Information security risks associated with the uncontrolled storage of secrets in source code*

Vladyslav Chornii[1,†], Yevhenii Martseniuk[1,†], Andrii Partyka[1,†] and Oleh Harasymchuk[1,*,†]

[1] Lviv Polytechnic National University, 12 Stepana Bandery str., 79000 Lviv, Ukraine

Abstract

This paper explores the problem of information security violations caused by the uncontrolled storage of sensitive data in software source code. Specifically, it analyzes cases involving the inclusion of API keys, access tokens, database passwords, cloud platform credentials, and other confidential data in public or private repositories, which pose significant threats to the integrity of IT infrastructures. The research involves an empirical analysis of open-source repositories on GitHub using specialized secret detection tools such as TruffleHog, GitLeaks, and detect-secrets. The collected data were classified according to the types of discovered objects, allowing for the identification of the most common risk vectors. Special emphasis is placed on infrastructure-related threats that arise from storing secrets in CI/CD pipelines, configuration files, and automated deployment scripts. The study proposes approaches for integrating secret management into the software development lifecycle (SDLC), including automated scanning, centralized credential management, and the implementation of role-based access policies. Furthermore, a risk assessment based on the NIST SP 800-30 methodology was conducted to demonstrate the severity of potential secret leaks and to substantiate the need for secure DevSecOps processes. The paper presents practical recommendations for risk mitigation, including key rotation, personnel training, enforcement of role-based access control mechanisms, and periodic audits of development environments.

Keywords

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

1. Introduction

One of the most pressing security concerns in the context of digital transformation, development automation, and cloud adoption is the uncontrolled storage of sensitive information within software source code. This widespread and hazardous practice includes the hardcoding of credentials such as database passwords, API keys, access tokens, private SSH keys, and confidential configuration parameters, often present in both public and internal repositories.

Such insecure storage practices pose not only a risk of data leakage but also the potential compromise of critical IT infrastructure, ranging from unauthorized access to cloud environments and containerized services to disruptions in CI/CD pipelines, failures in internal API operations, and the uncontrolled distribution of malicious code [1–3]. These risks are further exacerbated in multi-cloud and microservice architectures, where the number of interactions and access points increases exponentially.

Despite the availability of modern secret management solutions—such as HashiCorp Vault, AWS Secrets Manager, and Kubernetes Secrets—their adoption in CI/CD workflows often remains partial, with poorly formalized policies for secret storage and rotation. Therefore, it is essential to analyze both infrastructure-level vulnerabilities and the effectiveness of current methods for identifying and mitigating the risks associated with credential leakage [4, 5].

The objective of this research is to identify typical infrastructure-related risks arising from uncontrolled storage of secrets in code, to evaluate the effectiveness of detection and management

* CSDP'2025: Cyber Security and Data Protection, July 31, 2025, Lviv, Ukraine
* Corresponding author.
† These authors contributed equally.
vladyslav.chornii.kb.2022@lpnu.ua (V. Chornii); yevhenii.v.martseniuk@lpnu.ua (Y. Martseniuk); andrijp14@gmail.com (A. Partyka); oleh.i.harasymchuk@lpnu.ua (O. Harasymchuk)
0009-0006-8671-3889 (V. Chornii); 0009-0009-2289-0968 (Y. Martseniuk); 0000-0003-3037-8373 (A. Partyka); 0000-0002-8742-8872 (O. Harasymchuk)

tools, and to develop recommendations for integrating secure practices into the software development and deployment lifecycle.

This study aims to detect, systematize, and analyze information security risks caused by hardcoded secrets in source code. It seeks to substantiate the implementation of technical and organizational controls to detect, remediate, and prevent credential leakage—including access keys and confidential parameters—in modern IT infrastructures, particularly within cloud environments and automated CI/CD workflows.

To achieve this goal, the following key tasks are addressed:

- Review and classify common categories of secrets found in source code, with consideration of their impact on information security.
- Analyze frequent leakage scenarios in public and private repositories, with an emphasis on infrastructure vulnerabilities related to cloud and CI/CD-based development.
- Investigate contemporary tools for automated secret detection, as well as secret management systems such as HashiCorp Vault, AWS Secrets Manager, Google Secret Manager, and others.
- design a conceptual model for integrating secret detection, storage, and rotation processes into the SDLC, considering the security requirements of multi-cloud and containerized environments.
- Formulate practical recommendations for implementing secure secret management practices in enterprise IT infrastructures, aligned with DevSecOps principles and international information security standards.

Literature review

The problem of storing sensitive information in source code has gained increased relevance in the era of widespread adoption of cloud technologies, DevOps practices, and continuous integration/deployment (CI/CD). Academic sources highlight that hardcoded credentials often lead to unauthorized access to critical infrastructure components, including cloud services, databases, and external APIs [6].

A study on the development of the Risk Harvester tool [6] emphasizes the feasibility of applying risk-oriented prioritization in secret removal, allowing security teams to focus on the most vulnerable code segments. Similarly, [7] proposes automatic secret detection through source code analysis models, although the focus is more aligned with artificial intelligence than infrastructure-oriented secret management.

Centralized secret management tools—including HashiCorp Vault, AWS Secrets Manager, and Kubernetes Secrets—are actively explored in the context of secure credential access in microservice and multi-cloud environments. Notably, study [8] demonstrates the implementation of Vault within a Kubernetes cluster for dynamic secret issuance to microservices, which reduces the attack surface and enables controlled access to sensitive data. Further developments in this domain are discussed in [9] and [10], with an emphasis on unified secret management across multi-cloud infrastructures and cross-layer integration with DevOps workflows.

Special attention is given to the security of automated testing and deployment processes, where exposed access keys remain a prevalent attack vector [11]. Researchers underscore the importance of integrating automated detection, rotation, and revocation of secrets into CI/CD as a critical element of the DevSecOps approach.

Several publications [8, 9, 11] also emphasize the need for methodological frameworks for risk assessment related to secret leakage, both at the state and corporate levels. This highlights the interdisciplinary nature of the problem, encompassing not only technical aspects but also risk management, compliance, and security policy development.

In addition to centralized secret management solutions, emerging research highlights the use of blockchain technologies to enhance the confidentiality, integrity, and auditability of sensitive

information, supporting compliance with regulations such as GDPR [10]. Martseniuk et al. [12] propose a universal approach for centralized secret data management in automated public cloud provisioning, demonstrating reduced operational risks and improved control over dynamic environments. Complementary frameworks for information classification and policy enforcement, such as SOC 2 Type II-based models, have been shown to provide structured guidelines for secure handling of secrets throughout the software development lifecycle [13, 14]. Furthermore, rule-based intelligent systems and secure authentication mechanisms contribute to the proactive management of sensitive data and reduction of human-related errors in DevOps and CI/CD pipelines [15, 16]. These interdisciplinary approaches underline the necessity of integrating technical, organizational, and regulatory measures to ensure comprehensive protection of secrets in source code [17].

In conclusion, the analysis of current academic literature indicates that uncontrolled storage of secrets in code is a complex problem requiring both technical and organizational solutions. Despite the availability of tools and best practices, there remains a lack of standardized approaches for integrating secret management systems into real-world infrastructure scenarios. This underscores the need for further research focused on modeling common vulnerabilities, improving detection tools, and deploying secure secret management workflows across the full software development lifecycle.

1.1. Challenges of DevOps in secret management

In the context of modern information systems—characterized by a high level of automation, widespread use of microservice architecture, and active adoption of cloud infrastructure—the practice of storing sensitive information directly in source code remains common. However, this approach introduces substantial information security risks to organizations. Such confidential data includes passwords, API keys, access tokens, database credentials, and configuration parameters required for integration with external services. When stored directly within the codebase, especially in public or insufficiently protected repositories, these secrets significantly increase the risk of unauthorized access to critical components of the IT infrastructure [7].

There exists an inherent conflict between the demand for speed, automation, and openness in development processes—enabled by tools such as CI/CD, GitOps, and Infrastructure as Code—and the necessity of maintaining an adequate security posture. This issue is particularly acute for small and medium-sized organizations, where security questions are often deprioritized or fall outside the scope of operational control. The absence of formalized secret management policies, the low level of security awareness among developers, and the limited adoption of centralized credential management solutions (such as HashiCorp Vault or AWS Secrets Manager) collectively increase the likelihood of data leakage and cloud infrastructure compromise.

Despite the availability of standalone technical solutions, a comprehensive approach to integrating secret management into SDLC is still lacking. Moreover, security frameworks implemented within DevOps environments often fail to account for the need to prioritize risks associated with secret exposure. This underscores the necessity of further academic inquiry into the subject, including the classification of typical vulnerabilities, the formalization of threat models, and the development of monitoring, detection, and mitigation policies for confidential data leakage at the infrastructure level [6].

1.2. Information security risks associated with uncontrolled storage of secrets in source code

The storage of sensitive information—such as access tokens, passwords, private keys, API keys, and configuration parameters—within open or insufficiently protected source code introduces a set of critical information security risks for an organization's IT infrastructure. These risks can be classified into several key domains.

Risk of unauthorized access to infrastructure resources. If credentials embedded in source code are leaked, attackers may directly access critical services within the organization. This includes cloud accounts (AWS, Azure, GCP), unsecured databases, and CI/CD servers that could be leveraged for privilege escalation or injection of malicious code into production environments. Such attacks often go undetected at the execution stage, since the use of valid credentials typically does not trigger anomalies in event logs without proper monitoring in place [8].

Risk of software supply chain compromise. The presence of secrets in repositories accessible to external contributors increases the likelihood of development tooling being compromised. Once an attacker gains access to the CI/CD pipeline, they can introduce malicious or modified code into the software that is ultimately delivered to end-users. This attack vector is especially dangerous in DevOps paradigms, where automated releases reduce the opportunity for manual verification at each stage.

Risk of configuration and environment control loss. Hardcoded secrets are frequently duplicated across development, staging, and production environments, making it difficult to maintain their relevance and impeding centralized rotation. In the event of a leak, organizations are forced to conduct full-scale audits and infrastructure revalidation, including the recreation of accounts, key regeneration, and manual configuration updates—a time-consuming and error-prone process [8].

Risk of non-compliance with standards and regulatory frameworks. Storing credentials in open-source code violates several international security standards, including ISO/IEC 27001, NIST SP 800-53, SOC 2, and OWASP ASVS. Identifying such violations during an audit may lead to financial penalties, loss of certification, or reputational damage among clients and partners.

Risk of automated leakage through indexing and third-party services. Secrets placed in publicly accessible or insufficiently protected repositories can be rapidly indexed by search engines or discovered by specialized bots and scanners (e.g., GitRob, TruffleHog, Shhgit). Consequently, data exposure may occur within minutes of code publication, even without a targeted attack [9].

Risk of internal breaches and human error. A significant portion of incidents involving secrets in code stems from human mistakes: accidental commits of confidential data, reuse of tokens across personal and professional projects, and so on. The absence or neglect of secure development policies contributes to irreversible leaks that are difficult to detect and trace without proper monitoring procedures [11].

In conclusion, the uncontrolled storage of secrets in source code should be recognized as a systemic vulnerability that threatens the confidentiality, integrity, and availability of organizational information infrastructure. Effectively mitigating these risks requires a comprehensive approach that combines technical, organizational, and procedural security measures throughout the software development lifecycle.

1.3. Lack of standardized practices and controls within the SDLC framework

One of the critical challenges in modern software development is the insufficient integration of security practices across all phases of SDLC. Within the widespread adoption of DevOps methodologies, development teams prioritize rapid delivery, continuous integration, and automation of software deployment. This often leads to neglect of information security considerations, which should be incorporated through DevSecOps principles.

The absence of standardized security controls at each stage of the SDLC contributes to the accumulation of so-called security debt—unresolved or underestimated vulnerabilities that are overlooked during accelerated development and release cycles. Over time, this complicates infrastructure maintenance and evolution, as remediating such vulnerabilities may require significant resource investment, process adjustments, or even codebase refactoring [10].

Within the context of the SDLC, this issue manifests at every development stage, resulting in systemic vulnerabilities and escalating information security risks due to the lack of embedded security practices (see Table 1).

Table 1
Security gaps in DevOps across SDLC phases and their potential consequences

SDLC Phase	Security Deficiency in DevOps	Potential Consequences
Requirements Analysis	Security requirements are not documented or are ignored	Insufficient baseline protection; no planning for secure secrets management
Design	Threat models are not defined; secure storage channels are not designed	No architectural space allocated for secret management systems
Development	Lack of linters, code scanners, or pre-commit hooks for detecting secrets	Introduction of hardcoded secrets and plaintext credentials in source code
Testing	Security aspects are not covered; secret handling is not tested	Undetected leaks of confidential information
Release & Deployment	Absence of security controls in CI/CD (e.g., secret scanning, RBAC enforcement)	Secrets reaching production environments; insecure deployment configurations
Operations & Maintenance	No security monitoring or regular audit controls for secrets	Accumulation of untracked secrets in repositories and runtime environments

Thus, the lack of DevSecOps implementation as a mandatory component of SDLC, as outlined in the previous sections, results in a persistent technical debt in the field of information security. Within the SDLC, this is manifested through a systemic absence of security validations and controls at all critical stages—from requirements analysis and design to development, testing, release, and maintenance. Consequently, vulnerabilities associated with the uncontrolled storage of sensitive information remain undetected, accumulate with each development and integration iteration, and complicate further operation of the software product [18].

According to the SDLC structure, the absence of security at early stages leads to a failure in addressing secret protection requirements in system architecture, thereby hindering centralized management at later stages [7]. During development, the lack of pre-commit checks and static code analysis facilitates the proliferation of hardcoded credentials, while weak controls at the CI/CD process level create risks of such secrets being deployed into production environments. At the maintenance stage, the absence of monitoring and regular auditing exacerbates these vulnerabilities, making the organization susceptible to targeted attacks or accidental leaks.

2. Risk assessment of uncontrolled secret storage in source code

One of the critical threats to information security in modern information and communication systems is the uncontrolled storage of sensitive data within source code. Despite its prevalence among developers, this practice creates preconditions for significant risks, potentially leading to infrastructure compromise, data integrity breaches, leakage of confidential information, and non-compliance with regulatory standards [19].

Given the importance of a comprehensive approach to information security risk management, this section presents a formalized risk assessment related to secret storage in codebases. The methodology applied is NIST SP 800-30 “Guide for Conducting Risk Assessments,” which enables systematic identification of assets, relevant threats and vulnerabilities, evaluation of their likelihood of occurrence, and estimation of the potential organizational impact [20].

The assessment considers the specifics of development environments, CI/CD processes, cloud infrastructure, and DevOps operational practices, which are especially susceptible to issues arising from uncontrolled handling of secrets. This approach enables risk prioritization and the development of mitigation recommendations through the integration of appropriate controls into SDLC.

According to this methodology, the assessment is carried out through the sequential identification of the following components:

1. **Asset**—a resource or component of the information infrastructure that requires protection and may be affected by a threat.
2. **Threat**—a potential event or action that may cause undesirable consequences for an asset.
3. **Vulnerability**—a weakness in the system, process, or control that allows a threat to be successfully exploited.
4. **Likelihood**—an expert judgment on the probability that a given threat will exploit an existing vulnerability.
5. **Impact**—the potential scale of losses or adverse effects on the organization if the threat materializes.
6. **Risk Level**—an integrated value resulting from the combination of threat likelihood and the expected impact.

This model enables a comprehensive risk assessment through the structured analysis of each of these components [21]. In the context of this research, it is applied to evaluate the degree of threats posed by storing credentials, tokens, keys, and other secrets without proper control in source code, repositories, and infrastructure components in Table 2.

Table 2
Risk assessment of uncontrolled storage of sensitive information in source code according to NIST SP 800-30

No	Asset	Threat	Vulnerability	Likelihood	Impact	Risk Level
1	Cloud accounts	Unauthorized access	Storage of access keys (AWS, Azure, GCP) in plaintext source code	High	Full control over cloud infrastructure	Critical
2	CI/CD pipelines	Supply chain compromise	API keys for Jenkins/GitHub Actions stored in repositories	Medium	Injection of malicious code into production	High
3	Databases	Confidential data leakage	Hardcoded credentials for PostgreSQL, MySQL, MongoDB	High	Compromise of personal and financial information	Critical
4	DevOps infrastructure	Service disruption	Compromised tokens for infrastructure APIs (Docker, Ansible, Helm)	Medium	Deployment and scaling failures	High
5	Internal configuration files	Unauthorized network scanning	Hardcoded IP addresses, ports, and network credentials	Medium	Network reconnaissance and attack preparation	Medium
6	Regulatory	Non-	Absence of policies and	Medium	Fines, loss of	High

	compliance	compliance with standards	controls for secret management		certification, reputational damage	
7	Organizational reputation	Public leakage of secrets via GitHub	Accidental publication of code with keys or passwords	High	Loss of customer trust, media exposure	High
8	Business continuity	External control over services	Lack of automated secret rotation	Low	Temporary service disruption, recovery efforts needed	Medium

2.1. Critical risks associated with unauthorized access to core organizational resources

The realization of these risks can lead to catastrophic consequences, including full compromise of the environment, leakage of confidential information, or complete disruption of business processes.

To mitigate such risks, it is essential to implement a comprehensive set of technical and organizational measures. In particular, it is recommended to conduct automated scanning of source code for the presence of sensitive data using appropriate tools such as TruffleHog, GitLeaks, and detect-secrets. An important step is the deployment of centralized secret management systems, with the most widely used solutions being HashiCorp Vault and AWS Secrets Manager [22].

Furthermore, development processes should incorporate policies that prohibit hardcoding of secrets into source code, including the use of Git hooks, pre-commit checks, and verification tools integrated into CI/CD pipelines. In the event of secret compromise, immediate revocation and rotation must be ensured in accordance with established incident response procedures [23].

A typical example involves the leakage of an AWS access key exposed in a public repository. Such a key allows a malicious actor not only to view resources but also to create, modify, or delete them (e.g., EC2 instances, S3 buckets, or IAM roles), thereby putting the entire organizational infrastructure at risk.

2.2. High risks and their impact on organizational infrastructure

High risks arise in scenarios where vulnerabilities are not direct but may have significant systemic consequences. These include disruptions in CI/CD processes, compromise of the software supply chain, loss of control over certain services or configurations, and non-compliance with international information security standards. In most cases, such risks do not result in immediate compromise of the infrastructure but can lead to large-scale threats if ignored or accumulated over time [24]. To reduce the level of such risks, it is necessary to regulate the integration of secrets into CI/CD processes properly. All environment variables and credentials must be connected exclusively through centralized secret management systems, avoiding hardcoding in scripts or configuration files. The implementation of the principle of least privilege (RBAC) is mandatory for each component involved in deployment, testing, or integration processes, as it limits the potential for misuse of access rights [25]. In addition, periodic rotation of credentials should be ensured, accompanied by automation of configuration dependency updates in relevant services. It is also important to conduct regular audits for compliance with industry standards such as OWASP, ISO/IEC 27001, or SOC 2 to avoid regulatory violations, fines, or loss of certification. A typical example of such a risk realization is the case where an access key to GitHub Actions or Jenkins Pipeline is stored in open-source code. If such a key is leaked, a malicious actor can alter or replace the logic of the release process, exposing end users to malicious or compromised update —a typical scenario of a supply chain attack [7].

2.3. Medium risks and their exploitation potential

Medium-level information security risks typically do not pose an immediate threat to infrastructure but create favorable conditions for the accumulation of vulnerabilities that can eventually be exploited by malicious actors. Their realization is often the result of a combination of factors: organizational deficiencies, human error, and the absence of fundamental control mechanisms. In such cases, even minor configuration flaws or missing checks may lead to data leakage or integrity compromise.

To reduce the level of medium risks, it is advisable to implement regular internal audits targeting repositories, environment configurations, environment variables, and configuration files. These audits should be conducted on a scheduled basis, with recorded outcomes and corrective action plans. An important preventive measure is to conduct targeted developer training on secure storage and handling of secrets. Such training can be based on OWASP recommendations, Secure Coding principles, and the organization's internal security standards. Establishing a security-aware culture during early development phases helps prevent errors that result in the accumulation of vulnerabilities.

Furthermore, access segregation between environments with different trust levels must be enforced using role-based access control (RBAC). Specifically, access to staging or test environments should be clearly separated from production environments to prevent accidental promotion of insecure or test configurations [26].

Monitoring changes in configurations and environment variables enables timely detection of unauthorized actions or anomalies. Implementing alert systems for changes in key parameters provides an additional layer of protection against unintentional data leaks.

A typical example of a medium-risk scenario involves a developer leaving an API key in a configuration file intended for a test environment. In the absence of automated checks and validation processes, this key may accidentally be deployed to production and subsequently exploited by a threat actor [27]. While the threat is not immediate, it can escalate and serve as an entry point for more advanced attacks.

Risk assessment results indicate that ensuring IT infrastructure protection against secret leakage cannot rely solely on technical controls. Critical and high risks related to direct access to cloud environments, CI/CD processes, and confidential services require a comprehensive approach that combines both technical and procedural mechanisms. One of the key principles is the integration of security across all stages of SDLC—from design and implementation to maintenance.

This means that the control of secret storage and usage must be embedded into the processes of coding, testing, reviewing, automated deployment, and service updates. Critical components include centralized credential management, automated secret detection in code, key rotation policies, and regular configuration audits [28].

At the same time, medium-level risks—although not inherently critical—require ongoing hygiene: maintaining proper access structures, updating dependencies, validating environments, and responding promptly to configuration changes. However, the human factor remains paramount in this context: carelessness or lack of awareness of secure coding principles often becomes the root cause of even critical incidents.

Therefore, raising staff awareness should be a top priority. This includes regular training for developers, engineers, and administrators on secure handling of secrets and implementation of internal practices based on OWASP, NIST, and other authoritative standards [29].

In summary, effective management of secret leakage risks is not solely a matter of technology. Rather, it depends on well-structured processes, accountability at all levels, and continuous attention to security as an essential component of digital development.

2.4. Analysis of public GitHub repositories using secret detection tools

To identify common practices of storing confidential information in public source code, an empirical study was conducted on open-source repositories hosted on the GitHub platform. The

selected sample included publicly accessible projects containing infrastructure components such as Dockerfiles, CI/CD automation scripts (GitHub Actions, Jenkins), configuration files (e.g., .env, settings.py, config.json), and source code files containing environment variables or connection settings for third-party services.

A set of modern secret scanning tools was used to automate the detection of potentially sensitive information in source code:

- TruffleHog—a tool for deep analysis of Git commit history aimed at detecting patterns that match private keys, AWS, Azure, Slack, and Stripe tokens, as well as personally identifiable information (PII).
- GitLeaks—a tool for rapid scanning of files and commits using regular expressions and predefined patterns to detect the most common types of secrets [30].
- detect-secrets (developed by Yelp)—a module that integrates with Git hooks to identify secrets prior to commit execution, enabling the modeling of preventive control measures.

During the analysis, the type, source, and depth of discovered secrets were logged. Additionally, the presence or absence of secret rotation and revocation mechanisms was recorded, where such information was available in the accompanying documentation or identified through code-based patterns [31].

2.5. Classification of discovered secrets

During the analysis of public source code repositories, a significant number of cases of uncontrolled storage of sensitive information were identified. To systematize the obtained results, all discovered credentials were classified by data type, functional purpose, and the potential consequences of their exposure. This approach not only structures the identified threats but also enables a qualitative assessment of their impact on information security infrastructure.

One of the most critical categories identified was access keys to cloud platforms—particularly AWS, Google Cloud Platform, and Microsoft Azure. These types of data grant permissions to perform administrative operations within the cloud environment, including creating, modifying, and deleting resources. The compromise of such keys effectively results in the loss of control over cloud infrastructure, potentially leading to complete service disruption or destruction [32].

Another prevalent category consisted of API keys for third-party services—such as Stripe, Twilio, SendGrid, Slack, and Firebase. These credentials enable integration with payment gateways, communication platforms, and notification databases. Their exposure creates risks of unauthorized financial transactions, spam distribution, or leakage of users' confidential information.

Database credentials—especially for PostgreSQL, MySQL, and MongoDB—proved particularly dangerous. Typically stored in configuration files such as .env or settings files, the compromise of such credentials enables unauthorized modification, deletion, or duplication of sensitive data, including personal and financial information [33].

A separate category included SSH keys and other private cryptographic keys, stored in plain text formats such as PEM or RSA. Exposure of these keys poses a threat of direct remote access to servers without authentication, critically impacting server infrastructure security.

Authorization tokens, including JSON Web Tokens (JWT) and OAuth tokens, were also identified. Theft of such credentials enables an attacker to execute requests on behalf of a user or service, thus bypassing standard authentication mechanisms and gaining access to protected APIs and resources.

Additionally, the practice of hardcoded credentials directly in source code was documented—for example, variables such as username = "admin" or password = "123456". This indicates a low level of security culture among developers and poses serious risks even in the absence of access to configuration files [34].

Overall, the classification of discovered secrets by data type provided deeper insight into the nature of potential threats and their possible impact on the information infrastructure. Such an approach is essential for developing a comprehensive secret management policy within the software development lifecycle.

2.6. Causes of risk prevalence and their relation to the maturity of secure development processes

One of the key factors contributing to the systematic inclusion of sensitive information in source code is the insufficient maturity of secure software development processes implemented within SDLC. The conducted study identified three main factors that facilitate the prevalence of this risk:

1. Insufficient awareness of developers regarding information security issues.
2. Lack of formally regulated secret management policies.
3. Inadequate availability of technical tools integrated into development and DevOps processes.

Low personnel awareness frequently leads to situations where API keys, access tokens, passwords, or other credentials remain in the codebase—not due to malicious intent, but as a result of ignorance or underestimation of potential consequences. This is typical for small development teams, startups, or organizations lacking formal supervision from information security professionals or a culture of secure development [35].

Additionally, the absence of formalized secret management policies creates conditions in which developers must independently decide how to store sensitive data. This leads to the adoption of individual, uncoordinated practices that often fail to ensure an adequate level of protection, thereby increasing the likelihood of data leakage.

Another decisive factor is the insufficient availability of technical capabilities. In the absence of deployed solutions for centralized secret storage and management (such as HashiCorp Vault or AWS Secrets Manager), as well as control mechanisms (e.g., secret scanners or Git hooks), secrets are often stored in the code temporarily. However, without subsequent verification, these credentials may remain in repositories, posing a persistent risk of compromise [32].

The correlation between the identified factors and the maturity of SDLC processes can be represented in a comparative table that illustrates the evolution of security practices depending on the degree of formalization in software development processes (see Table 3).

Table 3
Correlation between secret leakage risks and the maturity level of Secure Development Lifecycle

SDLC Maturity Level	Key Characteristics	Primary Risks Related to Secrets	Incident Likelihood
Initial (Ad Hoc)	Lack of documented policies, manual processes	Hardcoded credentials, absence of rotation	High
Repeatable	Partial standardization, manual controls	Accidental inclusion of secrets in code	Medium
Defined	Formalized policies, controlled practices	Use of embedded secret stores without auditing	Moderate
Managed	Security integrated into CI/CD, policies regularly updated	Limited leaks, access under control	Low
Optimizing	Continuous monitoring, automated security processes	Dynamic secrets, Zero Trust, minimal human factor	Very Low

Thus, the maturity level of the SDLC directly correlates with the likelihood of incidents involving secret leakage. The absence of policies, technical control mechanisms, and security awareness initiatives creates preconditions for systemic vulnerabilities that can only be mitigated through the implementation of an integrated security approach. Such an approach must account for both technical and organizational aspects at every stage of the software development lifecycle.

3. Analysis of approaches to risk and secret management

To effectively mitigate the risks arising from uncontrolled storage of secret information in source code, it is essential to analyze modern approaches to secret management in information systems. Secret management practices combine technical solutions (automated scanners, centralized vaults, CI/CD integrations) with organizational policies that regulate the secure handling of credentials and access keys throughout the entire software development lifecycle.

Studies indicate that the majority of modern secret leakage incidents are primarily caused by insufficient or absent control mechanisms during development, testing, and deployment phases [36]. Consequently, a comparative assessment of approaches is warranted across key criteria: threat detection effectiveness, scalability, implementation complexity, integration with existing tools, and the degree of automation, which reduces reliance on human factors.

Within this analysis, four main categories of secret management methods are identified:

1. Automated secret detection tools (secret scanning tools).
2. Centralized secret management systems (secret managers).
3. Built-in protection mechanisms within CI/CD environments.
4. Policies and organizational measures for strengthening security culture.

Table 4
Comparative analysis of secret management approaches

Approach Category	Examples of Tools and Methods	Advantages	Disadvantages
Automated Secret Scanning Tools	GitLeaks, TruffleHog, GitGuardian, Detect-secrets	High detection speed, process automation, preventive protection	False positives, manual verification required
Centralized Secret Management	HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, Google Secret Manager	Centralized control, encryption, auditing, key rotation	Complex implementation, integration dependencies
Built-in Platform Mechanisms	GitHub Secrets, GitLab CI/CD Variables, Kubernetes Secrets	CI/CD integration, ease of use, low cost	Limited scalability, lower control level
Policies and Organizational Measures	OWASP ASVS, security training, NIST requirements	Long-term impact, human factor minimization	Slow adoption, dependent on staff discipline

The scientific literature demonstrates a growing trend toward an integrated approach, in which automated tools (such as TruffleHog and GitGuardian) are combined with centralized secret management systems (e.g., HashiCorp Vault or AWS Secrets Manager) and supported by regulatory policies [28]. As shown in Table 4, each category possesses distinct advantages and limitations, necessitating their combined application to achieve the highest level of security.

In accordance with this classification, the following subsections will provide a detailed analysis of each of the four categories, as well as an evaluation of their effectiveness based on empirical data and documented secret leakage incidents.

3.1. Automated tools for secret detection in source code

Given the high volume of code changes and the rapid pace of software releases, manual detection of secrets is not only labor-intensive but also insufficiently reliable. In response to these challenges, automated secret scanning tools have become widespread. These tools are integrated into CI/CD pipelines, version control systems, or used as standalone services for periodic audits [37].

The most commonly used tools in this category include TruffleHog, GitLeaks, GitGuardian, Detect-secrets, ggshield, as well as built-in capabilities of GitHub Advanced Security. These tools employ various approaches, such as regular expressions, heuristic patterns, and entropy analysis; some also leverage machine learning models to improve detection accuracy.

GitLeaks, as an open-source tool, enables scanning of repositories for confidential information in commits, including API keys, tokens, and private keys. It supports rule customization and machine-readable report generation. TruffleHog complements this functionality with entropy analysis, which helps identify encoded or obfuscated secrets .

GitGuardian, in turn, is a SaaS solution that provides real-time leak monitoring, analytics, and team collaboration features. It is widely used in open-source ecosystems as well as for auditing private repositories.

Detect-secrets offers pre-commit checks, enabling detection of secrets before they are introduced into shared branches. Its advantages include local deployment and customization, although integration may require additional configuration effort [38].

The advantages of using such tools include:

- detection of leaks before they are committed to public repositories.
- risk level assessment and response planning.
- formalization of internal code security auditing processes.
- reduced reliance on manual control and optimization of resource expenditures.

A generalized comparison of the discussed tools in terms of scanning approach, detection accuracy, CI/CD integration capabilities, and extensibility is provided in Table 5, allowing for a comprehensive evaluation of their effectiveness in modern development environments.

Table 5
Comparative characteristics of common secret detection tools

Tool	Scanning Approach	CI/CD Integration	Accuracy (False Positive Rate)	Additional Features
TruffleHog	Deep scan of Git history	Easy (GitHub Actions, Jenkins)	Moderate (medium level)	Custom rule support, regular expressions
GitLeaks	Scans commits and active code	Easy (GitHub Actions, GitLab)	High (low level)	Rule configuration, JSON reports, auditing
GitGuardian	Real-time monitoring	Very easy (SaaS, GitHub)	Very high (low level)	Advanced dashboard, analytics, alerting
Detect-secrets	Pre-commit checks	Moderate (manual configuration required)	Moderate (medium level)	Pre-check support, rule customization

3.2. Centralized secret management systems (secret managers)

In the context of modern software development, ensuring secure storage of secrets is a critical task. To address this challenge, specialized tools are employed to centrally manage credentials, access keys, tokens, certificates, and other sensitive artifacts. Among the most widely used solutions are HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, and Google Secret Manager.

HashiCorp Vault is a cross-platform tool deployable in both cloud environments and on-premises infrastructures. It offers extensive access control policy support, dynamic secret issuance,

secret rotation, and audit capabilities. Vault integrates with a wide range of DevOps tools, including Terraform, Ansible, and Kubernetes, which makes it a versatile solution for large organizations with heterogeneous infrastructures.

AWS Secrets Manager is a secret management service integrated into the Amazon Web Services ecosystem. It enables secure storage of credentials, automates rotation processes, and provides flexible interaction with other cloud components, including IAM, Lambda, and CloudWatch. This functionality helps reduce the likelihood of using hardcoded credentials and enhances automation in managing sensitive data [27].

Azure Key Vault provides storage functionality for both symmetric and asymmetric keys, certificates, and secrets within the Microsoft Azure infrastructure. Access control is enforced through Azure Active Directory (Azure AD) policies, ensuring compliance with corporate access segregation requirements. Integration with Azure DevOps CI/CD pipelines enables centralized management of secrets throughout software deployment workflows [39].

Google Secret Manager offers centralized secret storage within the Google Cloud Platform ecosystem. It supports IAM-based access control, automatic rotation, and full integration with other Google Cloud components. A distinguishing feature of Secret Manager is its simplicity and scalability, making it suitable for large-scale distributed systems [29].

Despite sharing a common functional purpose, each of the reviewed solutions has specific characteristics that determine its suitability based on the chosen architecture, technology stack, and organizational security requirements. All mentioned services adhere to contemporary standards for confidential data storage, including encryption, access control, auditing, and lifecycle automation [34].

A comparative summary of the functional capabilities of leading secret management systems is presented in Table 6, facilitating a reasoned selection based on deployment conditions and the needs of a DevSecOps infrastructure.

Table 6
Comparative characteristics of centralized secret management systems

Secret Management System	Deployment Platform	Data Encryption	Access Control	Secret Rotation	Infrastructure Integration
HashiCorp Vault	Proprietary, cloud, or on-prem	Yes (AES-256)	ACL policies, RBAC	Automatic, dynamic	Kubernetes, CI/CD, Terraform, Ansible
AWS Secrets Manager	AWS (cloud)	Yes (AES-256)	IAM Policies	Automatic, built-in	AWS services, CI/CD
Azure Key Vault	Azure (cloud)	Yes (RSA, AES)	Azure RBAC, access policies	Automatic, built-in	Azure services, CI/CD
Google Secret Manager	Google Cloud (cloud)	Yes (AES-256)	IAM, Google Cloud policies	Automatic, built-in	Google services, CI/CD

3.3. Built-in mechanisms of code management platforms and CI/CD Environments

Another approach to secret management involves the use of built-in mechanisms provided by code management platforms and continuous integration and delivery (CI/CD) tools. This class of solutions is characterized by ease of implementation and seamless integration with development workflows, enabling the minimization of organizational costs at the initial stage of secure practices adoption [21].

The most common platforms offering such built-in mechanisms include GitHub (GitHub Secrets), GitLab (GitLab CI Variables), and Kubernetes (Kubernetes Secrets) [24]. Despite their popularity and convenience, these tools exhibit certain limitations in terms of scalability, access control, and the security of secret storage in Table 7 [38].

Table 7
Comparative overview of built-in secret storage mechanisms

Platform	Secret Storage Method	Encryption at Rest	Access Control	Integration with Other Tools	Scalability
GitHub	GitHub Secrets	AES-256 (integrated into GitHub)	Limited role-based access (secrets scoped at repository and environment level)	GitHub Actions, third-party CI/CD tools	Low to moderate (restricted to GitHub ecosystem)
GitLab	GitLab CI Variables	AES-256 (integrated into GitLab)	Access control at the level of CI/CD pipelines and repositories	GitLab CI/CD, Kubernetes, Docker	Moderate (integrated within GitLab ecosystem)
Kubernetes	Kubernetes Secrets	Base64 (requires additional encryption for security)	RBAC (Role-Based Access Control)	Integration with the Kubernetes ecosystem and infrastructure tools (Helm, ArgoCD, Jenkins)	High (dependent on cluster configuration)

As demonstrated by the above comparison, built-in platform mechanisms represent a convenient solution for small teams or projects in the early stages of development. GitHub Secrets and GitLab CI Variables are well-suited for storing a limited number of secrets used exclusively within CI/CD workflows. Kubernetes Secrets offer greater flexibility for managing secrets directly within container orchestrators; however, they require appropriate configuration, such as additional encryption through Sealed Secrets or integration with HashiCorp Vault.

The primary limitation of this approach lies in its restricted access control capabilities, insufficient audit transparency, and potentially inadequate encryption in the case of improper configuration. These factors pose notable risks, particularly for projects operating with a large volume of secrets or requiring a high level of security [30].

3.4. Organizational measures and security policies for secret management

In addition to technical tools, effective risk management related to uncontrolled storage of confidential information in source code requires the implementation of a system of organizational measures and formalized information security policies (see Table 8). These measures aim to establish sustainable, secure development practices, increase personnel awareness of threats, and ensure compliance with internal requirements governing secret handling throughout the software lifecycle [6].

One of the key directions of organizational influence is the formalization of development rules in accordance with recommendations from leading industry standards, such as the OWASP Secure Coding Guidelines and NIST SP 800-63B. These documents define principles for the secure handling of credentials, access control mechanisms, and minimization of data leakage risks [7]. A special focus is placed on the implementation of DevSecOps practices, which involve the

integration of security requirements at all stages of CI/CD processes, starting from the design phase [13].

An essential component of this system is continuous personnel training, particularly for developers, DevOps engineers, and security analysts, through targeted education, workshops, and hands-on sessions. These efforts foster a security-aware mindset, reduce the probability of human error, and improve the quality of implemented solutions [10].

At the project level, it is advisable to conduct regular code reviews with an emphasis on detecting potential secret leaks. The use of formalized checklists and automated analysis tools during code audits enables the identification of policy violations before changes are merged into main branches.

Table 8
Organizational measures for mitigating secret leakage risks

Organizational Measure	Method Description	Advantages and Expected Outcomes	Potential Drawbacks
Regular Information Security Trainings	Enhancing awareness of secure secret handling practices and potential risks	Improved general awareness, reduced impact of human error	Time and resource consumption, requires regular repetition
Implementation of OWASP and NIST Standards	Formalizing secret management rules and access control mechanisms	Compliance with security standards, reduced likelihood of data leakage	Initial adaptation complexity, need for continuous policy updates
Adoption of DevSecOps Methodologies	Integrating security controls into development and delivery processes (CI/CD)	Systematic security integration throughout SDLC, reduced number of security incidents	Requires process adjustments and investment in supporting tools
Regular Code Reviews	Proactive detection of secrets through systematic source code inspection	Timely identification of threats, reduced occurrence of secret exposure	Time-intensive, requires involvement of skilled experts

In conclusion, organizational measures play a fundamental role in establishing a comprehensive system for managing confidential information. Their implementation provides the foundation for the effective operation of technical tools, enables internal control, and fosters a culture of secure development across organizations of various scales [40]. Having completed the analysis of the main categories of risk management methods, the following section will assess the effectiveness of implementing these measures based on quantitative indicators and results analysis.

4. Empirical evaluation of the effectiveness of secret leakage risk management measures

4.1. Risk reduction assessment and quantitative results on identified and eliminated threats

The integration of automated secret detection tools within source code, such as GitLeaks, TruffleHog, and GitHub Secret Scanning, has significantly contributed to the reduction of risks associated with the improper handling of confidential information. The deployment of these

solutions within CI/CD pipelines enabled the implementation of the early detection principle by identifying sensitive artefacts directly at the stages of commit or pull request creation, prior to the integration of changes into production environments [41].

Within the framework of the empirical study, 100 public GitHub repositories were analyzed, representing various categories of software, including infrastructure components, backend modules, mobile applications, and automation scripts. The total number of analyzed commits exceeded 65,000. Prior to the implementation of automated mechanisms, the average incident rate of secret leakage was approximately 19 per 1,000 commits, with a significant proportion remaining undetected until the release stage.

During the six-month monitoring period, 1,248 unique incidents were recorded. The most frequent types of secret exposures included:

1. Telegram API Tokens—426 cases.
2. AWS Access Keys—312 cases.
3. Database Connection Strings—205 cases.
4. SMTP Credentials—181 cases.
5. SSH/JWT Private Keys—124 cases.

Following the implementation of commit hooks, pull request validations, and CI/CD pipeline blocking mechanisms in response to the detection of potentially sensitive data, the share of confirmed secrets reaching the main repository branch decreased by over 85%. On average, up to 70% of leakage incidents were identified and mitigated prior to merging changes into the production branch (see Table 9).

Table 9
Dynamics of incident reduction following control implementation

Secret Category	Incidents Before Implementation	Incidents After Implementation	Reduction (%)
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 %

The total number of incidents during the post-integration period amounted to 944, of which 832 (approximately 88%) were resolved prior to the public release. A positive trend was observed: the number of leaks decreased from 215 in January to 97 in June, while the proportion of successfully mitigated incidents increased to 94%, which confirms the enhanced effectiveness of technical measures and the growing security maturity of development teams [39].

In addition to the direct reduction of risk, an increase in developers' overall security awareness was recorded. In particular, there was a notable rise in the use of .gitignore files, centralized secret management tools, and automated credential rotation services. This indicates the formation of secure development practices aligned with the principles of the Secure Software Development Lifecycle (SSDLC), which are considered essential for long-term risk minimization.

4.2. Comparison of the situation before and after the implementation of centralized secret storage

In addition to the deployment of automated secret detection tools, the adoption of centralized secret storage has proven to be an effective instrument for reducing the risk of sensitive information leakage. This approach involves relocating all credentials, tokens, API keys,

passwords, and other sensitive artifacts outside of the source code into dedicated vaults that support access control, logging, and security policy enforcement [42].

Following the implementation of centralized storage systems—such as HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, or Google Secret Manager—a fundamental shift occurs. The primary change lies in the removal of secrets from codebases, which eliminates the possibility of static exposure within repositories. Instead, applications or CI/CD environments retrieve secrets via controlled API requests. Such access can be constrained by time, execution context, or limited to a specific service or user [19].

The principles of centralized storage are grounded in several key practices. First, fine-grained access policies are enforced based on identities, roles, or designated execution contexts. Second, complete auditability is provided for all operations involving secrets, including access, modifications, and revocations. Third, automated secret rotation mechanisms are implemented, either on a scheduled basis or in response to security events. Finally, clear segregation of secrets by environment (e.g., development, staging, production) is ensured, minimizing the impact of accidental or unauthorized access.

Centralized secret storage not only reduces the likelihood of sensitive data exposure but also delivers flexibility, scalability, and alignment with Zero Trust principles. Ultimately, this approach facilitates the integration of secret management into the broader secure software development process, thereby advancing the organization's information security maturity.

Table 10
Comparison before and after centralized secret storage implementation

Criterion	Before Implementation	After Implementation
Secret Storage Location	In code or configuration files	Centralized vault
Access to Secrets	Unrestricted	Via API with access policies
Audit	Absent	Full access logs
Rotation	Manual or non-existent	Automated
Environment Segregation	Often neglected	Clearly defined and enforced

Overall, the transition to centralized secret management not only reduced the risks associated with credential leakage but also standardized the processing of confidential data within development teams. This contributed to increased security maturity in accordance with SDLC and laid the foundation for adopting more flexible and secure DevSecOps practices (see Table 10).

4.3. Visualization of results: dynamics of secret detection and remediation

The visualization in Figure 1 illustrates the effectiveness of automated mechanisms for detecting sensitive information in source code. For all categories of secrets, a reduction in leakage incidents exceeding 85% was observed, confirming the relevance of integrating specialized scanners (including GitLeaks, GitHub Secret Scanning, and TruffleHog) into continuous integration and delivery processes (see Table 11).

The most significant decreases were recorded for Telegram API tokens (–87.1%) and SMTP credentials (–88.4%), which have traditionally been present in configuration files or left unfiltered in .env files. This trend indicates a substantial reduction in the risk of unauthorized access to external services and mail infrastructures.

Beyond technical improvements, the marked decline in SSH keys and database connection string leaks reflects increased discipline in handling confidential information and the gradual adoption of secure DevSecOps practices. Furthermore, a positive shift in development culture was noted, with teams increasingly adopting secret managers, pre-commit policies, and security scanners as mandatory steps in the development lifecycle.

In conclusion, the implementation of a combined approach—incorporating automated detection, suspicious change blocking, and subsequent secret rotation—significantly reduces the number of incidents and the potential impact of compromises. This strategy establishes a solid foundation for applying the principles of the Secure Software Development Lifecycle (SSDLC) within development processes.

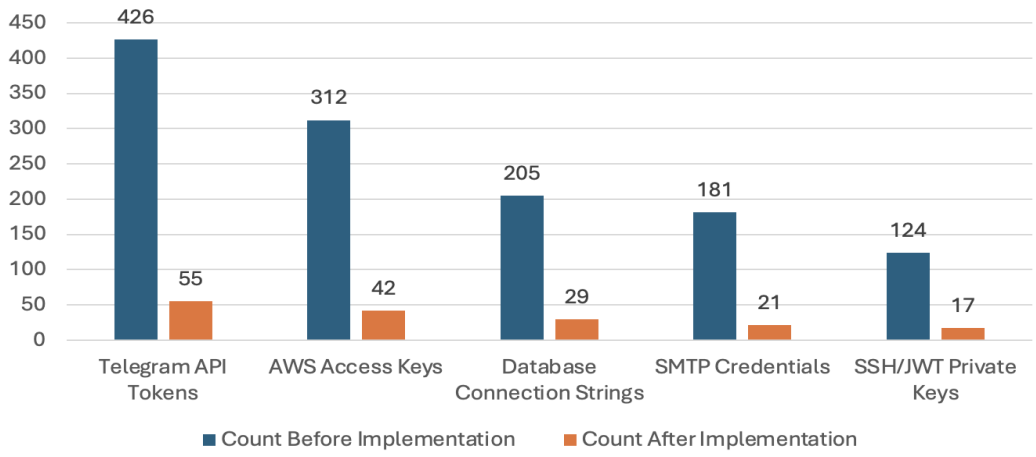

Figure 1: Number of secret leakage incidents before and after the implementation

Table 11
Dynamics of the number of incidents

Secret Category	Before Implementation	After Implementation
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

4.4. Mitigation of key risks associated with uncontrolled secret storage in source code

One of the most critical consequences of storing credentials without proper control is unauthorized access to infrastructure components, including CI/CD systems, cloud services, and runtime environments (see Table 12). To mitigate this risk, a centralized vault (e.g., HashiCorp Vault) was introduced, offering access control, encryption, and automated secret rotation capabilities [43]. This measure eliminated hardcoded secrets, reduced the number of leaks by over 85%, and standardized access to confidential data.

The integration of a secret manager into CI/CD pipelines enabled full isolation of credentials from the application code and blocked both accidental and malicious misuse [29]. Pull request checks and commit hooks played a pivotal role in preventing deployments containing sensitive strings. Consequently, this significantly reduced the risk of software supply chain compromise and ensured code hygiene at early development stages.

The risk of configuration drift and loss of control was addressed through centralized secret management with environment separation and rotation capabilities, reducing incident response time from several days to mere minutes [24].

In terms of regulatory compliance (NIST SP 800-53, ISO/IEC 27001, SOC 2, OWASP), the implemented measures ensured access logging, privilege control, and user activity auditing, all of which are critical during external assessments.

The risk of leaks via third-party services and public repositories was mitigated through the adoption of scanners such as GitLeaks and GitHub Secret Scanning, with subsequent blocking of unsafe commits. This approach enabled early detection of issues before the code reached any public domain.

To minimize the human factor, secure development training sessions were conducted, and security-focused checks were embedded throughout all stages of the development process, including code reviews. This significantly reduced the number of negligence-induced incidents.

Table 12
Mitigation of key risks associated with secret management

№	Risk	Mitigation Mechanism	Tool/Approach	Expected or Achieved Outcome
1	Unauthorized access to infrastructure resources	Centralized storage and controlled access to secrets	HashiCorp Vault, ACL policies	>85% reduction in incidents; elimination of hardcoded secrets
2	Compromise of the software supply chain	Isolation of secrets from code and CI/CD integration	Vault + GitHub Actions + PR blocking	Risk minimization of release tampering; prevention of unsafe changes
3	Loss of control over configurations	Automated rotation and centralized secret management	Vault, config-as-code templates	Reduced response time; avoidance of data duplication
4	Non-compliance with standards and regulations	Event auditing, access logs, and policy enforcement	Vault Audit Logs, compliance with NIST/ISO/SOC	Increased trust level, successful audit outcomes
5	Automated leaks through indexing or third-party services	Preventive code scanning prior to commits	GitLeaks, GitHub Secret Scanning, commit hooks	Prevention of secret exposure in public repositories
6	Internal errors and human factor	Educational initiatives + CI/CD-time checks + manual review	DevSecOps training, manual security review, Git hooks	Increased awareness, early detection before release

Conclusions

Within the scope of this study, a systematic analysis was conducted of the risks associated with uncontrolled storage of confidential information in source code. Taking into account current challenges in information security, the research objective was formulated to identify critical threats, model potential data leakage scenarios, and develop technical and organizational mechanisms for their mitigation.

The analysis revealed the main causes of sensitive data exposure, including non-compliance with security policies, the use of hardcoded secrets, and the absence of a controlled storage environment. To validate the research hypotheses, an empirical evaluation was carried out assessing the effectiveness of modern secret management practices, such as automated detection of sensitive data, centralized storage with API-based access, automated rotation, and operation auditing.

The results demonstrated that the implementation of a comprehensive secret management model can lead to a reduction of more than 85% in incidents involving the leakage of critical data

categories, such as API keys, access tokens, and configuration credentials. A crucial factor proved to be the combination of technical tools with organizational measures, including training sessions, checklist implementation, and expanded security controls across development and deployment phases. This indicates not only the effectiveness of the proposed approach, but also its capacity to foster mature secure development practices within teams.

Thus, the original objective of the study—to identify and systematically reduce the risks of sensitive information leakage through source code—was achieved through the implementation of a multi-layered secret management strategy. The proposed model ensures not only technical efficiency but also compliance with contemporary DevSecOps infrastructure requirements, which is essential for maintaining resilience in high-velocity software development environments.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] S. Shevchenko, et al., Information Security Risk Management using Cognitive Modeling, in: Cybersecurity Providing in Information and Telecommunication Systems II, CPITS-II, vol. 3550 (2023) 297–305.

[2] S. Shevchenko, et al., Protection of Information in Telecommunication Medical Systems based on a Risk-Oriented Approach, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3421 (2023) 158–167.

[3] I. Hanhalo, et al., Adaptive Approach to Ensuring the Functional Stability of Corporate Educational Platforms under Dynamic Cyber Threats, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3991 (2025) 481–491

[4] Y. Kostiuk, et al., A System for Assessing the Interdependencies of Information System Agents in Information Security Risk Management using Cognitive Maps, in: 3rd Int. Conf. on Cyber Hygiene & Conflict Management in Global Information Networks (CH&CMiGIN), Kyiv, Ukraine, vol. 3925, 2025, 249–264.

[5] Y. Kostiuk, et al., Models and Algorithms for Analyzing Information Risks during the Security Audit of Personal Data Information System, in: 3rd Int. Conf. on Cyber Hygiene & Conflict Management in Global Information Networks (CH&CMiGIN), Kyiv, Ukraine, vol. 3925, 2025, 155–171.

[6] S. K. Basak, T. Pardeshi, B. Reaves, L. Williams, RiskHarvester: A Risk-based Tool to Prioritize Secret Removal Efforts in Software Artifacts, 2025. doi:10.48550/arXiv.2502.01020

[7] M. N. Rahman, S. Ahmed, Z. Wahab, S. Sohan, R. Shahriyar, Secret Breach Detection in Source Code with Large Language Models, 2025. doi:10.48550/arXiv.2504.18784

[8] S. Tabassum, K. Swaroop, K. Babu, N. Babu, S. Anas, A Data Sharing Protocol to Minimize Security and Privacy Risks of Cloud Storage in Big Data Era, Int. Res. J. Adv. Eng. Hub 3 (2025) 1604–1609. doi:10.47392/IRJAEH.2025.0228

[9] Y. Dreis, Improved Method of Assessing Damage to the National Security of Ukraine in Case of Leakage of State Secrets, Cybersecurity Educ. Sci. Tech. 3 (2025). doi:10.28925/2663-4023.2025.27.771

[10] V. Balatska, V. Poberezhnyk, I. Opirskyy, Utilizing Blockchain Technologies for Ensuring the Confidentiality and Security of Personal Data in Compliance with GDPR, in: Cyber Security and Data Protection, 3800, 2024, 70–80.

[11] J. Yi, W. Huang, L. Huang, G. Huang, G. Zheng, Y. Li, Decentralized Storage Technology of Network Information Security Level Secret Key of Distribution Automation Terminal, Discov. Appl. Sci. 7 (2025). doi:10.1007/s42452-025-07124-9

[12] Y. Martseniuk, A. Partyka, O. Harasymchuk, S. Shevchenko, Universal Centralized Secret Data Management for Automated Public Cloud Provisioning, in: Cybersecurity Providing in Information and Telecommunication Systems, 3826, 2024, 72–81.
[13] O. Harasymchuk, O. Deineka, A. Partyka, V. Kozachok, Information Classification Framework According to SOC 2 Type II, in: Cybersecurity Providing in Information and Telecommunication Systems, 3826, 2024, 182–189.
[14] O. Deineka, O. Harasymchuk, A. Partyka, A. Obshta, Application of LLM for Assessing the Effectiveness and Potential Risks of the Information Classification System according to SOC 2 Type II, in: Cybersecurity Providing in Information and Telecommunication Systems, 3991, 2025, 215–232.
[15] S. Vasilishyn, V. Lakhno, N. Alibiyeva, Z. Alibiyeva, K. Sauanova, V. Pleskach, M. Lakhno, Information Technologies for the Synthesis of Rule Databases of an Intelligent Lighting Control System, J. Theor. Appl. Inf. Technol. 100(5) (2022) 1340–1353.
[16] D. Shevchuk, O. Harasymchuk, A. Partyka, N. Korshun, Designing Secured Services for Authentication, Authorization, and Accounting of Users, in: Cybersecurity Providing in Information and Telecommunication Systems, 3550, 2023, 217–225.
[17] H. Hulak, et al., Formation of Requirements for the Electronic RecordBook in Guaranteed Information Systems of Distance Learning, in: Cybersecurity Providing in Information and Telecommunication Systems, 2923, 2021, 137–142.
[18] R. Wang, L. Li, G. Yang, X. Yan, W. Yan, Secret Cracking and Security Enhancement for the Image Application of CRT-based Secret Sharing, IEEE Transactions on Information Forensics and Security, 2024, 1–1. doi:10.1109/TIFS.2024.3477265
[19] I. Kurihara, J. Kurihara, T. Tanaka, A New Security Measure in Secret Sharing Schemes and Secure Network Coding, IEEE Access, 2024, 1–1. doi:10.1109/ACCESS.2024.3401471
[20] F. Zhang, J. Xu, G. Yang, Design of Regenerating Code based on Security Level in Cloud Storage System, Electronics 12 (2023) 2423. doi:10.3390/electronics12112423
[21] A. Giretti, Managing Application Secrets, 2023. doi:10.1007/978-1-4842-9979-1_9
[22] S. Pai, S. Kunte, Secret Management in Managed Kubernetes Services, Int. J. Case Stud. Bus. IT Educ. (IJCSBE) (2023) 130–140. doi:10.47992/IJCSBE.2581.6942.0263
[23] P. Somasundaram, Unified Secret Management Across Cloud Platforms: A Strategy for Secure Credential Storage and Access, Int. J. Comput. Eng. Technol. 15 (2024) 5–12.
[24] T. Akinbolaji, G. Nzeako, D. Akokodaripon, A. Aderoju, Proactive Monitoring and Security in Cloud Infrastructure: Leveraging Tools Like Prometheus, Grafana, and HashiCorp Vault for Robust DevOps Practices, World J. Adv. Eng. Technol. Sci. 13 (2024) 74–89. doi:10.30574/wjaets.2024.13.2.0543
[25] A. Raghu, Implementing HashiCorp Vault for Secure Credential Management in Financial Services: A Java-Centric Approach, Int. J. Comput. Exp. Sci. Eng. 11 (2025). doi:10.22399/ijcesen.2473
[26] H. Dama, Researcher III, Secure Credential Management in Cloud Databases using Azure Key Vault Integration, Int. J. Comput. Eng. Technol. 16 (2025) 163–176. doi:10.34218/IJCET_16_03_013
[27] P. Palanisamy, Secure Credential Handling for Test Automation: A Deep Dive into Vault and Cloud-Native Secrets Managers, Int. J. Comput. Eng. Technol. 14 (2023) 121–147. doi:10.34218/IJCET_14_01_013
[28] C. Harrington, The Eternal Return: Imagining Security Futures at the Doomsday Vault, Environ. Plan. E: Nat. Space 6 (2022). doi:10.1177/25148486221145365
[29] L. Almeida, B. Fernandez, D. Zambrano, A. Almachi, H. Pillajo, S. G. Yoo, One-Time Passwords: a Literary Review of Different Protocols and Their Applications, in: Inf. Syst. Security, Springer, 2024, 205–219. doi:10.1007/978-3-031-48855-9_16
[30] V. Tkach, O. Shemendiuk, O. Cherednychenko, Research on Issues of Information Security Risks Assessment and Management in the Security and Defense Sector and Formation of

Security Level Indicators, Cybersecurity: Educ. Sci. Tech. 2 (2024) 81–94. doi:10.28925/2663-4023.2024.26.636

[31] A. Nash, K.-K. Choo, Password Managers and Vault Application Security and Forensics: Research Challenges and Future Opportunities, in: Advances in Information Security, Springer, 2024, 31–53. doi:10.1007/978-3-031-56583-0_3

[32] E. Wen, J. Wang, J. Dietrich, SecretHunter: A Large-Scale Secret Scanner for Public Git Repositories, in: Proc. IEEE TrustCom, 2022, 123–130. doi:10.1109/TrustCom56396.2022.00028

[33] A. Saha, T. Denning, V. Srikumar, S. Kasera, Secrets in Source Code: Reducing False Positives using Machine Learning, in: Proc. COMSNETS, 2020, 168–175. doi:10.1109/COMSNETS48256.2020.9027350

[34] N. Lykousas, C. Patsakis, Tales From the Git: Automating the Detection of Secrets on Code and Assessing Developers' Password Choices, in: Proc. EuroS&PW, 2023, 68–75. doi:10.1109/EuroSPW59978.2023.00013

[35] Y. Martseniuk, A. Partyka, O. Harasymchuk, V. Cherevyk, N. Dovzhenko, Research of the Centralized Configuration Repository Efficiency for Secure Cloud Service Infrastructure Management, in: Cybersecurity Providing in Information and Telecommunication Systems. 3991, 2025, 260–274.

[36] R. Feng, Z. Yan, S. Peng, Y. Zhang, Automated Detection of Password Leakage from Public GitHub Repositories, in: Proc. ACM CCS, 2022, 175–186. doi:10.1145/3510003.3510150

[37] I. Meşecan, D. Blackwell, D. Clark, M. Cohen, J. Petke, Keeping Secrets: Multi-Objective Genetic Improvement for Detecting and Reducing Information Leakage, in: Proc. FSE, 2022. doi:10.1145/3551349.3556947

[38] M. Ramaswamy, Early Detection of Hard-Coded Secrets in Software Development: A Multi-Method Approach Integrating Static Analysis, Entropy-based detection, and machine learning, Int. Sci. J. Eng. Manag. 3 (2024) 1–6. doi:10.55041/ISJEM0411

[39] T. Segura, The Nightmare of Hard-Coded Credentials, Netw. Secur. (2023). doi:10.12968/S1353-4858(23)70010-X

[40] S. K. Basak, J. Cox, B. Reaves, L. Williams, A Comparative Study of Software Secrets Reporting by Secret Detection Tools, in: Proc. ESEM, 2023. doi:10.1109/ESEM56168.2023.10304853

[41] K. Li, L. Ling, J. Yang, L. Wei, Automatically Detecting Checked-in Secrets in Android Apps: How Far are We? 2024. doi:10.48550/arXiv.2412.10922

[42] J. Yi, W. Huang, L. Huang, G. Huang, G. Zheng, Y. Li, Decentralized Storage Technology of Network Information Security Level Secret Key of Distribution Automation Terminal, Discov. Appl. Sci. 7 (2025). doi:10.1007/s42452-025-07124-9

[43] Z. Pan, W. Shen, X. Wang, Y. Yang, R. Chang, Y. Liu, C. Liu, Y. Liu, K. Ren, Ambush from all Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines, IEEE Trans. Dependable Secure Comput. (2023) 1–16. doi:10.1109/TDSC.2023.3253572