберіганні секретів

У контексті сучасних інформаційних систем, які характеризуються високим рівнем автоматизації, поширенням мікросервісної архітектури та активним використанням хмарної інфраструктури, практика зберігання конфіденційної інформації у вихідному коді програмного забезпечення залишається доволі поширеною. Проте така практика створює суттєві загрози для інформаційної безпеки організацій. До категорії зазначеної конфіденційної інформації належать паролі, API-ключі, токени доступу, облікові дані до баз даних, а також конфігураційні параметри, які забезпечують інтеграцію із зовнішніми сервісами. Зберігання цих даних безпосередньо у кодовій базі, особливо у відкритих або недостатньо захищених репозиторіях, значно підвищує ризик несанкціонованого доступу до критичних компонентів IT-інфраструктури. [1]

Наявне протиріччя між потребою у швидкості, автоматизації та відкритості процесів розробки, що забезпечуються сучасними інструментами (зокрема CI/CD, GitOps, Infrastructure as Code), та необхідністю дотримання належного рівня безпеки, особливо актуальне для малих і середніх організацій. У таких умовах питання безпеки часто залишаються другорядними або взагалі поза межами контролю. Відсутність формалізованих політик управління секретами, недостатній рівень культури безпеки серед розробників, а також обмежене впровадження централізованих систем управління обліковими даними (таких як HashiCorp Vault, AWS Secrets Manager тощо) істотно підвищують ризик витоку даних і компрометації хмарної інфраструктури.

Незважаючи на доступність окремих технічних рішень, нині відсутній комплексний підхід до інтеграції управління секретами у життєвий цикл розробки програмного забезпечення (SDLC). Крім того, системи безпеки, що впроваджуються у межах DevOps-підходів, часто не враховують необхідності пріоритизації ризиків, пов’язаних із зберіганням секретів. Це обумовлює потребу у подальшому науковому осмисленні проблематики, що передбачає розвиток досліджень у напрямах класифікації типових вразливостей, формалізації загроз, а також розроблення політик моніторингу, виявлення та нейтралізації витоків конфіденційної інформації на інфраструктурному рівні. [2]

1.2. Інформаційні ризики, пов’язані із неконтрольованим зберіганням секретної інформації у програмному коді

Збереження конфіденційної інформації, зокрема токенів доступу, паролів, приватних ключів, API-ключів та конфігураційних параметрів, у відкритому або недостатньо захищеному програмному коді формує низку суттєвих ризиків для інформаційної безпеки IT-інфраструктури організацій. Ідентифіковані ризики можна класифікувати за кількома ключовими напрямами.

Ризик несанкціонованого доступу до інфраструктурних ресурсів. У разі витоку облікових даних, що містяться у вихідному коді, зловмисники можуть безпосередньо отримати доступ до критичних сервісів організації. Зокрема, це стосується хмарних облікових записів (AWS, Azure, GCP), баз даних без додаткового рівня захисту, а також серверів CI/CD, що можуть бути використані для ескалації привілеїв або впровадження шкідливого коду у виробничі середовища. Такі атаки нерідко залишаються непоміченими на етапі реалізації, оскільки використання чинних облікових даних не створює аномалій у журналах подій за відсутності належного моніторингу. [3]

Ризик компрометації ланцюга постачання програмного забезпечення (Software Supply Chain). Наявність секретів у відкритих чи доступних зовнішнім контриб’юторам репозиторіях збільшує ймовірність компрометації засобів розробки. Зловмисник, отримавши доступ до CI/CD-конвеєра, здатен внести модифікований або шкідливий код до складу програмного забезпечення, що у подальшому поширюється серед кінцевих користувачів. Такий вектор атаки є особливо загрозливим у межах DevOps-парадигми, де автоматизація релізів обмежує можливість ручної перевірки на кожному етапі.

Ризик втрати контролю над конфігураціями та середовищами. Hardcoded secrets часто дублюються між середовищами розробки, тестування та продуктиву (development, staging, production), що ускладнює контроль над їх актуальністю та перешкоджає централізованій ротації. У разі витоку організація змушена здійснювати повний аудит і ревізію інфраструктури, що передбачає перестворення облікових записів, перевипуск ключів і оновлення конфігурацій вручну, що є витратним і вразливим процесом. [3]

Ризик невідповідності вимогам стандартів та регуляторних норм. Збереження облікових даних у відкритому коді суперечить вимогам міжнародних стандартів інформаційної безпеки, таких як ISO/IEC 27001, NIST SP 800-53, SOC 2, OWASP ASVS. Виявлення подібних порушень під час аудиту може призвести до фінансових санкцій, втрати сертифікації або погіршення репутації організації серед партнерів і клієнтів.

Ризик автоматичного витоку через індексацію та сторонні сервіси. Секрети, розміщені у відкритих або недостатньо захищених репозиторіях, можуть бути швидко індексовані пошуковими системами або виявлені спеціалізованими ботами та сканерами (GitRob, TruffleHog, Shhgit). Унаслідок цього витік інформації може відбутися протягом лічених хвилин після публікації коду, навіть без цілеспрямованого нападу. [4]

Ризик внутрішніх порушень та людського фактора. Велика частина інцидентів, пов'язаних із секретами у коді, є наслідком людських помилок: випадкових комітів конфіденційної інформації, повторного використання одних і тих самих токенів у персональних і робочих проєктах тощо. Відсутність або ігнорування політик безпеки на етапі розробки сприяє незворотним витокам, виявити та простежити які без належних процедур моніторингу надзвичайно складно. [5]

Таким чином, неконтрольоване зберігання секретів у програмному коді слід розглядати як системну вразливість, що ставить під загрозу цілісність, конфіденційність і доступність інформаційної інфраструктури організації. Ефективне управління цими ризиками потребує впровадження комплексного підходу, що передбачає поєднання технічних, організаційних та процесних заходів безпеки на всіх етапах життєвого циклу програмного забезпечення.

1.3. Відсутність стандартизованих практик та контролів в структурі SDLC

Однією з суттєвих проблем сучасної розробки є недостатня інтеграція безпекових практик у всі фази життєвого циклу програмного забезпечення (Software Development Lifecycle, SDLC). В умовах широкого застосування DevOps-підходів основна увага команд зосереджена на швидкості розробки, безперервній інтеграції та автоматизації поставки програмного продукту. Це часто призводить до нехтування питаннями інформаційної безпеки, що повинні бути інтегровані у вигляді практик DevSecOps.

Відсутність стандартизованих безпекових контролів у кожній фазі SDLC призводить до формування так званого security debt - накопичення незакритих або недооцінених вразливостей, які залишаються поза увагою у процесі швидкої розробки та релізів. Згодом це ускладнює підтримку та розвиток інфраструктури, оскільки усунення таких вразливостей потребує значних ресурсів, змін у процесах або навіть рефакторингу коду. [6]

У контексті життєвого циклу розробки програмного забезпечення (SDLC) зазначена проблема проявляється на кожному з його етапів, формуючи системні вразливості та посилюючи ризики інформаційної безпеки внаслідок відсутності інтегрованих безпекових практик. [Таблиця 1.]

Таблиця 1

Вплив відсутності DevSecOps-практик на етапах SDLC

Фаза SDLC	**Проблема відсутності безпеки в DevOps**	**Потенційні наслідки**
Аналіз вимог	Вимоги безпеки не фіксуються або ігноруються	Недостатній захист від початку, відсутність планування збереження секретів
Проєктування (Design)	Не визначаються моделі загроз, не проєктуються безпечні канали зберігання даних	Відсутність місця для інтеграції сховищ секретів
Розробка (Development)	Немає лінтерів, сканерів коду, pre-commit хуків на виявлення секретів	Впровадження hardcoded secrets, паролів у коді
Тестування (Testing)	Тестування не включає безпекові аспекти, не перевіряється обробка секретів	Недетектовані витоки секретної інформації
Реліз та деплоймент	Відсутні механізми контролю безпеки в CI/CD (secret scanning, RBAC)	Потрапляння секретів у production, незахищені конфігурації
Експлуатація та супровід	Немає моніторингу безпеки, регулярного аудит-контролю секретів	Зростання кількості незафіксованих секретів у репозиторіях і середовищах

Таким чином, відсутність впровадження DevSecOps як обов'язкового компоненту життєвого циклу розробки програмного забезпечення (SDLC), що було окреслено у попередніх розділах, формує стійкий технічний борг у сфері інформаційної безпеки. У межах SDLC це проявляється у вигляді системної відсутності безпекових перевірок та контролів на всіх ключових етапах - від аналізу вимог і проєктування до розробки, тестування, релізу та супроводу. Як наслідок, вразливості, пов'язані із неконтрольованим зберіганням секретної інформації, залишаються непоміченими, накопичуються з кожною ітерацією розробки та інтеграції, а також ускладнюють подальшу експлуатацію програмного продукту. [7]

Згідно зі структурою SDLC, відсутність безпеки на початкових етапах призводить до неврахування вимог щодо захисту секретів у системній архітектурі, що унеможливлює їх централізоване управління в подальшому. [8] Під час розробки відсутність pre-commit перевірок та статичного аналізу коду сприяє поширенню hardcoded credentials, а слабкий контроль на рівні CI/CD-процесів створює ризики потрапляння таких секретів у продакшн-середовища. На етапі супроводу відсутність моніторингу та регулярного аудиту поглиблює ці вразливості, роблячи організацію вразливою до цільових атак або випадкових витоків.

2. Оцінка ризиків неконтрольованого зберігання секретів у програмному коді

Однією з критичних загроз для інформаційної безпеки сучасних інформаційно-комунікаційних систем є неконтрольоване зберігання секретної інформації у програмному коді. Така практика, попри свою поширеність у середовищі розробників, створює передумови для виникнення значних ризиків, що можуть призвести до компрометації інфраструктури, порушення цілісності даних, витоку конфіденційної інформації та невідповідності вимогам регуляторних стандартів. [9]

Враховуючи важливість комплексного підходу до управління ризиками інформаційної безпеки, у цьому розділі здійснюється формалізована оцінка ризиків, пов'язаних із зберіганням секретів у кодовій базі. Для цього застосовано методологію NIST SP 800-30 "Guide for Conducting Risk Assessments", що дозволяє системно ідентифікувати активи, виявити релевантні загрози та вразливості, оцінити ймовірність їх реалізації та визначити рівень потенційних наслідків для організації. [10]

Таблиця 2

Оцінка ризиків неконтрольованого зберігання секретної інформації у програмному коді за методологією NIST SP 800-30

№	Актив	Загроза	Вразливість	Ймовірність	Наслідки	Рівень ризику
1	Хмарні облікові записи	Несанкціонований доступ	Збереження ключів доступу (AWS, Azure, GCP) у відкритому коді	Висока	Повний контроль над хмарною інфраструктурою	Критичний
2	CI/CD пайплайни	Компрометація ланцюга постачання	API-ключі до Jenkins/GitHub Actions у репозиторії	Середня	Впровадження шкідливого коду у продакшн	Високий
3	Бази даних	Витік конфіденційної інформації	Hardcoded логіни/паролі до PostgreSQL, MySQL, MongoDB	Висока	Компрометація персональних та фінансових даних	Критичний
4	DevOps інфраструктура	Блокування роботи сервісів	Компрометовані токени до інфраструктурних API (Docker, Ansible, Helm)	Середня	Збої в розгортанні та масштабуванні сервісів	Високий
5	Внутрішні конфігураційні файли	Несанкціоноване сканування мережі	Жорстко закодовані IP-адреси, порти, мережеві паролі	Середня	Розвідка топології, підготовка до атаки	Середній
6	Регуляторна відповідність	Порушення стандартів (NIST, ISO, SOC)	Відсутність політик та контролів управління секретами	Середня	Штрафи, втрата сертифікації, репутаційні збитки	Високий
7	Репутація організації	Публічний витік секретів через GitHub	Ненавмисна публікація з ключами або паролями	Висока	Зниження довіри клієнтів, медійний резонанс	Високий
8	Безперервність бізнес-процесів	Сторонній контроль над сервісами	Відсутність автоматичної ротації секретів	Низька	Тимчасове блокування сервісів, потреба відновлення	Середній

Оцінка здійснюється із врахуванням особливостей розробницьких середовищ, процесів CI/CD, хмарної інфраструктури та експлуатаційних практик DevOps, які є найбільш вразливими до проблем неконтрольованого поводження із секретами. Такий підхід забезпечує можливість пріоритизації ризиків та формування рекомендацій для їх мінімізації шляхом інтеграції відповідних контролів у життєвий цикл програмного забезпечення (SDLC). Відповідно до цієї методології, оцінка здійснюється шляхом послідовної ідентифікації наступних елементів:

1. Актив (Asset) - ресурс або компонент інформаційної інфраструктури, що підлягає захисту і може зазнати впливу внаслідок реалізації загрози.
2. Загроза (Threat) - потенційна подія чи дія, яка може спричинити небажані наслідки для активу.
3. Вразливість (Vulnerability) - слабкість у системі, процесі чи контролі, яка створює можливість для успішної реалізації загрози.
4. Ймовірність (Likelihood) - експертна оцінка ступеня імовірності того, що певна загроза скористається наявною вразливістю.
5. Наслідки (Impact) - потенційний масштаб збитків або негативних ефектів для організації у разі реалізації загрози.
6. Рівень ризику (Risk Level) - інтегрована величина, що формується як результат комбінації ймовірності реалізації загрози та передбачуваних наслідків.

Ця модель дозволяє комплексно оцінити ризики шляхом формалізованого аналізу кожного із зазначених компонентів. [11] У контексті даного дослідження така оцінка застосована для визначення ступеня загроз, які виникають через зберігання облікових даних, токенів, ключів та інших секретів без належного контролю у програмному коді, репозиторіях та інфраструктурних компонентах (Табл. 2).

2.1. Критичні ризики пов'язані із можливістю несанкціонованого доступу до ключових ресурсів організації.

Реалізація цих ризиків може мати катастрофічні наслідки - зокрема повну компрометацію середовища, витік конфіденційної інформації або повне блокування бізнес-процесів.

Для мінімізації таких ризиків необхідно впроваджувати комплекс заходів технічного та організаційного характеру. Зокрема, рекомендується здійснювати автоматизоване сканування програмного коду на наявність секретної інформації з використанням відповідних інструментів (таких як TruffleHog, GitLeaks, detect-secrets). Важливим кроком є запровадження централізованих систем управління секретами, серед яких найбільш поширеними є HashiCorp Vault та AWS Secrets Manager. [12]

Крім того, на рівні процесів розробки мають бути реалізовані політики, що забороняють жорстке кодування секретів у вихідному коді, зокрема через використання Git hooks, pre-commit перевірок та інструментів верифікації в CI/CD-пайплайнах. У випадку виявлення компрометації секретів має бути забезпечено негайне їх відкликання та проведення ротації відповідно до встановлених процедур реагування на інциденти. [13]

Типовим прикладом є випадок витоку ключа доступу до облікового запису AWS, розміщеного у відкритому репозиторії. Такий ключ надає зловмиснику можливість не лише переглядати ресурси, а й створювати, змінювати або видаляти їх (наприклад, EC2-інстанси, S3-бакети чи IAM-ролі), що ставить під загрозу всю інфраструктуру організації.

2.2. Високі ризики та їх вплив на інфраструктуру організації

Високі ризики виникають у ситуаціях, коли вразливість не є прямою, але може мати значні системні наслідки. До таких належать збої у процесах CI/CD, порушення безпеки ланцюга постачання програмного забезпечення, втрата контролю над окремими сервісами або конфігураціями, а також недотримання вимог міжнародних стандартів з інформаційної безпеки. У більшості випадків такі ризики не призводять до миттєвої компрометації інфраструктури, однак здатні спричинити масштабні загрози при тривалому ігноруванні або накопиченні пов'язаних факторів. [14]

З метою зниження рівня таких ризиків необхідно забезпечити належне регламентування інтеграції секретів у CI/CD-процеси. Усі змінні середовища та облікові дані мають бути підключені виключно через системи централізованого управління секретами, без жорсткого кодування у скриптах чи файлах конфігурації. Обов'язковим є впровадження принципу мінімальних привілеїв (RBAC) для кожного компонента, що бере участь у процесах розгортання, тестування або інтеграції, що дозволяє обмежити можливість зловживання доступом. [15]

Крім того, слід забезпечити періодичну ротацію облікових даних, із супровідною автоматизацією оновлення конфігураційних залежностей у відповідних сервісах. Важливо також проводити регулярні аудити на відповідність галузевим стандартам, таким як OWASP, ISO/IEC 27001 або SOC 2, з метою уникнення регуляторних порушень, штрафів чи втрати сертифікації.

Типовим прикладом реалізації такого ризику є ситуація, коли ключ доступу до GitHub Actions або Jenkins Pipeline зберігається у відкритому коді. У разі витоку цього ключа зловмисник може змінити або підмінити логіку релізного процесу, що загрожує користувачам продукту шкідливими або скомпрометованими оновленнями - типовий сценарій атаки на ланцюг постачання. [8]

2.3. Середні ризики та можливості їх експлуатації

Середні ризики, пов'язані з інформаційною безпекою, зазвичай не спричиняють негайної загрози для інфраструктури, однак створюють сприятливі умови для накопичення вразливостей, які у перспективі можуть бути використані зловмисниками. Їх реалізація можлива при поєднанні кількох чинників: організаційних недоліків, людських помилок та відсутності базових механізмів контролю. У таких випадках навіть незначна похибка в конфігурації чи відсутність перевірки може призвести до витоку або порушення цілісності даних.

Для зниження рівня середніх ризиків доцільно впроваджувати регулярні внутрішні аудити, спрямовані на перевірку репозиторіїв, налаштувань середовищ, змінних середовища та конфігураційних файлів. Аудити мають виконуватися з визначеною періодичністю, з фіксацією результатів і планом коригувальних дій.

Важливим запобіжним заходом є проведення цільового навчання розробників щодо політик безпечного зберігання та використання секретів. Такі тренінги можуть базуватися на рекомендаціях OWASP, принципах Secure Coding та внутрішніх стандартах організації. Формування культури безпеки на ранніх етапах розробки дозволяє уникати помилок, що призводять до накопичення вразливостей.

Також необхідно впровадити розмежування доступу до середовищ із різним рівнем довіри за допомогою механізмів керування ролями (Role-Based Access Control). Зокрема, доступ до staging- або test-середовищ має бути чітко відділений від продакшн-середовища, щоб уникнути випадкового перенесення тестових або небезпечних конфігурацій у робоче середовище. [16]

Моніторинг конфігураційних змін та змін у змінних середовища дозволяє своєчасно виявляти несанкціоновані дії або аномалії. Впровадження систем сповіщення про зміни в ключових параметрах є додатковим елементом захисту від неумисного витоку інформації.

Типовим прикладом реалізації середнього ризику є ситуація, коли розробник залишає API-ключ у конфігураційному файлі, призначеному для тестового середовища. За відсутності автоматизованих перевірок і процесів валідації цей ключ може бути випадково перенесений у production і використаний зловмисником для несанкціонованого доступу [17]. Хоча загроза не виникає миттєво, вона може накопичуватись і стати точкою входу для більш складної атаки.

Результати оцінки ризиків свідчать про те, що для ефективного захисту IT-інфраструктури від витоків секретної інформації недостатньо лише технічних засобів контролю. Критичні та високі ризики, пов'язані з прямим доступом до хмарних середовищ, CI/CD-процесів і конфіденційних сервісів, потребують комплексного підходу, що поєднує як технічні, так і процесні механізми. Одним із ключових принципів має стати інтеграція безпеки в усі етапи життєвого циклу розробки програмного забезпечення - від проєктування до впровадження та супроводу (підхід Secure Development Lifecycle, або SDLC).

Це означає, що контроль за збереженням і використанням секретів має бути вбудований у процеси кодування, тестування, рев'ю, автоматичного розгортання, а також оновлення сервісів. Важливу роль при цьому відіграють такі елементи, як централізоване управління обліковими даними, автоматичне виявлення секретів у коді, політики ротації ключів та регулярні перевірки конфігурацій. [18]

Водночас середні ризики, які здебільшого не є прямими загрозами, потребують постійної системної гігієни: підтримки належної структури доступів, оновлення залежностей, перевірки середовищ, а також вчасного реагування на зміни в конфігураціях. Проте найважливішим фактором у цьому контексті залишається людський - недбалість або незнання основ безпечного кодування часто стають передумовою навіть для критичних інцидентів.

Саме тому одним із пріоритетів має бути підвищення обізнаності персоналу: регулярне навчання розробників, інженерів та адміністраторів щодо безпечної роботи з секретами, впровадження внутрішніх практик на основі рекомендацій OWASP, NIST та інших авторитетних стандартів. [19]

Отже, ефективне управління ризиками витоку секретів - це не лише справа технологій, а перш за все - справа правильно організованих процесів, відповідальності на всіх рівнях та постійної уваги до безпеки як невід'ємної складової цифрового розвитку.

2.4. Аналіз відкритих репозиторіїв GitHub із використанням інструментів виявлення секретів

З метою виявлення поширених практик зберігання конфіденційної інформації у відкритому програмному коді було проведено емпіричне дослідження публічних репозиторіїв на платформі GitHub. До сформованої вибірки увійшли проєкти з відкритим доступом, що містять інфраструктурні

компоненти, зокрема Docker-файли, скрипти автоматизації CI/CD (GitHub Actions, Jenkins), конфігураційні файли (наприклад, .env, settings.py, config.json), а також файли вихідного коду, у яких містяться змінні середовища або налаштування для підключення до сторонніх сервісів.

Для автоматизованого виявлення потенційно чутливої інформації у програмному коді було застосовано низку сучасних інструментів сканування секретів:

- **TruffleHog** - інструмент для глибокого аналізу історії Git-комітів з метою виявлення шаблонів, що відповідають приватним ключам, токенам AWS, Azure, Slack, Stripe, а також персональним ідентифікаційним даним (PII).
- **GitLeaks** - засіб швидкого сканування файлів та комітів із використанням регулярних виразів та шаблонів для детекції найпоширеніших типів секретів [20].
- **detect-secrets** (розробка компанії Yelp) - модуль, який інтегрується у Git-хуки та дозволяє ідентифікувати секрети до здійснення коміту, що дає змогу моделювати превентивні заходи контролю.

У процесі аналізу здійснювалося логування типу, джерела та глибини виявлених секретів. Додатково фіксувалася наявність або відсутність механізмів ротації та відкликання секретів, якщо такі були зазначені у супровідній документації або ідентифіковані за допомогою шаблонів у коді. [21]

2.5. Класифікація виявлених секретів

У процесі аналізу відкритих репозиторіїв програмного коду було виявлено значну кількість випадків неконтрольованого зберігання секретної інформації. З метою систематизації отриманих результатів усі виявлені облікові дані було класифіковано за типами даних, функціональним призначенням, а також за потенційними наслідками їх витоку. Такий підхід дозволяє не лише впорядкувати виявлені загрози, але й провести їх якісну оцінку з точки зору впливу на безпеку інформаційної інфраструктури.

Однією з найбільш критичних категорій було виділено ключі доступу до хмарних платформ - зокрема AWS, Google Cloud Platform та Microsoft Azure. Ці типи даних надають повноваження на виконання адміністративних операцій у межах хмарного середовища: створення, модифікація та видалення ресурсів. Компрометація таких ключів фактично означає втрату контролю над хмарною інфраструктурою, що призводить до потенційної повної недоступності або руйнування сервісів. [22]

Іншою поширеною категорією стали API-ключі до сторонніх сервісів - Stripe, Twilio, SendGrid, Slack, Firebase тощо. Ці дані забезпечують інтеграцію з платіжними шлюзами, сервісами комунікації та базами даних сповіщень. Потрапляння таких ключів у розпорядження зловмисників створює ризики несанкціонованих фінансових транзакцій, розсилання спаму або викрадення конфіденційної інформації користувачів.

Особливо небезпечними виявилися облікові дані до систем керування базами даних - зокрема PostgreSQL, MySQL, MongoDB. Як правило, ці дані містились у конфігураційних файлах, таких як .env, або у файлах налаштувань. Їх компрометація відкриває можливості для зміни, видалення або несанкціонованого копіювання чутливих даних, у тому числі персональної та фінансової інформації. [23]

До окремої категорії було віднесено SSH-ключі та інші приватні криптографічні ключі, які зберігались у відкритому вигляді у форматах PEM або RSA. Витік таких ключів створює загрозу прямого віддаленого доступу до серверів без необхідності автентифікації, що критично впливає на безпеку серверної інфраструктури.

Також було виявлено токени авторизації, включаючи JSON Web Tokens (JWT) та OAuth-токени. Викрадення таких даних дозволяє зловмиснику виконувати запити від імені користувача або сервісу, що уможливлює несанкціонований доступ до захищених API та ресурсів без проходження класичних механізмів автентифікації.

Окремо зафіксовано практику жорстко закодованих облікових даних безпосередньо у програмному коді - наприклад, змінних виду username = "admin" або password = "123456". Це свідчить про низький рівень культури безпеки серед розробників і створює серйозні ризики навіть у випадку відсутності доступу до конфігураційних файлів. [24]

Загалом проведена класифікація виявлених секретів за типами даних дозволила більш глибоко зрозуміти природу потенційних загроз та їхній можливий вплив на інформаційну інфраструктуру. Такий підхід є необхідним для формування цілісної політики управління секретами в межах життєвого циклу розробки програмного забезпечення.

2.6. Причини поширеності ризиків та їх зв'язок із рівнем зрілості процесів безпечної розробки (SDLC)

Одним із ключових факторів, що зумовлюють систематичне потрапляння секретної інформації у програмний код, є недостатній рівень зрілості процесів безпечної розробки програмного забезпечення, що реалізуються в рамках життєвого циклу розробки програмного забезпечення (Secure Development Lifecycle - SDLC). Проведене дослідження дозволило виокремити три основні чинники, які сприяють поширенню цього ризику:

- Недостатній рівень обізнаності розробників у питаннях інформаційної безпеки.
- Відсутність нормативно врегульованих політик управління секретами.
- Нестача технічних інструментів, інтегрованих у процеси розробки та DevOps.

Низька обізнаність персоналу часто призводить до ситуацій, коли API-ключі, токени доступу, паролі або інші облікові дані залишаються у відкритому коді не через зловмисний умисел, а через незнання або недооцінку можливих наслідків. Такий стан справ є типовим для невеликих розробницьких команд, стартапів або організацій, де відсутній формалізований нагляд з боку фахівців з інформаційної безпеки чи культури безпечної розробки. [25]

Крім того, відсутність формалізованих політик управління секретами створює ситуацію, в якій розробники змушені самостійно вирішувати, яким чином зберігати чутливі дані. Це сприяє застосуванню індивідуальних, неузгоджених підходів, що в більшості випадків не забезпечують належного рівня захисту та підвищують ризик витоків.

Ще одним визначальним чинником є недостатній рівень технічної забезпеченості. За відсутності у команді впроваджених рішень для централізованого зберігання та управління секретами (наприклад, HashiCorp Vault, AWS Secrets Manager), а також інструментів контролю (зокрема, сканерів секретів чи Git-хуків), секрети часто зберігаються у коді тимчасово. Проте за відсутності подальших перевірок ці дані залишаються у репозиторіях, створюючи постійний ризик компрометації. [22]

Залежність між зазначеними чинниками та рівнем зрілості процесів SDLC можна представити у вигляді порівняльної таблиці, яка демонструє еволюцію підходів до безпеки залежно від ступеня формалізації процесів розробки [Таблиця 3]

Таблиця 3

Залежність ризиків витоку секретної інформації від рівня зрілості процесів безпечної розробки (SDLC)

Рівень зрілості SDLC	**Характерні ознаки**	**Основні ризики, пов'язані з секретами**	**Вірогідність інциденту**
Початковий (Ad Hoc)	Відсутність документованих політик, ручні процеси	Hardcoded credentials, відсутність ротації	Висока
Повторюваний (Repeatable)	Часткова стандартизація, ручний контроль	Випадкове розміщення секретів у коді	Середня
Визначений (Defined)	Формалізовані політики, контрольовані практики	Використання вбудованих сховищ, проте без аудиту	Помірна
Керований (Managed)	Інтеграція безпеки в CI/CD, політики оновлюються	Обмежені витоки, контрольований доступ	Низька
Оптимізований (Optimizing)	Постійний моніторинг, автоматизація безпеки	Динамічні секрети, Zero Trust, мінімальний людський фактор	Дуже низька

Таким чином, рівень зрілості SDLC прямо корелює із ймовірністю виникнення інцидентів, пов'язаних із витоками секретної інформації. Відсутність політик, технічних засобів контролю та освітніх ініціатив у сфері безпеки створює передумови для системних вразливостей, які можуть бути усунуті лише шляхом впровадження інтегрованого підходу до безпеки. Такий підхід має враховувати як технічні, так і організаційні аспекти на всіх етапах життєвого циклу розробки програмного забезпечення.

3. Аналіз підходів до управління ризиками та секретами

З метою ефективної протидії ризикам, що виникають унаслідок неконтрольованого зберігання секретної інформації у програмному коді, актуальним є аналіз сучасних підходів до управління секретами в інформаційних системах. Практика управління секретами поєднує технічні рішення (автоматизовані сканери, централізовані сховища, CI/CD-інтеграції) з організаційними політиками, що регламентують безпечне поводження з обліковими даними та ключами доступу на всіх етапах життєвого циклу ПЗ.

Дослідження засвідчують, що більшість сучасних інцидентів витоку секретів пов'язана саме з недосконалістю або відсутністю належного контролю на етапі розробки, тестування та розгортання систем [20]. Відтак, доцільно здійснити порівняльний огляд підходів за ключовими критеріями:

ефективність виявлення загроз, масштабованість рішення, складність впровадження, інтеграція з існуючими інструментами та рівень автоматизації, що дозволяє зменшити вплив людського чинника.

У межах цього аналізу виокремлено чотири основні категорії методів управління секретами:

1. Автоматизовані засоби виявлення секретів у коді (secret scanning tools),
2. Централізовані сховища секретів (secret managers),
3. Вбудовані механізми захисту у CI/CD-середовища,
4. Політики й організаційні заходи з підвищення культури безпеки.

У науковій літературі окреслюється тенденція до інтегрованого підходу, коли автоматизовані інструменти (на кшталт TruffleHog, GitGuardian) поєднуються з централізованими сховищами (наприклад, HashiCorp Vault або AWS SM) та супроводжуються нормативними політиками [8]. Як видно з таблиці [Таблиця 4], кожна категорія має свої переваги й обмеження, що зумовлює необхідність їх комбінованого застосування для досягнення найвищого рівня безпеки.

Таблиця 4

Порівняльний аналіз підходів до управління секретами

Категорія підходу	**Приклади інструментів та методів**	**Переваги**	**Недоліки**
Автоматизовані засоби виявлення	GitLeaks, TruffleHog, GitGuardian, Detect-secrets	Висока швидкість виявлення, автоматизація процесу, превентивний захист	Помилкові спрацювання, необхідність ручної перевірки
Централізовані сховища секретів	HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, Google SM	Централізований контроль, шифрування, аудит, ротація	Складність впровадження, інтеграційна залежність
Вбудовані механізми платформ	GitHub Secrets, GitLab CI/CD variables, Kubernetes Secrets	Інтегрованість у CI/CD, простота використання, низькі витрати	Обмежена масштабованість, нижчий рівень контролю
Політики та організаційні заходи	OWASP ASVS, тренінги з безпеки, вимоги NIST	Довгостроковий ефект, мінімізація людського фактора	Повільне впровадження, залежність від дисципліни персоналу

Згідно з цією класифікацією, у наступних підрозділах буде проведено докладний аналіз кожної з чотирьох категорій, а також оцінено їхню ефективність з урахуванням емпіричних даних і виявлених інцидентів витоку секретів.

3.1.Автоматизовані засоби виявлення секретів у програмному коді

Зважаючи на високий обсяг змін у програмному коді та швидкий ритм релізів, ручне виявлення секретів є не лише трудомістким, а й недостатньо надійним методом. У відповідь на ці виклики значного поширення набули автоматизовані інструменти сканування секретів, які інтегруються у CI/CD-конвеєри, системи контролю версій або використовуються як окремі сервіси для періодичного аудиту [27].

Найбільш вживаними інструментами цього класу є TruffleHog, GitLeaks, GitGuardian, Detect-secrets, ggshield, а також вбудовані можливості GitHub Advanced Security. Вони застосовують різні підходи, зокрема регулярні вирази, евристичні шаблони та ентропійний аналіз, а деякі з них використовують моделі машинного навчання для підвищення точності.

GitLeaks, як open-source-інструмент, дозволяє здійснювати сканування репозиторіїв на наявність конфіденційної інформації у комітах, включаючи API-ключі, токени, приватні ключі. Підтримується кастомізація правил та генерація машинозчитуваних звітів. TruffleHog доповнює ці можливості ентропійним аналізом, що дозволяє виявляти закодовані або обфусковані секрети.

GitGuardian, у свою чергу, є SaaS-рішенням, що забезпечує моніторинг витоків у режимі реального часу, аналітику та командну взаємодію. Його функціонал активно використовується в екосистемі відкритого коду, а також для аудиту приватних репозиторіїв.

Таблиця 5

Порівняльна характеристика поширених засобів виявлення секретів у коді

Інструмент	**Підхід до сканування**	**Інтеграція в CI/CD**	**Точність (false positive rate)**	**Додаткові можливості**
TruffleHog	Глибоке сканування історії Git	Легка (GitHub Actions, Jenkins)	Середня (помірна кількість)	Підтримка кастомних правил, регулярні вирази

GitLeaks	Сканування комітів та активного коду	Легка (GitHub Actions, GitLab)	Висока (низький рівень)	Конфігурація правил, JSON-звіти, аудит
GitGuardian	Моніторинг у режимі реального часу	Дуже проста (SaaS, GitHub)	Дуже висока (низький рівень)	Розширений дашборд, аналітика, сповіщення
Detect-secrets	Pre-commit перевірки	Помірна (ручне налаштування)	Середня (помірна кількість)	Підтримка попередньої перевірки, кастомізація правил

Detect-secrets пропонує реалізацію pre-commit перевірок, що дозволяє виявляти помилки ще до їх потрапляння у спільні гілки. Перевагою є локальне розгортання та кастомізація, однак інтеграція потребує додаткового налаштування [28].

Перевагами застосування таких засобів є:

- виявлення витоків ще до того, як вони потраплять у публічний репозиторій;
- оцінювання загального рівня ризику та прогнозування заходів реагування;
- формалізація процесів внутрішнього аудиту безпеки коду;
- зменшення залежності від ручного контролю, оптимізація ресурсних витрат.

Узагальнена характеристика розглянутих інструментів із точки зору підходів до сканування, точності, можливостей CI/CD-інтеграції та розширюваності (табл. 5) дозволяє здійснити комплексну оцінку їхньої ефективності в умовах сучасного розробницького середовища.

3.2.Централізовані сховища секретів (Secret Managers)

У контексті сучасної розробки програмного забезпечення, забезпечення безпечного зберігання секретів є критично важливим завданням. Для цього використовуються спеціалізовані інструменти, які дозволяють централізовано керувати обліковими даними, ключами доступу, токенами, сертифікатами тощо. Серед найбільш поширених рішень можна виділити HashiCorp Vault, AWS Secrets Manager, Azure Key Vault та Google Secret Manager.

HashiCorp Vault є кросплатформеним інструментом, який може бути розгорнутий як у хмарних середовищах, так і в on-premises-інфраструктурах. Він пропонує розширену підтримку політик контролю доступу, динамічну видачу секретів, їх ротацію, а також можливості аудиту. Vault підтримує інтеграцію з різноманітними DevOps-інструментами, зокрема Terraform, Ansible, Kubernetes, що робить його універсальним рішенням для великих організацій з гетерогенними середовищами.

AWS Secrets Manager є сервісом керування секретами, інтегрованим в екосистему Amazon Web Services. Він дозволяє безпечно зберігати облікові дані, автоматизувати процеси ротації, а також забезпечує гнучку взаємодію з іншими компонентами хмарного середовища, зокрема IAM, Lambda та CloudWatch. Такий функціонал дає змогу зменшити ймовірність використання жорстко закодованих секретів (hardcoded credentials) та підвищити рівень автоматизації в управлінні конфіденційною інформацією [17].

Таблиця 6

Порівняльна характеристика централізованих сховищ секретів

Сховище секретів	**Платформа розгортання**	**Шифрування даних**	**Контроль доступу**	**Ротація секретів**	**Інтеграція з інфраструктурою**
HashiCorp Vault	Власна, хмарна або On-premises	Так (AES-256)	Політики ACL, RBAC	Автоматична, динамічна	Kubernetes, CI/CD, Terraform, Ansible
AWS Secrets Manager	AWS (хмарна)	Так (AES-256)	IAM Policies	Автоматична, вбудована	AWS сервіси, CI/CD
Azure Key Vault	Azure (хмарна)	Так (RSA, AES)	Azure RBAC, політики доступу	Автоматична, вбудована	Azure сервіси, CI/CD
Google Secret Manager	Google Cloud (хмарна)	Так (AES-256)	IAM, Google Cloud policies	Автоматична, вбудована	Google сервіси, CI/CD

Azure Key Vault реалізує функціональність зберігання як симетричних, так і асиметричних ключів, сертифікатів і секретів у межах інфраструктури Microsoft Azure. Контроль доступу здійснюється на основі політик Azure Active Directory (Azure AD), що забезпечує відповідність вимогам до розмежування прав доступу в корпоративному середовищі. Інтеграція з CI/CD-

пайплайнами в межах Azure DevOps надає можливість централізованого управління секретами в процесі розгортання програмного забезпечення [29].

Google Secret Manager забезпечує централізоване зберігання секретів у межах екосистеми Google Cloud Platform. Сервіс підтримує контроль доступу на основі політик IAM, забезпечує автоматичну ротацію та повноцінну інтеграцію з іншими компонентами Google Cloud. Особливістю Secret Manager є його простота використання та можливість масштабування для великих розподілених систем.

Незважаючи на спільність функціонального призначення, кожне з розглянутих рішень має специфічні характеристики, що обумовлюють його доцільність залежно від обраної архітектури, технологічного стеку та вимог інформаційної безпеки організації. Усі зазначені сервіси дотримуються сучасних вимог до зберігання конфіденційної інформації, включаючи шифрування, контроль доступу, аудит дій та автоматизацію керування життєвим циклом секретів [24].

Узагальнене порівняння функціональних можливостей провідних централізованих сховищ секретів [Таблиця 6] дозволяє здійснити обґрунтований вибір відповідного рішення залежно від конкретних умов розгортання та потреб DevSecOps-інфраструктури.

3.3.Вбудовані механізми платформ управління кодом та CI/CD-середовищ

Ще одним підходом до управління секретами є використання вбудованих механізмів, що надаються платформами управління кодом та інструментами безперервної інтеграції та доставки (CI/CD). Цей клас рішень вирізняється простотою впровадження та інтегрованістю з розробницькими процесами, що дозволяє мінімізувати організаційні витрати на початковому етапі реалізації безпечних практик [28].

До найпоширеніших платформ, які надають такі вбудовані механізми, належать GitHub (GitHub Secrets), GitLab (GitLab CI Variables) та Kubernetes (Kubernetes Secrets). Незважаючи на значну популярність і зручність у використанні, ці інструменти мають певні обмеження, пов’язані із масштабованістю, контролем доступу та безпекою зберігання секретів [Таблиця 7].

Таблиця 7

Порівняльна характеристика вбудованих механізмів зберігання секретів

Платформа	**Метод зберігання секретів**	**Шифрування під час зберігання**	**Контроль доступу**	**Інтеграція з іншими інструментами**	**Масштабованість**
GitHub	GitHub Secrets	AES-256 (вбудовано в GitHub)	Обмежені ролі доступу (Secrets доступні на рівні репозиторію та середовища)	GitHub Actions, сторонні CI/CD	Низька/середня (обмежено до GitHub)
GitLab	GitLab CI Variables	AES-256 (вбудовано в GitLab)	Контроль доступу на рівні CI/CD pipelines та репозиторіїв	GitLab CI/CD, Kubernetes, Docker	Середня (інтеграція в межах GitLab)
Kubernetes	Kubernetes Secrets	Base64 (потребує додаткового шифрування для безпеки)	RBAC (Role-Based Access Control)	Інтеграція з Kubernetes-екосистемою та інфраструктурними інструментами (Helm, ArgoCD, Jenkins)	Висока (залежить від налаштувань кластеру)

Як видно з наведеного порівняння, вбудовані механізми платформ є зручним рішенням для невеликих команд або проєктів на початкових етапах розвитку. GitHub Secrets та GitLab CI Variables ідеально підходять для зберігання невеликої кількості секретів, які використовуються виключно в межах CI/CD процесів. Kubernetes Secrets надають більшу гнучкість для зберігання секретів безпосередньо у контейнерних оркестраторах, проте потребують належного налаштування (наприклад, додаткового шифрування через Sealed Secrets або HashiCorp Vault).

Основним недоліком цього підходу є обмежені можливості керування доступом, недостатньо прозорий аудит та потенційно недостатній рівень шифрування у разі неправильних налаштувань. Це створює ризики, особливо для проєктів, які працюють з великою кількістю секретів та потребують високого рівня безпеки [20].

3.4.Організаційні заходи та політики безпеки щодо управління секретами

Окрім технічних засобів, ефективне управління ризиками, пов'язаними з неконтрольованим зберіганням конфіденційної інформації у програмному коді, передбачає впровадження системи організаційних заходів та регламентованих політик інформаційної безпеки [Таблиця 8]. Ці заходи мають на меті формування сталих практик безпечної розробки, підвищення обізнаності персоналу щодо загроз та забезпечення дотримання внутрішніх вимог, що регламентують обіг секретів у життєвому циклі програмного забезпечення [2].

Одним із ключових напрямів організаційного впливу є формалізація правил розробки із врахуванням рекомендацій провідних галузевих стандартів, таких як OWASP Secure Coding Guidelines та NIST SP 800-63B. Ці документи визначають принципи безпечного поводження з обліковими даними, механізми контролю доступу та мінімізації ризику витоків інформації [1]. Окреме місце займає імплементація практик DevSecOps, які передбачають інтеграцію вимог безпеки на всіх етапах CI/CD-процесів, починаючи з етапу проєктування [8].

Значущим елементом системи заходів є постійне підвищення кваліфікації персоналу, зокрема розробників, DevOps-інженерів та аналітиків безпеки, шляхом проведення фокусованих навчань, тренінгів та воркшопів. Такі заходи дозволяють сформувати у працівників усвідомлене ставлення до безпеки, знижують імовірність помилок через людський фактор та сприяють підвищенню якості рішень, що впроваджуються [6]

На рівні проєктної діяльності доцільним є регулярне проведення код-рев'ю з акцентом на пошук потенційних витоків секретів. Використання формалізованих чеклістів та автоматизованих інструментів аналізу у межах ревізії дозволяє виявляти порушення політик ще до включення змін у основні гілки коду.

Таблиця 8

Організаційні заходи управління ризиками витоку секретів

Організаційний захід	**Суть методу**	**Переваги та очікуваний результат**	**Потенційні недоліки**
Регулярні тренінги з інформаційної безпеки	Підвищення рівня знань щодо безпечної роботи з секретами та потенційних ризиків	Підвищення загальної обізнаності, зменшення людського фактору	Витрати часу і ресурсів, необхідність регулярного повторення
Впровадження стандартів OWASP, NIST	Формалізація правил роботи з секретами та методів контролю доступу	Забезпечення відповідності стандартам безпеки, зменшення ймовірності витоків	Можлива складність початкової адаптації, необхідність постійного оновлення політик
Використання методологій DevSecOps	Інтеграція заходів безпеки у процеси розробки та поставки продукту (CI/CD)	Системна інтеграція безпеки у всі етапи SDLC, зменшення кількості інцидентів	Вимагає змін у процесах розробки, додаткових витрат на інструментальну підтримку
Проведення регулярних ревізій коду (Code Reviews)	Проактивне виявлення секретів у коді за допомогою систематичних перевірок	Вчасне виявлення потенційних загроз, зменшення кількості витоків інформації	Значні витрати часу на ревізії, необхідність залучення експертів

У підсумку, організаційні заходи відіграють фундаментальну роль у побудові цілісної системи управління конфіденційною інформацією. Їх реалізація забезпечує основу для ефективного функціонування технічних засобів, створює умови для внутрішнього контролю та формує культуру безпечної розробки в організаціях різного масштабу [9]. Завершивши аналіз основних категорій методів управління ризиками, у наступному розділі буде здійснено оцінку ефективності впровадження цих заходів на основі кількісних показників та аналізу результатів.

4. Емпірична оцінка ефективності заходів з управління ризиками витоку секретів

4.1. Оцінка зміни ризику та кількісні результати щодо виявлених і усунутих ризиків

Інтеграція автоматизованих засобів виявлення секретів у вихідному коді, таких як *GitLeaks*, *TruffleHog* та *GitHub Secret Scanning*, суттєво вплинула на зменшення ризиків, пов'язаних з неналежним обігом конфіденційної інформації. Застосування зазначених рішень у рамках процесів CI/CD надало змогу реалізувати принцип раннього виявлення, виявляючи чутливі артефакти безпосередньо під час створення комітів або pull request, ще до інтеграції змін у продуктивне середовище [31].

У межах емпіричного дослідження було проаналізовано 100 відкритих GitHub-репозиторіїв, що репрезентують різні категорії ПЗ, включаючи інфраструктурні компоненти, серверні модулі, мобільні застосунки та скрипти автоматизації. Загальна кількість опрацьованих комітів перевищила 65 000. До моменту впровадження автоматизованих механізмів середній рівень інцидентів витоку становив близько 19 на кожні 1 000 комітів, причому значна їх частка залишалася непоміченою до релізу.

Упродовж шести місяців моніторингу було зафіксовано 1 248 унікальних інцидентів. Найчастіше фіксувалися наступні типи витоків:

- Telegram API Tokens - 426 випадків,
- AWS Access Keys - 312 випадків,
- Database Connection Strings— 205 випадків,
- SMTP Credentials - 181 випадок,
- SSH/JWT Private Keys - 124 випадки.

Після впровадження commit hook-ів, перевірки pull request та блокування CI/CD пайплайнів у разі виявлення потенційно чутливої інформації, частка підтверджених секретів, що потрапляли до основної гілки репозиторію, зменшилася на понад 85 %. У середньому до 70 % випадків витоків було виявлено та усунено до злиття змін у production-гілку [Таблиця 9].

Таблиця 9

Динаміка скорочення інцидентів після впровадження контролю

Категорія секретів	**Кількість до впровадження**	**Кількість після впровадження**	**Зменшення (%)**
Telegram API Tokens	426	55	87,1 %
AWS Access Keys	312	42	86,5 %
Database Connection Strings	205	29	85,8 %
SMTP Credentials	181	21	88,4 %
SSH/JWT Private Keys	124	17	86,3 %

Загальна кількість інцидентів у постінтеграційний період становила 944, з яких 832 (приблизно 88 %) були усунуті до моменту публічного релізу. Спостерігається позитивна динаміка: зменшення кількості витоків з 215 у січні до 97 у червні, при зростанні частки успішно усунутих інцидентів до 94 %, що підтверджує підвищення ефективності технічних заходів та зростання безпекової зрілості команд (див. п. 4.5) [29].

Крім безпосереднього зниження ризику, зафіксовано зростання загальної обізнаності розробників. Зокрема, значно активізувалося застосування .gitignore-файлів, централізованих секрет-менеджерів та сервісів автоматичної ротації облікових даних. Це свідчить про формування практик безпечного розроблення у відповідності до принципів Secure Software Development Lifecycle (SSDLC), що розглядаються як необхідна умова мінімізації ризиків у довгостроковій перспективі [9].

4.2.Порівняння ситуації до і після впровадження централізованого зберігання секретів

На додаток до впровадження автоматизованих засобів виявлення, ефективним інструментом зменшення ризику витоку конфіденційної інформації стало централізоване зберігання секретів. Такий підхід передбачає винесення всіх облікових даних, токенів, ключів API, паролів та інших чутливих елементів за межі програмного коду до спеціалізованих сховищ з контрольованим доступом, журналюванням та підтримкою політик безпеки [32].

Після впровадження централізованих систем зберігання, зокрема таких як HashiCorp Vault, AWS Secrets Manager, Azure Key Vault або Google Secret Manager, ситуація змінюється докорінно. Основною зміною стає винесення секретів за межі коду, що унеможливлює їх статичне розміщення у репозиторії. Замість цього, застосунок або середовище CI/CD отримує тимчасовий доступ до секрету через контрольований API-запит. При цьому доступ може бути обмежений у часі, контексті виконання або призначений виключно конкретному сервісу чи користувачу [9].

Принципи централізованого зберігання базуються на кількох ключових підходах. По-перше, запроваджується детальне управління політиками доступу до секретів на основі ідентичностей, ролей або призначених контекстів. По-друге, забезпечується можливість аудиту всіх дій, пов’язаних з обробкою секретів, включно з доступом, змінами та відкликанням. По-третє, реалізуються механізми автоматичної ротації секретів через визначені інтервали або у відповідь на події безпеки. Нарешті, відбувається чітке розмежування секретів за середовищами (наприклад, development, staging, production), що дозволяє мінімізувати наслідки випадкового або несанкціонованого доступу.

Централізоване зберігання секретів не лише знижує ймовірність витоку чутливої інформації, але й забезпечує гнучкість, масштабованість та відповідність принципам Zero Trust. У підсумку, перехід

до такого підходу дозволяє інтегрувати управління конфіденційними даними у загальний процес забезпечення безпеки розробки програмного забезпечення, підвищуючи зрілість інформаційної безпеки на рівні організації.

Таблиця 10

Порівняння до та після впровадження централізованого зберігання

Критерій	**До впровадження**	**Після впровадження**
Місце зберігання секретів	У коді або конфіг-файлах	Централізоване сховище
Доступ до секретів	Безконтрольний	Через API з політиками
Аудит	Відсутній	Повний журнал доступу
Ротація	Переважно вручну або відсутня	Автоматизована
Розмежування середовищ	Часто ігнорується	Чітке та контрольоване

Узагальнено, перехід до централізованого керування секретами дозволив не лише знизити ризики, пов’язані з витоками, а й стандартизувати процеси обробки конфіденційних даних у командах розробки. Це сприяло зростанню зрілості безпеки в межах Secure Development Lifecycle (SDLC) та заклало основу для впровадження більш гнучких і безпечних DevSecOps-підходів [Таблиця 10].

4.3.Візуалізація результатів: динаміка виявлення та усунення секретів

Візуалізація [Рис. 1] ілюструє ефективність впровадження автоматизованих механізмів виявлення чутливої інформації у програмному коді. Для усіх категорій секретів спостерігається зменшення кількості витоків більш як на 85 %, що підтверджує доцільність інтеграції спеціалізованих сканерів (зокрема, GitLeaks, GitHub Secret Scanning та TruffleHog) у процеси безперервної інтеграції та доставки [Таблиця 11].

Найбільш виражене зменшення зафіксовано для Telegram API токенів (−87,1 %) та SMTP облікових даних (−88,4 %), що традиційно часто фігурували у конфігураційних файлах або залишалися у .env-файлах без належної фільтрації. Така динаміка свідчить про значне зменшення ризику несанкціонованого доступу до зовнішніх сервісів та поштових інфраструктур.

Крім технічного ефекту, значне зниження витоків у категоріях SSH-ключів та рядків підключення до баз даних вказує на підвищення рівня дисципліни у поводженні з конфіденційною інформацією та поступове впровадження безпечних практик DevSecOps. Також простежується позитивна зміна в культурі розробки: команди почали використовувати секрет-менеджери, pre-commit політики та сканери безпеки як обов’язкові етапи розробки.

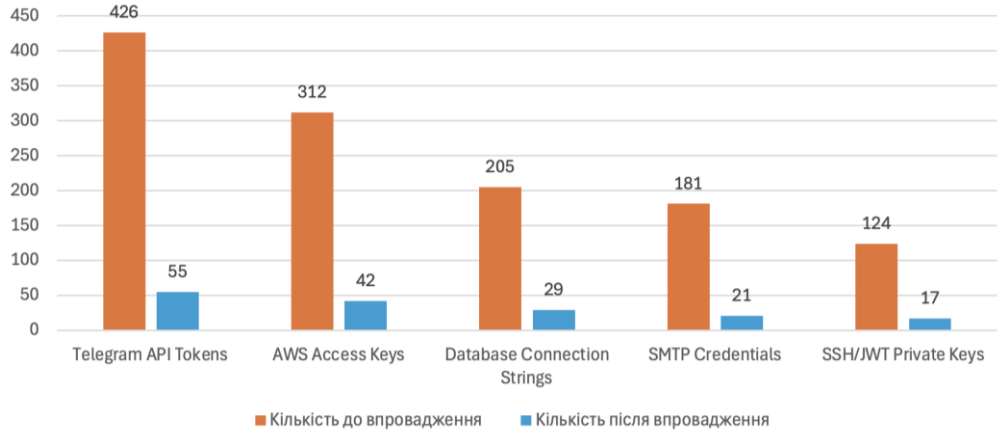

Рис. 1. Кількість витоків секретів до та після впровадження

Таблиця 11

Динаміка кількості інцидентів

Категорія секретів	**Кількість до впровадження**	**Кількість після впровадження**
Telegram API Tokens	426	55
AWS Access Keys	312	42
Database Connection Strings	205	29
SMTP Credentials	181	21
SSH/JWT Private Keys	124	17

Таким чином, впровадження комбінованого підходу, що поєднує автоматизоване виявлення, блокування підозрілих змін та наступну ротацію секретів, дозволяє суттєво зменшити кількість

інцидентів, а також імовірні наслідки компрометації. Цей підхід формує надійну основу для реалізації принципів *Secure Software Development Lifecycle (SSDLC)* на рівні процесів розробки.

4.4.Мітігація ключових ризиків, пов'язаних із неконтрольованим зберіганням секретів у програмному коді

Одним з найкритичніших наслідків зберігання облікових даних без належного контролю є несанкціонований доступ до інфраструктурних компонентів, включаючи CI/CD, хмарні сервіси та середовища виконання [Таблиця 12]. Для зниження такого ризику було впроваджено централізоване сховище (наприклад, HashiCorp Vault) з контролем доступу, шифруванням і можливістю автоматичної ротації [13]. Завдяки цьому вдалося виключити hardcoded секрети, скоротити кількість витоків на понад 85 % та стандартизувати доступ до конфіденційних даних.

Інтеграція секрет-менеджера у пайплайни CI/CD дозволила повністю ізолювати облікові дані від коду, а також заблокувати можливість випадкового або зловмисного їх використання. Особливу роль відіграли перевірки pull request-ів і commit hooks, які запобігали розгортанню коду у випадку виявлення чутливих рядків. Таким чином, було знижено ризик компрометації ланцюга поставки та забезпечено належну гігієну коду на ранніх етапах.

Ризик втрати контролю над конфігураціями був усунений за рахунок централізованого управління секретами з розділенням середовищ та можливістю ротації без ручного втручання. Це дозволило скоротити час реагування на інциденти з декількох діб до лічених хвилин [33].

У контексті відповідності нормативно-правовим вимогам (NIST SP 800-53, ISO/IEC 27001, SOC 2, OWASP), реалізовані заходи забезпечили ведення журналів доступу, контроль привілеїв та аудит дій користувачів, що є важливим фактором під час зовнішніх перевірок.

Автоматичне виявлення витоків через сторонні сервіси та публічні репозиторії було усунене через впровадження сканерів, таких як GitLeaks та GitHub Secret Scanning, з подальшим блокуванням небезпечних комітів. Цей підхід дозволив виявляти проблеми ще до того, як код з'являвся у публічному просторі.

Для мінімізації впливу людського фактора проведено серії тренінгів з безпечної розробки та впроваджено перевірки з безпековим фокусом на всіх етапах розробки, включаючи рев'ю коду, що дозволило знизити кількість інцидентів унаслідок недбалості.

Таблиця 12

Мітігація ключових ризиків, пов'язаних із секретами

№	Ризик	Механізм мітігації	Інструмент/Підхід	Очікуваний або досягнутий результат
1	Несанкціонований доступ до інфраструктурних ресурсів	Централізоване зберігання та контрольований доступ до секретів	HashiCorp Vault, політики ACL	>85 % зменшення кількості інцидентів; виключення hardcoded секретів
2	Компрометація ланцюга поставки ПЗ	Ізоляція секретів від коду та інтеграція в CI/CD	Vault + GitHub Actions + блокування PR	Мінімізація ризику втручання в реліз; блокування небезпечних змін
3	Втрата контролю над конфігураціями	Автоматична ротація, централізоване управління секретами	Vault, шаблони config-as-code	Зменшення часу реагування; уникнення дублювання даних
4	Невідповідність стандартам і регуляціям	Аудит подій, ведення журналів, політики доступу	Vault Audit Logs, відповідність NIST/ISO/SOC	Підвищення рівня довіри, успішне проходження аудитів
5	Автоматичний витік через індексацію або сторонні сервіси	Превентивне сканування коду перед комітом	GitLeaks, GitHub Secret Scanning, commit hooks	Запобігання публікації секретів у відкриті репозиторії
6	Внутрішні помилки та людський фактор	Освітні ініціативи + перевірки під час CI/CD + ручне рев'ю	DevSecOps training, manual security review, Git hooks	Підвищення обізнаності, виявлення проблем до релізу

Висновки

У межах проведеного дослідження було здійснено системний аналіз ризиків, пов'язаних із неконтрольованим зберіганням конфіденційної інформації у програмному коді. З урахуванням актуальних викликів для інформаційної безпеки, було сформульовано дослідницьку задачу, що

передбачала ідентифікацію критичних загроз, моделювання потенційних сценаріїв витоків, а також розробку технічних і організаційних механізмів їх мінімізації.

У процесі роботи проаналізовано ключові причини витоків чутливої інформації, включаючи недотримання політик безпеки, використання hardcoded-секретів та відсутність контрольованого середовища зберігання. З метою підтвердження гіпотез дослідження було проведено емпіричну оцінку ефективності сучасних підходів до управління секретами, таких як автоматизоване виявлення чутливих даних, централізоване сховище з API-доступом, автоматична ротація та аудит операцій.

Отримані результати засвідчили, що впровадження комплексної моделі управління секретами дозволяє досягти понад 85 % зниження інцидентів витоку ключових категорій даних, зокрема API-ключів, токенів доступу та конфігураційних облікових даних. Важливим аспектом виявилося поєднання технічних засобів із організаційними заходами, зокрема проведенням тренінгів, впровадженням чеклістів та розширенням контролю на етапах розробки та деплойменту. Це вказує не лише на ефективність обраного підходу, але й на його здатність формувати зрілі практики безпечної розробки в командах.

Таким чином, поставлена на початку дослідження мета — виявлення та системне зниження ризиків витоку конфіденційної інформації через програмний код — була досягнута шляхом реалізації багаторівневого підходу до управління секретами. Запропонована модель забезпечує не лише технічну ефективність, але й відповідність сучасним вимогам до DevSecOps-інфраструктури, що є ключовою умовою забезпечення інформаційної стійкості у високодинамічному середовищі розробки програмного забезпечення.

Крім того, результати дослідження демонструють тісний взаємозв’язок між рівнем зрілості процесів SDLC та ймовірністю витоку секретів. Показано, що системне впровадження політик безпеки, інтеграція інструментів контролю у пайплайни CI/CD, а також автоматизація ротації та моніторингу секретів значно знижують навантаження на команди безпеки та сприяють зменшенню технічного боргу.

Запропонований підхід може бути адаптований до різних масштабів організацій та типів IT-інфраструктур, включаючи мультихмарні та мікросервісні середовища. З огляду на швидкі темпи розвитку цифрових сервісів, особливої актуальності набуває інтеграція таких рішень у рамках Zero Trust-архітектури та платформенного DevSecOps, що відкриває перспективи для подальших досліджень у сфері автоматизованої безпеки та політик самовідновлення (self-healing security posture).

Список використаної літератури

1. Rahman, Md Nafiu & Ahmed, Sadif & Wahab, Zahin & Sohan, S & Shahriyar, Rifat. (2025). Secret Breach Detection in Source Code with Large Language Models. 10.48550/arXiv.2504.18784.
2. Basak, Setu Kumar & Pardeshi, Tanmay & Reaves, Bradley & Williams, Laurie. (2025). RiskHarvester: A Risk-based Tool to Prioritize Secret Removal Efforts in Software Artifacts. 10.48550/arXiv.2502.01020.
3. Tabassum, S & Swaroop, K & Babu, K & Babu, N & Anas, S. (2025). A Data Sharing Protocol to Minimize Security and Privacy Risks of Cloud Storage in Big Data Era. International Research Journal on Advanced Engineering Hub (IRJAEH). 3. 1604-1609. 10.47392/IRJAEH.2025.0228.
4. Dreis, Yurii. (2025). IMPROVED METHOD OF ASSESSING DAMAGE TO THE NATIONAL SECURITY OF UKRAINE IN CASE OF LEAKAGE OF STATE SECRETS. Cybersecurity Education Science Technique. 3. 10.28925/2663-4023.2025.27.771.
5. Yi, Jin & Huang, Weijie & Huang, Lianghao & Huang, Guozheng & Zheng, Guangyong & Li, Yongle. (2025). Decentralized storage technology of network information security level secret key of distribution automation terminal. Discover Applied Sciences. 7. 10.1007/s42452-025-07124-9.
6. Kurihara, Itaru & Kurihara, Jun & Tanaka, Toshiaki. (2024). A New Security Measure in Secret Sharing Schemes and Secure Network Coding. IEEE Access. PP. 1-1. 10.1109/ACCESS.2024.3401471.
7. Wang, Rui & Li, Longlong & Yang, Guozheng & Yan, Xuehu & Yan, Wei. (2024). Secret Cracking and Security Enhancement for the Image Application of CRT-Based Secret Sharing. IEEE Transactions on Information Forensics and Security. PP. 1-1. 10.1109/TIFS.2024.3477265.
8. Martseniuk, Y., Partyka, A., Harasymchuk, O., Shevchenko, S. Universal centralized secret data management for automated public cloud provisioning (2024) CEUR Workshop Proceedings, 3826, pp. 72-81.
9. Martseniuk, Y., Partyka, A., Harasymchuk, O., Nyemkova, E., Karpinski, M. Shadow IT risk analysis in public cloud infrastructure (2024) CEUR Workshop Proceedings, 3800, pp. 22-31.
10. Zhang, Fan & Xu, Jian & Yang, Gangqiang. (2023). Design of Regenerating Code Based on Security Level in Cloud Storage System. Electronics. 12. 2423. 10.3390/electronics12112423.
11. Giretti, Anthony. (2023). Managing Application Secrets. 10.1007/978-1-4842-9979-1_9.

12. Pai, Santosh & Kunte, Srinivasa. (2023). Secret Management in Managed Kubernetes Services. International Journal of Case Studies in Business, IT, and Education. 130-140. 10.47992/IJCSBE.2581.6942.0263.

13. Somasundaram, Prakash. (2024). Unified Secret Management Across Cloud Platforms: a Strategy for Secure Credential Storage and Access. INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY. 15. 5-12.

14. Akinbolaji, Taiwo & Nzeako, Godwin & Akokodaripon, David & Aderoju, Akorede. (2024). Proactive monitoring and security in cloud infrastructure: leveraging tools like Prometheus, Grafana, and HashiCorp Vault for Robust DevOps Practices. World Journal of Advanced Engineering Technology and Sciences. 13. 074–089. 10.30574/wjaets.2024.13.2.0543.

15. Raghu, Aravind. (2025). Implementing HashiCorp Vault for Secure Credential Management in Financial Services: A Java-Centric Approach. International Journal of Computational and Experimental Science and Engineering. 11. 10.22399/ijcesen.2473.

16. Dama, Hari & III, Researcher. (2025). Secure Credential Management in Cloud Databases Using Azure Key Vault Integration. INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY. 16. 163-176. 10.34218/IJCET_16_03_013.

17. Palanisamy, Pradeepkumar & Scholar, Reseacher. (2023). Secure Credential Handling for Test Automation: A Deep Dive into Vault and Cloud-Native Secrets Managers. INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY. 14. 121-147. 10.34218/IJCET_14_01_013.

18. Harrington, Cameron. (2022). The eternal return: Imagining security futures at the Doomsday Vault. Environment and Planning E: Nature and Space. 6. 251484862211453. 10.1177/25148486221145365.

19. Almeida, Luis & Fernandez, Brayan & Zambrano, Daliana & Almachi, Anthony & Pillajo, Hilton & Yoo, Sang Guun. (2024). One-Time Passwords: A Literary Review of Different Protocols and Their Applications. 205-219. 10.1007/978-3-031-48855-9_16.

20. Tkach, Volodymyr & Shemendiuk, Oleksandr & Cherednychenko, Oleksiy. (2024). RESEARCH ON ISSUES OF INFORMATION SECURITY RISKS ASSESSMENT AND MANAGEMENT IN THE SECURITY AND DEFENSE SECTOR AND FORMATION OF SECURITY LEVEL INDICATORS. Cybersecurity: Education, Science, Technique. 2. 81-94. 10.28925/2663-4023.2024.26.636.

21. Nash, Aleck & Choo, Kim-Kwang. (2024). Password Managers and Vault Application Security and Forensics: Research Challenges and Future Opportunities. 31-53. 10.1007/978-3-031-56583-0_3.

22. Wen, Elliott & Wang, Jia & Dietrich, Jens. (2022). SecretHunter: A Large-scale Secret Scanner for Public Git Repositories. 123-130. 10.1109/TrustCom56396.2022.00028.

23. Saha, Aakanksha & Denning, Tamara & Srikumar, Vivek & Kasera, Sneha. (2020). Secrets in Source Code: Reducing False Positives using Machine Learning. 168-175. 10.1109/COMSNETS48256.2020.9027350.

24. Lykousas, Nikolaos & Patsakis, Constantinos. (2023). Tales from the Git: Automating the detection of secrets on code and assessing developers' passwords choices. 68-75. 10.1109/EuroSPW59978.2023.00013.

25. Martseniuk, Y., Partyka, A., Harasymchuk, O., Cherevyk V. and Dovzhenko N. Research of the Centralized Configuration Repository Efficiency for Secure Cloud Service Infrastructure Management (2025) CEUR Workshop Proceedings, 3991, pp. 260-274.

26. Feng, Runhan & Yan, Ziyang & Peng, Shiyan & Zhang, Yuanyuan. (2022). Automated detection of password leakage from public GitHub repositories. 175-186. 10.1145/3510003.3510150.

27. Meşecan, İbrahim & Blackwell, Daniel & Clark, David & Cohen, Myra & Petke, Justyna. (2022). Keeping Secrets: Multi-objective Genetic Improvement for Detecting and Reducing Information Leakage. 10.1145/3551349.3556947.

28. Ramaswamy, Mithilesh. (2024). Early Detection of Hard-Coded Secrets in Software Development: A Multi-Method Approach Integrating Static Analysis, Entropy-Based Detection, and Machine Learning. International Scientific Journal of Engineering and Management. 03. 1-6. 10.55041/ISJEM0411.

29. Segura, Thomas. (2023). The nightmare of hard-coded credentials. Network Security. 2023. 10.12968/S1353-4858(23)70010-X.

30. Basak, Setu Kumar & Cox, Jamison & Reaves, Bradley & Williams, Laurie. (2023). A Comparative Study of Software Secrets Reporting by Secret Detection Tools. 10.1109/ESEM56168.2023.10304853.

31. Li, Kevin & Ling, Lin & Yang, Jinqiu & Wei, Lili. (2024). Automatically Detecting Checked-In Secrets in Android Apps: How Far Are We?. 10.48550/arXiv.2412.10922.

32. Yi, Jin & Huang, Weijie & Huang, Lianghao & Huang, Guozheng & Zheng, Guangyong & Li, Yongle. (2025). Decentralized storage technology of network information security level secret key of distribution automation terminal. Discover Applied Sciences. 7. 10.1007/s42452-025-07124-9.

33. Pan, Ziyue & Shen, Wenbo & Wang, Xingkai & Yang, Yutian & Chang, Rui & Liu, Yao & Liu, Chengwei & Liu, Yang & Ren, Kui. (2023). Ambush From All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines. IEEE Transactions on Dependable and Secure Computing. PP. 1-16. 10.1109/TDSC.2023.3253572.