UNIVERSITÀ DEGLI STUDI DI BARI "ALDO MORO"

DEPARTMENT OF COMPUTER SCIENCE

Bachelor's Degree in computer science

Thesis in Models and Methods for Application Security

Application security through DevSecOps

Supervisor:
Prof. Donato Impedovo

Co-supervisor:
Dott. Stefano Galantucci

Candidate:
Vincenzo Quagliarella

ACADEMIC YEAR 2021/22

Abstract

This thesis focuses on DevSecOps, an innovative methodology that emphasizes the theme of security, an aspect that is often overlooked by companies that produce software. This methodology is an extension of DevOps and integrates security into all phases of the software development life cycle. In the paper, the methodology is introduced and described, followed by a collection and analysis of difficulties and their respective solutions. In addition, the paper describes aspects of the methodology that require further research for improvement. The aim is to motivate and encourage its adoption in today's world.

Contents

Summary

Chapter 1 – Introduction

1.1 Context of the thesis

The use of outdated and inefficient methodologies that slow down software production is very common. However, customers are becoming more demanding, and the market is increasingly competitive. To this end, there is an urgent need to adopt new methodologies that allow for the faster production and release of software, to minimize manual errors with the help of automation.

In this scenario, DevOps [1] emerged as a software development methodology born from the combination of Development and Operations. Thanks to collaboration and communication between the parties, as well as the use of automation, this methodology enables faster software lifecycle phases, allowing for faster releases without compromising quality, continuous integration, and automated release.

Although DevOps brings improvements over outdated methodologies, unfortunately, it often neglects the security aspect. Companies, driven by the need to release software quickly, tend to underestimate the importance of security and not invest sufficient resources in this area. However, this choice can lead to security vulnerabilities and problems with customers. To this end, the DevOps methodology has evolved into DevSecOps, which combines Development, Security, and Operations. This methodology is an approach to software culture, automation, and design that integrates security as a shared responsibility throughout the entire IT lifecycle. A key aspect of this methodology is that each member of the team is responsible for security and must address this aspect. Even developers must be able to address the security of the code, without it falling solely on security experts.

DevSecOps [2] enables the integration of security from the earliest stages of the software lifecycle, ensuring the timely detection of vulnerabilities and preventing security issues from propagating to later stages. The main advantage of this methodology is that it allows for a high level of security to be achieved through automation, without increasing production times.

1.2 Thesis objective

This thesis is focused on the adoption of the DevSecOps methodology, with the aim of promoting and justifying its adoption. In this sense, a detailed description of the methodology, its elements, and the security culture that characterizes it is provided. Subsequently, an analysis is conducted to identify the difficulties related to the adoption of the methodology, assigning a level of criticality to each of them and proposing appropriate solutions.

To facilitate the analysis, each difficulty is assigned to a specific theme, to define the context in which it may arise.

The proposed solutions derive from combinations of partial solutions that can be reused to solve other difficulties. Once the analysis is completed, the areas requiring further research to improve the methodology are identified. The results obtained are used to strengthen the motivations in favour of the diffusion of DevSecOps.

1.3 Content organization

Chapter 1 of the thesis defines the context and objective of the thesis work, followed by a detailed description of the DevSecOps methodology and the integration of security in DevOps in Chapter 2. In Chapter 3, the challenges that may arise during the adoption of DevSecOps are identified and analysed, with an evaluation of their importance, a description of their respective solutions, and the presentation of results in tabular form. Furthermore, areas requiring further research are defined. In the concluding chapter, conclusions on the DevSecOps methodology are drawn, reinforcing the reasons why an organization producing software should invest in such an approach.

Chapter 2 – State of the art

2.1 DevOps

DevOps [3] represents a software development methodology, used in the field of computer science, which focuses on communication, collaboration, and integration between developers and IT operations managers.

Companies that adopt DevOps are generally oriented towards making frequent software releases. This release cycle is called Continuous Deployment, in which modified software is automatically transferred to the production environment, an area accessible to users who can use the software.

The DevOps methodology supports companies in managing releases through the standardization of development environments. The integration of this methodology aims to automate the product release, test it, and facilitate its maintenance through monitoring, bug fixing, and minor version releases. The goal is to shorten the software lifecycle, promoting rapid and timely releases.

DevOps aims to optimize the interaction between the development team and the operations team through the implementation of a set of processes and methods aimed at promoting communication and mutual collaboration, to ensure project success.

Executing a proper transition to DevOps is not a straightforward process, as it requires a mindset shift within the organization, particularly among the working groups, which must be cross-functional. This implies the formation of groups composed of individuals with diverse skills and experiences, who collaborate to achieve a common goal.

As shown in Figure 1, DevOps requires collaboration among the Development group, the IT Operations group, and the Quality Assurance (QA) group.

Adopting the DevOps methodology requires organizational change, which brings several benefits, including an improvement in the skills of group members over time. In other words, many members of the groups, especially developers, may encounter technologies they are not familiar with and learn through practice. Indeed, in DevOps, it is not necessary for all group members to have complete proficiency in the use of available tools from the outset.

Figure 1 : DevOps [4]

2.2 DevSecOps

The DevOps methodology aims to speed up all stages of the software development lifecycle, enabling fast and high-quality releases. However, an aspect of software development that is often neglected is its security.

In the IT field, security issues can cause considerable damage, especially if sensitive data is lost or leaked, and services become inaccessible. The severity of the damage caused by such security incidents varies according to the business sector but can also result in severe economic and legal consequences. Furthermore, the recent introduction of the General Data Protection Regulation (GDPR) has made cybersecurity an increasingly critical issue for companies. Therefore, it is essential for companies to be aware of the risks associated with security incidents and to take measures to safeguard their financial resources and reputation.

The DevSecOps methodology is recommended for organizations that aim to release software quickly and securely, as it seeks to integrate security at every stage of the software development lifecycle.

However, developers are often reluctant to perform code checks as they consider it a time-consuming activity that slows down software development and updates. Nevertheless, this neglect often results in the presence of vulnerabilities in the code that can persist even in subsequent phases, with the risk of releasing software that presents vulnerabilities. This can cause severe damage to the company, expose users to security risks, and result in loss of time necessary to identify and mitigate security flaws.

2.2.1 Security-oriented culture

For DevSecOps to be adopted by a company, it requires a cultural shift towards a security-oriented culture to be established [5]. This is a crucial practice, as it is not desirable to impose a methodology without first making the necessary changes to facilitate its adoption, in this case, a culture oriented towards security.

A security-oriented culture is characterized by the following points:

- Collaboration: to integrate security principles, it is necessary to promote collaboration between the development, IT operations, and security groups within the company. It is desirable to establish a single working group composed of members from the main groups to promote teamwork and collaboration among all team members, regardless of their specialization as developers or security experts.
- Sharing knowledge: it is essential to share knowledge in security matters because DevSecOps uses numerous security automation tools that do not guarantee security if not used by specialized personnel. Therefore, security experts must share their knowledge with other groups. It is advisable to appoint "Security Champions," i.e., programmers who represent members of their respective groups with greater knowledge in security matters. The security group is responsible for teaching these champions security practices, which they, in turn, should spread the knowledge gained to the rest of their groups.
- Feedback: feedback should be immediate and continuous. It is crucial that all members of distinct groups are aware of any issues, generally identified by security experts, so that they can be informed promptly.
- Continuous improvement mindset: the ability to include activities that not only ensure quality but also integrate security without slowing down software production. This includes continuous monitoring of application security, use of security tools, threat analysis, and, most importantly, the use of measures to evaluate software performance.
- Communication: in DevSecOps, the security team plays a key role in communication, making themselves available to other teams from the very beginning. Communication is straightforward, with security experts asking developers or IT operations members if they need help and in what area. However, their involvement should not significantly slow down their work.
- Responsibility: each member of every team is responsible for their own activities, particularly those that integrate security. However, it is not required that everyone be a security expert. Nevertheless, it is essential to spread an adequate level of knowledge throughout the organization, so that everyone can take on these responsibilities.

- Trust: the security team plays a crucial role in DevSecOps, but for their work to be effective, it is essential that other teams trust them. Developers may often find themselves thanking security experts, and in turn, security team members should treat developers with respect. Criticizing developers' work excessively is not recommended unless strictly necessary.
- Experimentation: experimentation is crucial in DevSecOps since there is no standardization of tools to use, and each member can explore different tools before finding the right one, which is especially useful for practice and learning new tools.
- Leadership: to promote adoption and support of the DevSecOps culture, leadership actions are necessary. However, further initiatives are required for this approach to be spread throughout the entire organizational structure.
- Blameless: in an organization that adopts DevSecOps, each member is responsible for their actions. However, one should never blame or accuse someone who has made a mistake. Criticizing a developer's work harshly brings no advantages, but rather conflicts and tensions within the organization.
- Hiring new personnel: it is necessary to introduce into the organization people who have practical and theoretical knowledge of DevSecOps, such as those who can use some code static analysis tools and who can identify false positives, often produced by the same tools.
- Transparency: it is essential that all groups have a complete view of the software components, as well as all planned or ongoing activities.

Being able to fully meet these characteristics represents a challenge for all business organizations. In the next chapter, the challenges that DevSecOps can present will be analysed, many of which represent obstacles to the characteristics of a security-focused culture. In adopting this methodology, it is crucial to respect the defining principle, which defines the way security is integrated into the software development lifecycle phases, commonly known as *shift-left*, which will be examined in the next section.

2.2.2 Shift-Left

One of the fundamental principles of DevSecOps methodology is the *shift-left* principle [6], which refers to the idea that groups should ensure application security from the initial stages of the software development life cycle.

Shift-left allows for the integration of security measures, typically integrated only at the end, throughout the software development life cycle. As a result, the software is designed with best security practices from the outset, facilitating the early detection of vulnerabilities, on which appropriate countermeasures will be applied. This minimizes delays and problems that are often caused by the late integration of security.

The *shift-left* principle of DevSecOps states that security integration should be considered from the initial stages of the software development life cycle, to identify and address vulnerabilities early on. While security tools are important, the fundamental component is represented by the people involved in the development process. DevOps aims to improve collaboration between the development group and the operations group, while DevSecOps has the additional responsibility of integrating security throughout the process. This has led to greater involvement of development groups in managing security practices, for which they must take responsibility.

2.2.3 CI/CD

CI/CD is a fundamental method used in this context [7]. Its objective is to speed up the software release process.

CI/CD is an acronym that stands for the combination of Continuous Integration (CI) and Continuous Delivery/Deployment (CD):

1. Continuous Integration: It is a software development approach that is part of the DevOps and DevSecOps methodology. It involves developers continuously integrating code into a centralized repository, with builds and tests automatically executed. Continuous integration concerns the build creation phase and the software release process integration. Typically, CI consists of two components: an automation component such as CI itself or a build creation service, and a cultural component such as the decision to integrate new code more frequently.
   In the context of the DevOps and DevSecOps methodology, developers use a version control

system like Git to commit to a shared repository. Before actual integration, developers can perform unit tests locally.

Continuous integration involves the automatic creation of a build and the execution of unit tests to identify any errors. Continuous integration concerns the build creation and unit testing phases of the software release process. For each new released version, a new build is created, and testing activity is performed.

2. Continuous Delivery/Deployment: In software development, this phase occurs after continuous integration and requires that the code has been built and evaluated in different environments. This phase allows for automatic implementation of code updates in the case of Continuous Deployment, without the need for manual verification, which is required in Continuous Delivery.

   Incorporating continuous implementation involves the rapid delivery of new features to users without compromising software quality. However, in critical sectors such as finance, where greater attention is required for security and system stability, Continuous Delivery is preferable, as it requires human intervention to control and proceed with implementation, via a button click.

   In general, the phases of Continuous Integration (CI) and Continuous Delivery/Deployment (CD) represent a complex software development process that requires a high degree of technical and organizational skills to ensure software quality. The CI phase focuses on continuous code integration and validation through automated testing. On the other hand, the CD phase is concerned with deploying the application to the main server, which can have significant financial consequences for the company.

   In particular, the CD phase represents a significant technical and organizational challenge, as any incorrect release could result in the loss of customers and economic damages for the company. For example, when transitioning from one version of an application to a subsequent version, the release must be managed carefully, and if there are issues with the new version, a roll-back to the previous version may be necessary. Therefore, managing the CD phase requires a high level of attention, constant monitoring, and well-defined roll-back procedures to ensure successful implementation.

Continuous Integration/Continuous Deployment (CI/CD) tools can be used to automate software development, deployment, and testing. There are specific tools for continuous integration, continuous development and deployment, and continuous testing.

Among the well-known open-source CI/CD tools, Jenkins is an automation server designed to handle the entire process from simple continuous integration to continuous delivery hub. Tekton Pipelines, on the other hand, is a specific CI/CD framework for Kubernetes platforms.

Below are some of the open-source CI/CD tools that can be considered in addition to Jenkins and Tekton Pipelines: Spinnaker, a CD platform designed for multi-cloud environments; GoCD, a CI/CD server focused on modelling and visualization; Concourse, an open-source continuous resource; Screwdriver, a creation platform designed for CD.

Additionally, teams can evaluate the use of CI/CD tools offered by various providers, including major public cloud providers, GitLab, CircleCI, Travis CI, and others.

It should be noted that essential DevOps tools such as configuration automation (Ansible, Chef, and Puppet), container runtimes (Docker, rkt, and cr-io), and container orchestration (Kubernetes), while not strictly CI/CD tools, are often part of CI/CD workflows.

Figure 2 shows that the CI phase precedes the CD phase and shows the subtle difference between Continuous Deployment and Continuous Delivery.

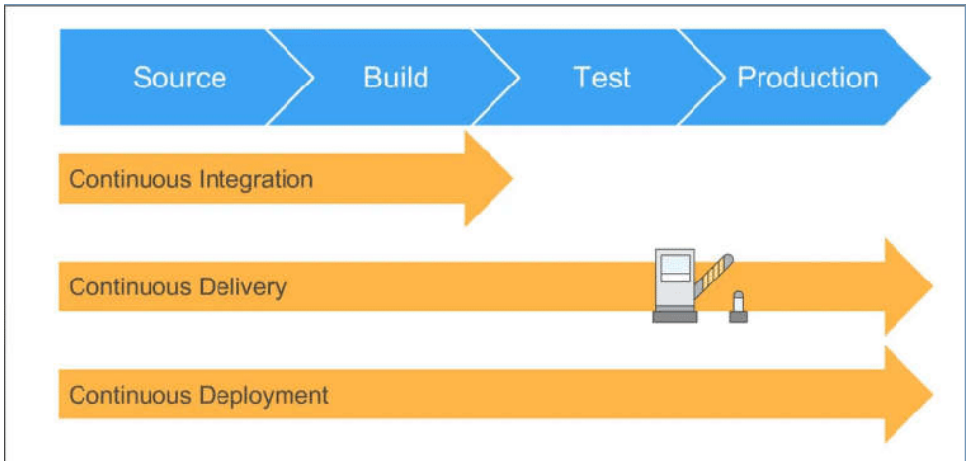

Figure 2: CI/CD [8]

2.2.4 Pipeline DevSecOps

The DevSecOps pipeline [9] encompasses all stages of the DevOps process but necessitates the integration of specific security activities in each phase. Table 1 illustrates all the process stages, their execution, and the security activities integrated into each stage.

Phase	Execution	Integrated security activities
Plan	During this phase, requirements are collected, a strategy is defined, and activities are planned to be conducted before moving to the next phase.	This phase includes the evaluation of potential security threats and risk analysis for the identification and management of threats and risks. In addition, security requirements are gathered, and security impacts are analysed to establish necessary modifications.
Code	During this phase, the system functionalities are implemented, during which the code is uploaded to a shared central repository.	This phase includes code review by security experts and the use of code security guidelines.
Build	In this phase, the code is compiled and packaged into a package or image.	During this phase, SAST (Static Analysis Security Testing) tools are used to identify security vulnerabilities in the source code, and SCA (Software Composition Analysis) tools to automatically identify third-party dependencies and detect any vulnerabilities in external code.

Test	A dedicated phase for defining and executing tests in a dedicated environment updated with the latest version of the system.	During this phase of the process, a vulnerability analysis of the system is conducted, and penetration testing is performed, aimed at identifying any weak points in the system and, if possible, exploiting them to test the effectiveness of the security measures adopted.
Release	After the software has undergone testing, it can be considered ready to be deployed in production environments.	During these phases, appropriate measures are taken to ensure secure distribution of the software, using artifact repositories that allow for storage of all artifacts generated during the deployment phase.
Deploy	The software is released in a production environment and made available to clients.	
Operate	Phase that involves maintenance activities, in case of any malfunctions or unexpected errors.	Different tools are used to ensure the security of the system: Security Information and Event Management (SIEM), which offers real-time monitoring and analysis of events; Security Orchestration, Automation and Response (SOAR), which deals with responding to security incidents with automated operations; finally, Intrusion Detection Systems (IDS), which is a software or hardware device that detects unauthorized access to computers or local networks.
Monitor	Phase that involves monitoring activities through which data, metrics, and software performance are collected and evaluated. This phase is especially important as it allows for product improvement; therefore, after this phase, the process cyclically starts again from the planning phase.	

Table 1: Phases of DevSecOps and integrated security activities

The DevSecOps pipeline, which involves all the phases and the integrated security activities, is represented in Figure 3.

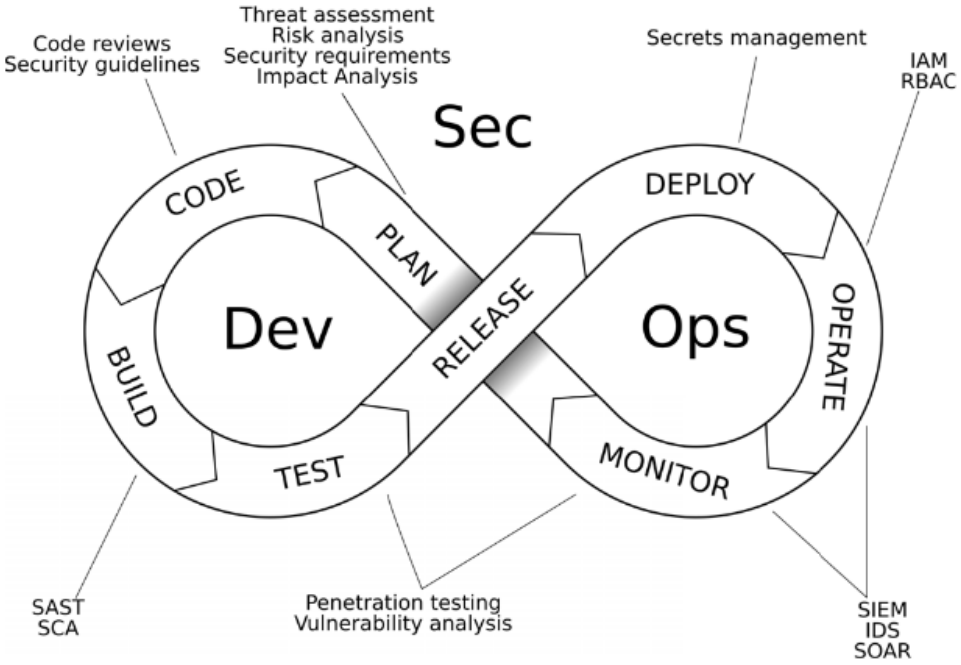

Figure 3: Pipeline DevSecOps [9]

2.2.5 Elements of DevSecOps

Before proceeding with the analysis of the methodology, it is necessary to introduce some distinctive elements of DevSecOps, as they too may present issues that will be examined later. The elements are as follows:

- Container: [10] they represent executable units of software that allow packaging of application code, libraries, and their dependencies into a format that can run on any platform, whether desktop or cloud. Since containers require some form of operating system virtualization, the features of that operating system are used to isolate and manage process resources. One of the main advantages of containers is portability, as they do not require a host operating system, i.e., a virtual copy of the hardware needed to run the operating system. In DevOps, the use of containers often refers to Docker, a popular software specifically designed for running processes in containers, and Kubernetes, an open-source system for orchestrating and managing containerized workloads and services that can interact constructively.

The container image may be subject to two security practices, image scanning, and image minimization. Image scanning is performed to identify outdated, expired, or vulnerable software components that may be present in the container image. Image minimization is performed to reduce the likelihood of attacks by reducing the number of files and components contained in the container. Additionally, the container image may contain sensitive information, such as passwords, certificates, and cryptographic keys, which have been inadvertently entered, and malfunctions may occur due to configuration errors or attacks that should have been prevented.

- Security tools: there are various tools available to identify and mitigate security issues from the initial stages of the software development life cycle. It is particularly important to prevent such issues from being passed on to later stages, and therefore DevSecOps uses different security technologies and tools to achieve this goal.

    - Static application security testing (SAST): it is a technique for analysing the source code of an application, which is performed without running the application itself. In practice, a SAST analysis software evaluates the source code of the application to identify any security vulnerabilities.

      During the analysis, the SAST software can identify security issues such as invalid input vulnerabilities, unauthorized access vulnerabilities, insecure code vulnerabilities, and other common vulnerabilities that could be used by an attacker to compromise the application or access sensitive data. The advantage of SAST is that it can identify security issues before the application is executed, allowing developers to fix vulnerabilities before the application is released. However, SAST may also produce false positives, i.e., identify as vulnerabilities some source code behaviours that are secure.

    - Dynamic application security testing (DAST): it is a testing method that involves analysing the application during its execution. Unlike SAST, which requires access to the source code or knowledge of the internal structure or mechanisms of the application, DAST does not use it. In DAST, attack activities are simulated to evaluate the behaviour of the application and identify any security vulnerabilities.

      DAST tests can be performed manually or automated and can evaluate different areas of the application. They can detect vulnerabilities such as SQL injections, cross-site scripting (XSS), authentication vulnerabilities, authorization vulnerabilities, and other common vulnerabilities that could allow an attacker to compromise the application or

access sensitive data. However, DAST tests may produce false positives as they are based on an attack simulation and may identify as vulnerabilities some secure behaviours of the application.

- Software composition analysis (SCA): it is a methodological approach used for the identification and management of risks associated with the use of third-party software and open-source libraries within an application. As the use of third-party software has become increasingly common, it is essential to understand software dependencies and the associated risks. SCA employs analytical tools to evaluate the libraries used in the application and provides recommendations to mitigate risks. Additionally, SCA helps ensure compliance with the open-source licenses used, thereby avoiding legal issues associated with them.

- Interactive application security testing (IAST): it is a dynamic application testing technique that combines security testing with functional testing of the application. It is a hybrid of SAST and DAST and can identify and report application security vulnerabilities during runtime, providing quick issue identification and reducing the risk of false positives.

  IAST works by incorporating testing agents within the application, which monitor the application's activity during runtime, detecting any security vulnerabilities and reporting these vulnerabilities in real-time. Additionally, the data collected by testing agents can be used to provide detailed information about security issues, such as the root cause of the vulnerability. IAST offers several advantages over other security testing techniques, such as the reduced risk of false positives, real-time vulnerability identification, and the ability to provide detailed information about the root cause of the vulnerability.

- Threat modelling: it is a computer security technique aimed at identifying and assessing risks and threats that may arise within computer systems and applications. This technique aims to analyse system vulnerabilities, as well as potential threats, to understand how an attacker could compromise the system and the consequences of such an attack. The process of threat modelling involves several phases, including identifying assets to protect, identifying threats and vulnerabilities, defining security controls, and documenting results. This technique can be applied at various stages of the software development life cycle and can be conducted manually or with the help of software tools. Furthermore, it can be integrated with other security techniques, such as

dynamic and static security testing, to further improve the security of computer systems and applications.

- Infrastructure as Code (IaC): [12] it is a practice aimed at defining and automating the software testing and deployment environment. Instead of manually configuring a virtual machine with the necessary libraries and specifications for the software system, the infrastructure configurations are defined once using automated tools to generate the virtual machine image. The "as-code" suffix refers to the practice of developing infrastructure specification files using specific languages such as Puppet and Chef. IaC minimizes manual intervention by automating the process but may require maintenance activities to ensure the effectiveness and security of the infrastructure.

Chapter 3 – Analysis of the methodology

This chapter conducts an analysis of the DevSecOps methodology. The challenges that this methodology presents were collected and analysed, and for each of them, the level of criticality was defined, and a solution was proposed. Subsequently, the obtained results were presented in tabular form. Finally, the areas of the methodology that require further research were identified.
DevSecOps [13] gains interest in academia and organizations but poses challenges due to complexity and cultural changes required for security practices.

An organization that intends to adopt DevSecOps must consider many factors to avoid errors in adaptation, use of tools, and application of security practices, which could worsen the company's situation. Therefore, it is crucial to be aware of all the challenges related to the methodology, as well as to have the necessary skills to address them through appropriate solutions.

To classify all the difficulties of the methodology, four main themes have been defined:

- Tools**:** theme that addresses the difficulties related to security tools in DevSecOps, including incorrect use and lack of standardization of the same.
- Practices**:** theme that addresses the difficulties related to DevOps, including the integration of manual security practices into DevSecOps, as well as the problems encountered in the CI/CD pipeline.
- Infrastructures**:** theme that addresses all the issues that could arise in the implementation of DevSecOps in complex environments.
- Personnel and organizational culture**:** theme that addresses the issues related to personnel and organizational culture, including security skill gaps, communication and collaboration issues, role division, and cultural transformation difficulties.

The levels of difficulty criticality are assigned according to the following criteria:

- Low level: the difficulty does not compromise the adoption of the methodology; therefore, the organization is not required to address it.
- Medium level: the difficulty may compromise the adoption of the methodology; therefore, it is strongly recommended that the organization address it.
- High level: the difficulty compromises the adoption of the methodology; therefore, it is essential that the organization address it.

The following sections provide a description of the difficulties related to each of the themes, along with an evaluation of their level of criticality and the presentation of an appropriate solution.

3.1 – Challenges and solutions related to security tools

As reported in the previous chapter, DevSecOps employs a shift-left policy that enforces the use of security tools from the initial stages of the development cycle to promptly identify and mitigate potential security threats.

Tools are used to monitor, identify, and resolve all issues encountered through the pipeline.

Another crucial aspect of DevSecOps concerns the standardization of processes, which implies that automated tools are preferred, as they allow for the implementation of triggers within the pipeline that conduct certain automatic actions or security assessments, often without human intervention.

Security tools are essential for DevSecOps, so it is crucial for an organization to be aware of the associated issues and how to act accordingly.

To this end, this section addresses and examines all the issues related to security tools, assigning a degree of criticality and a solution for each one.

3.1.1 – C1: difficulties in selecting tools.

The use of tools is essential in DevSecOps, as there are various tools available for each phase of the methodology. However, it is difficult to establish standardization of tools.

Variations in toolsets [14] constitute an organizational problem, as coordination and collaboration among different security, development, and IT operations teams are crucial for effective security management. However, these differences create a barrier to implementing security appropriately.

Every programmer has individual preferences for the tools to use in each phase of development, and while many of these tools can be integrated with others, they require specific configuration and attention to resources for their maintenance. A simple solution to this problem would be to impose a set of standard tools for all programmers. However, since each tool has its own advantages and disadvantages, there could be a situation where the ideal tool is not part of the standard set.

The lack of standardization of tools [15] is one of the causes of poor security awareness. Some developers argue that, in a DevSecOps context, it would be preferable for everyone to adopt easy and common tools. The complexity of automating security practices is amplified by this tool issue.

Another factor to consider is the variation of tools used in distinct phases of the DevSecOps pipeline: there may be additions of tools in later stages [16]. However, this choice should be avoided at all

costs. Therefore, it is essential to select the right tools at the beginning of the project, even if it requires a training period to learn how to use them, thus avoiding interruptions in development later.

Finally, an aspect to consider when choosing tools is that inadequate distribution tools could be employed [17]. It should be noted that the choice of improper distribution tools can cause insecure software deployment. Although automatic software release can be advantageous for system security, improper use of distribution tools or lack of adoption of proper security practices can cause problems. It is therefore important that those who work in all phases of the software life cycle pay close attention to the selection and use of distribution tools.

Criticality level

Based on the criticality evaluation, this difficulty can be considered of medium level, as in most cases, organizations are still able to produce secure software in a timely manner. However, such a level of success could be compromised when integrating tools during the project or when using inappropriate tools.

Solution

The main difficulty is that there are too many tools available at each stage of the DevSecOps pipeline. It has been observed that developers are not able to have a comprehensive overview of what to use [14], and it is also important to avoid improper use of the tools and especially modifications or additions in the advanced stages of the pipeline, as they would lead to avoidable time and costs. Almost all developers agree that to solve this problem, the community should converge towards standardized tools. In this way, organizations will be much more inclined to follow the selection and use guidelines of these standardized tools, and at the same time, group members would have a general view of what to use, including and especially with regards to distribution tools.

3.1.2 – C2: security issues related to tool complexity and integration.

The complexity of tools used in a toolchain [18], often consisting of code files dependent on each other and in different file formats, can make their understanding and use difficult. Furthermore, documentation on tools tends to be lacking, especially regarding the security settings of the principle of least privilege, which assigns broad permissions by default that are not ideal for security.

It is therefore difficult to establish the minimum policies to be respected to ensure proper tool functioning. Despite this, organizations may impose the use of a tool even if developers do not have the necessary skills and knowledge to use it, increasing the risk of incorrect tool usage.

Given the complexity of tools and the limited documentation available, it becomes necessary for developers who intend to build a secure system to possess advanced knowledge in different areas to properly configure and integrate tools.

Tool integration is a crucial aspect to ensure proper DevSecOps model functioning. However, developers encounter difficulties in integrating testing tools [19], as the supply of such tools is limited. The lack of automated tools for integrating security tests represents a barrier to DevOps adoption by organizations, as this problem can have a negative impact on system security.

Integrating security tools [15] is a rather difficult activity, as there is no time for manual testing and verification. Therefore, tool integration is a costly, manual, and challenging activity.

Criticality level

Based on the criticality assessment, this difficulty can be considered of low level, as in most cases, organizations still manage to produce secure software in a timely manner. The lack of documentation and complexity does not constitute a serious problem.

Solution

The main issue to address concerns the lack of detailed documentation regarding the proper configuration of the tools, which makes it difficult to set them up correctly. A vulnerability scanning activity, using scripts [20], has identified security issues, where a high percentage of the scripts agree on the presence of specific vulnerabilities, such as the use of weak encryption algorithms and the use of HTTP without TLS. For the latter vulnerability, professionals highlight the poor documentation and support of the tools, stating that this could have been avoided if there had been better documentation and support for the tools. Therefore, it is believed that more detailed documentation and better tool support are necessary to ensure correct configuration, such as adopting HTTP with TLS.

Through documentation, developers will be able to configure security tools correctly and manage these configurations in the best viable way.

However, even if such documentation is available, a practice has not yet been defined that defines how everyone can access it. [21] Good documentation allows the correct security settings and tool configurations to be implemented, but it is essential that it is accessible to all members of the groups.

In terms of document management, it is important to ensure that documentation is traceable and securely and reliably accessible to all project stakeholders, including group members. To this end, it is recommended to avoid sharing data on untraceable means, such as customer chats, and instead adopt a centralized repository to store all necessary information. This way, an adequate level of security

and control can be ensured on access and data management, promoting sharing and collaboration among all project members.

3.1.3 – C3: tool configuration management issues.

In a DevSecOps environment, the management of tool configurations is a critical issue as incorrect configurations can introduce security vulnerabilities in software. Developers [22], who are often negligent, may make software and infrastructure configuration errors, compromising the system.

The adoption of DevOps introduces greater complexity and speed in software release through mechanisms such as Continuous Delivery, therefore, the problem of configuration management becomes even more important. To avoid security issues, it is necessary to ensure that tool configuration is done accurately, and it is also important to ensure software security through mechanisms such as Continuous Security.

In DevOps, developer negligence is evident in default configurations [23], which are often inappropriate and pose a risk to application security.

In the specific case of the cloud, many security solutions rely on security tools with default configurations, leading to unnecessary resource waste and unprotected parts of the application. Therefore, a configuration change is necessary to adapt to the specific security requirements of each application component to ensure system security and efficiency.

Criticality level

Based on the criticality evaluation, this difficulty can be considered high level as the use of default configurations poses a danger to software, as numerous application components could be vulnerable and exposed to attacks. Consequently, it is essential for developers to configure adopted security tools appropriately, avoiding reliance on default settings to ensure the security of the entire application.

Solution

The management of tool configurations represents a difficulty that can be addressed through tool documentation and support [20], which helps avoid the use of default or incorrect configurations. Additionally, it is crucial to adopt best practices for tool use to ensure an adequate level of security.

It is important to note that the use of default configurations [23], can compromise software security; therefore, each tool must be configured according to the context in which it operates.

To address this difficulty, studies have been conducted on the use of ASCAT, i.e., automated static code analysis tools [24], which allow for defining specific project checks, excluding irrelevant ones

(e.g., web checks would be irrelevant in a non-web-based project). Additionally, they allow for excluding irrelevant code from the production environment from tool scanning, configuring tools appropriately based on internal guidelines, and avoiding check duplication.

3.1.4 – C4: limitations of static analysis tools that affect rapid distribution cycles.

Tools belonging to the SAST (Static Application Security Testing) family, such as static analysis tools [25], are problematic as they inspect source code, byte code, and binary code without the software running.

These tools represent a valuable resource for detecting defects, vulnerabilities, and inconsistencies in code early on. However, these tools have limitations due to the complexity of semantic analysis algorithms and the execution times required for code analysis, especially for particularly complex code. For example, the use of pointers or ambiguous code structures can make code analysis difficult and compromise the effectiveness of analysis tools.

In many cases, code static analysis tools may lack understanding of certain code behaviours, which can lead to the generation of false negatives. However, at the same time, such tools may also make incorrect assumptions about code behaviour, causing the generation of false positives. The latter can be attributed to developers' poor knowledge and skills in security [14]. Therefore, it is important for developers to increase their knowledge and skills in security so that they can use these tools correctly and interpret their results because the tools are not able to detect all errors on their own and can generate false positives.

Unfortunately, this situation leads to dissatisfaction among many developers who, instead of deepening and improving their knowledge, tend to completely ignore SAST tools because they require greater knowledge and are difficult to integrate into the DevSecOps environment, which requires speed. The main problem with SAST concerns the time required to detect and scan all false positives that these tools generate, causing delays that are often unacceptable in a system that adopts DevSecOps, where releases are multiple and in short periods. As a result, groups, especially developers, should be able to detect false positives during code writing to avoid wasting time caused by tool usage.

In a DevSecOps context where speed is a critical factor, the use of SAST tools is problematic due to the long time required for code scanning and their high resource consumption [16]. Developers tend to make frequent and small commits, however, if SAST scans were to be performed for each commit,

there would be significant waste of time. In practice, it is not possible to perform complete code scans for every minor change made because this would be inefficient.

Criticality level

Based on the criticality assessment, this difficulty can be considered low-level because SAST tools remain usable and effective in DevSecOps, provided they are used correctly, and false positives and false negatives are managed appropriately.

Solution

Adopting best practices for SAST tool usage [16] is essential to build an optimized pipeline. To solve the problem of code scanning time, for example, in the case of deep scanning, SAST tools could be executed simultaneously without interrupting code control.

Another solution to manage this difficulty is the use of Cloud services [25], which constitute a popular trend in the current context. Customers who use Cloud services can avoid many inconveniences caused by standalone SAST tool usage. This solution is particularly effective because it resolves the issue of the substantial number of false positives, in addition to the difficulties of configuring SAST tools in a DevSecOps environment.

IAST tools [26], also known as Interactive Application Security Testing, represent an alternative to traditional SAST tools. This type of tool is particularly suitable for the software development paradigm adopted in DevSecOps.

IAST combines dynamic testing and static testing, sitting halfway between SAST and DAST. Typically, IAST tools are executed during functional unit testing, which involves critical parts of the code responsible for specific software functionality. However, to use IAST tools, it is necessary to create specific unit tests that can be executed with these tools. Many of these tools, which fit very well in the DevSecOps environment, are available in both commercial and open-source versions.

3.1.5 – C5: limited use of DAST tools in DevSecOps.

In the context of DevSecOps, the use of Dynamic Application Security Testing (DAST) tools [27] and penetration testing tools is frequent. The purpose of these tools is to identify and sometimes exploit security vulnerabilities within a program. They cover a wide range of vulnerabilities, including memory security errors such as buffer overflow and misuse of dynamic memory, as well as issues related to input sanitization, such as SQL injection and command injection. Additionally, developers and security experts often manually use these tools to identify security holes within the program

before malicious hackers exploit them. However, to use these tools, the software must be running. This presents another issue, namely that these tools require the software to be built, installed, and configured together with all necessary software, such as a database or web server [25]. Furthermore, a test suite that is run alongside the software is required, which is a very time-consuming activity.

In a DevSecOps system configuration where code is released frequently, performing all these steps with every release becomes very complex. Another problem that could slow down the release process is caused by the manual configuration and execution of these tools, as well as the need to define test cases manually [27].

Like SAST tools, DAST tools typically require longer execution times [28]. Nevertheless, their use is especially important even within Docker containers, as they help identify malicious behaviour in the container, such as the presence of malicious images or files that were not present before.

All these issues related to DAST tools undermine frequent DevOps releases, although they are very useful for security purposes. Unfortunately, all these disadvantages limit their use in DevSecOps.

Criticality level

Based on the criticality evaluation, this difficulty can be considered low level, since DAST tools, for the same reason as SAST, remain usable and effective in DevSecOps, provided that the correct usage practices are adopted.

Solution

An alternative to DAST tools, which has also been proposed to replace SAST tools, is the use of Interactive Application Security Testing (IAST) tools [26], whose function and benefits have already been examined. If the organization wishes to continue using DAST tools, then in this case, it is necessary to adopt best security practices for the use of such tools, as seen previously with SAST tools.

3.1.6 – C6: inherent security limitations or vulnerabilities that affect the container ecosystem.

In the context of DevSecOps, the use of containers is widely adopted but comes with security and complexity issues [29]. Specifically, the increase in software abstraction levels caused by container usage results in increased complexity, which can generate a greater number of attack vectors.

DevOps provides reusable elements, including container images; however, these images can be compromised and contain vulnerabilities. Despite the encouragement of securing these elements, security is often overlooked.

Containers use third-party elements that typically run on different platforms and are provided by different vendors, raising concerns about the code's integrity. If the code were compromised, vulnerabilities potentially exploitable by malicious hackers would be present.

Inappropriate use of containers can lead to other security implications. The adoption of insecure configurations and inadequate access control settings can lead to security breaches that undermine the integrity of source files located in the container [30].

Finally, other vulnerabilities may arise when containers are launched as virtual machines by developers, for example, through virtualization [31]. This practice exposes well-defined resources to the host system's virtual hardware resources.

Criticality level

Based on the criticality assessment, this difficulty can be considered high-level. In the context of DevSecOps, the use of containers is an essential pillar, so it is vital to ensure their security and implement the best security practices.

Solution

Many studies suggest using orchestration platforms as they can handle container-related issues. These orchestrators can typically correct the improper use of containers [31], particularly Docker, which is widely used in this area. Specifically, containers are poorly isolated.

This isolation state [30] could be improved by introducing elevated levels of abstraction (replication controllers, remote persistent storage, etc.) that eliminate the dependency on the host, achieving high isolation. Additionally, orchestrators elevate automation as a key factor to achieve repeatable, predictable, and continuously improved security.

To ensure the safe distribution of services, the orchestration platform must meet high-security standards to prevent and detect intrusions [32]. Fundamental security features of an orchestration platform may include container image sanitization, reducing the container's attack surface, restricting user access, and strengthening the host.

Another practice that helps address container-related issues is using tools for continuous vulnerability assessment [27]. These tools are considered a key solution in the DevSecOps field, as they enable continuous checks on security vulnerabilities whenever new code is uploaded to a repository. This allows developers to be immediately alerted in the event of newly identified security flaws.

Additionally, the use of multiple DAST and SAST tools for continuous verification of container vulnerabilities [28] provides further benefits, such as mitigation of issues with certain tools, such as false positives.

3.1.7 – C7: vulnerabilities related to Continuous Integration systems.

In the DevOps context, the Continuous Integration (CI) phase is of fundamental importance, however, several studies highlight the higher vulnerability of the tools used in this phase compared to others [33]. This results in a high number of attack vectors associated with such systems. During code build, it is necessary to execute the code provided by developers, which can present several issues, including excessive resource consumption that can cause system malfunctions, or more serious issues such as DoS attacks, viruses, and other types of malicious attacks.

Criticality level

Based on the criticality assessment, this difficulty can be considered of high level. In the context of DevSecOps, the CI phase is crucial; therefore, it is of vital importance that this phase is conducted in a secure manner.

Solution

This solution has been developed to address the problems of attacks on the Continuous Integration (CI) phase or build servers [33]. The use of virtualization tools plays a key role in encapsulating the build process, to create a secure build server. The objective is to maintain the state of the build server uncompromised after each build process, preventing potential attacks.

3.1.8 – C8: limitations of IaC or CaC scripts and tools.

The practice of Configuration as Code (CaC) [34] involves defining network and system configurations through source code that can be managed using version control systems to ensure reproducibility and testing. This technique is widely used to automate the release of new software versions.

Although CaC tools, particularly Puppet, are especially useful in this context, it has been found that developers may have difficulty using them, which can lead to security problems if used improperly.

Infrastructure as Code (IaC) [20] is a technique and an essential element of DevSecOps, often used through tools and scripts such as Chef and Puppet. Many organizations use these IaC scripts to

manage their servers, providing development environments quickly and reliably. However, such scripts are vulnerable to various security issues, including seven common security problems such as default administrators, lack of passwords, integration of sensitive information into code (hard-coded secrets), use of invalid IP bindings, use of suspicious comments by developers, use of HTTP without TLS, and use of poor cryptographic algorithms.

In general, there has been insufficient attention paid to the security of IaC and CaC tools and scripts, despite their widespread use. Consequently, greater attention is required from developers in this area.

Criticality level

Based on the criticality assessment, this difficulty can be considered of medium level. IaC is often used in DevSecOps; therefore, it is crucial to pay more attention to security to avoid cyber-attacks.

Solution

The proposed solution aims to address the security issues associated with IaC scripts and can also be adapted to CaC scripts [35]. Such scripts are often used as input for IaC tools. A static analysis tool, called Security Linter for Infrastructure as Code scripts (SLIC), has been developed to automatically identify security issues in IaC scripts.

3.1.9 – C9: limits and security vulnerabilities affecting the CD pipeline.

One of the relevant issues for the Continuous Delivery (CD) part of software concerns the vulnerabilities and limitations present in the Continuous Delivery Pipeline (CDP). Most CDPs have not been designed to ensure an elevated level of security [36].

Firstly, it would be desirable to allow for the attribution of several types of access to the pipeline based on roles within the development or operations group. For example, a developer should not have the ability to release directly to the production environment without their code changes passing through the pipeline. Some testing and build tasks should be performed only by certain individuals with specific roles.

Secondly, it is essential that the testing and production environments are completely isolated, as system failures often occur when a test environment component is accidentally connected to the production environment database. Finally, in a poorly configured Continuous Delivery Pipeline (CDP), malicious or simply incorrect code may flow into the production environment. Despite the importance of the CDP, it is quite common to design it without considering all security issues.

Most CDP components are executed in an environment that uses online libraries, which are often sources of vulnerabilities [37]. Among the components of the CDP are the repository (usually GitHub), the main server, and the Continuous Integration (CI) server. The repository is at risk of unauthorized access by malicious users who can accept or reject pull requests, make commits, and perform other activities. This can occur due to weak passwords or an internal user with malicious intent. Regarding the main server, a single password is often used for authentication, but it would be advisable to implement additional authentication systems to prevent unauthorized access. Finally, for the CI server, Jenkins is often used, which provides support for project distribution, automation, and compilation. However, when Jenkins is installed, it allows free access to everyone, but it would be advisable to assign roles and limit access only to certain people.

Through a study in which two CD pipelines were used [38], it was found that pipeline security is not a high priority for the development group, even though most members have access to CD pipeline configurations. The lack of security precautions can seriously endanger the business of the organization.

This study showed that both pipelines have vulnerabilities ranging from medium risk to elevated risk. Some vulnerabilities are present because the project team relies on the infrastructure imposed by the client. For both pipelines, the team must trust the security of the external infrastructure and their own group members.

Criticality level

Based on the criticality assessment, this difficulty can be considered high-level. The Continuous Delivery Pipeline (CDP) represents a crucial element within DevSecOps; therefore, it is essential to ensure its security. To this end, it is necessary to promote greater interest among development team members regarding the CDP.

Solution

It is advisable to adopt security best practices to improve the security level of the CD pipeline [37]. Two different CDPs have been studied, in one of which five security practices have been integrated, and it has been demonstrated that the security level of the CDP in question has improved significantly. These practices are: making the repository secure through controlled access, in order to have control over who can perform commits on specific branches of the repository; securing the connection to the main server by using a private key on the Secure Shell (SSH), which is a protocol that allows for a secure and encrypted session via a command-line interface with another host on the network; using roles on the main server to control access through the use of AWS IAM; configuring the CI server to launch a clean state virtual machine, leveraging the Jenkins virtual machine plugin; and finally, using

a Jenkins plugin that allows for role assignment within the CI server. This way, the repository, main server, and CI server are secure.

In addition to these security measures, it is advisable to adopt a threat analysis model [38] that can identify, communicate, understand threats, and provide mitigation methods. In this case, the STRIDE model was employed, which is a framework used for vulnerability assessment in the field of computing. It allows threats to be classified into six distinct categories: spoofing, tampering, non-repudiation, information disclosure, denial-of-service attack, and elevation of privilege.

3.2 – Challenges and solutions related to practices

In this section, we will examine the issues related to continuous practices, such as CI/CD, and security integration practices, such as difficulties in integrating manual security practices within the DevSecOps context.

3.2.1 – C10: difficulties in automating traditional manual security practices.

Automation is a fundamental aspect in DevOps, but its extension in DevSecOps presents some challenges, as many traditional security practices are still manually executed.

The main problem arises when an organization tries to maintain the same speed of production of goods and services, but at the same time adheres to information security standards, frameworks, and best practices. [37]

There are some complex practices to integrate, such as security and privacy by design [39], which is an approach to software and hardware development that seeks to protect systems from the initial stages of development by implementing adequate security measures and risk management methodologies. These practices represent a challenge, as they are not easy to automate and require a significant investment, due to the need for security experts in the development team, coupled with the scarcity of automatic tools that examine risks and assess security in the design and operation phases.

Another complex security practice to integrate is architecture risk analysis [40], which is one of the most powerful software security activities. However, many DevOps projects do not adequately address this practice and fail to provide appropriate software architects to manage security.

Many organizations that adopt DevOps often overlook the importance of software architecture, even though it is essential. Security vulnerabilities in software architecture can cause severe damage, so it is important to perform architecture risk analysis without interfering with the proper functioning of DevOps.

Finally, threat modelling, which is a fundamental security element, also presents integration problems. It is difficult to automate, and the situation becomes more complicated if developers are unable to execute the practice.

Criticality level:

Based on the criticality assessment, this difficulty can be considered of medium level. While automating security practices is fundamental, this issue does not prevent the production of secure software. Alternatively, the organization could choose to perform some practices manually, although this would result in a longer production time.

Solution

A solution is proposed to automate some security practices, such as compliance testing, risk management, security, and privacy by design, by adapting standards, policies, models, and service contracts to testable criteria. For example, service contracts can be used to define a security-by-design methodology that can be integrated into DevSecOps [39].

Service contracts allow for the modelling and evaluation of security requirements, specifying the security capabilities of the application and quantifying the level of security provided. Additionally, they enable the automation of risk analysis techniques and security assessments. The key to implementing this solution is adapting testable criteria. These criteria must be adapted to the various tools and methodologies used in the organization.

Service contracts are an important mechanism, as they enable the assurance of data security and privacy even in multi-cloud environments [41], where applications or services make use of distributed components in different cloud services. This solution allows for the complete management of the cloud application lifecycle, based on risk prioritization, using service contracts.

To perform compliance testing practices at the speed of DevOps [42], developers can use the organization-provided tools to customize information security standards to testable criteria. These tools can perform compliance tests, which enable code validation within an automated release/deployment pipeline, in efficient times. The implementation of automatic and continuous compliance testing mechanisms is fundamental in the DevSecOps approach [22]. In fact, implementing these compliance tests in the initial stages of deployment supports rapid and timely feedback on the compatibility of the ideal environment with the latest version of the software.

3.2.2 – C11: inability to perform a rapid assessment of security requirements.

The adoption of DevSecOps in a project makes it challenging to perform a quick evaluation of security requirements. Due to the speed of automated phases, it is difficult to fully verify security requirements before the software is passed to the production environment. Despite the importance of this practice, it is often not conducted in practice. Once security requirements are evaluated and developed, it becomes necessary to evaluate all security-related objects in a specific environment. However, developers are often reluctant to perform manual tests. The lack of appropriate tools and methods represents the main obstacle to the proper execution of this practice [43]. In fact, most tools focus on testing the vulnerabilities of existing systems, rather than evaluating architecture and design oriented towards security principles and properties.

Criticality level

Based on the criticality evaluation, this difficulty can be considered of medium level. The analysis and evaluation of security requirements are of fundamental importance in this context, particularly when manual tests are not performed. In these cases, the problem becomes critical.

Solution

To overcome the difficulty of fully evaluating security requirements in the DevSecOps context, it is necessary to adopt methods of automating the analysis of security requirements. To this end, specific toolchains have been developed to automatically convert requirement documents into ontological models, analyse them, and provide useful results [43]. Once these results are obtained, developers can intervene on problems from the initial stages of the process.

3.2.3 – C12: difficulties related to security measurement practices in DevOps.

Assessing security in a system is a challenge, particularly when adopting DevOps, as it requires keeping up with the fast and continuous releases of software [44].

It is impossible to anticipate all the possible attacks to which the system may be exposed, as many organizations keep hidden vulnerabilities for years, even when exposed to espionage. Furthermore, measuring software insecurity is a complex task due to the lack of adequate security metrics, which represents the main obstacle to evaluating software security.

In many organizations, traditional data collection methods are still widely used, despite their slowness, which presents a challenge for the adoption of more agile methodologies like DevOps. The

cause lies in the resistance to change of the corporate culture [45]. However, more suitable data collection methods for DevOps must be identified, as traditional techniques such as interviews and surveys do not fit well with the DevOps methodology, which requires continuous and rapid code releases.

Traceability is a crucial property for promptly identifying and subsequently resolving issues. To maintain this property in a system, it is essential to create short and rapid feedback loops [15], which are parts of the system where some portions of the output are used as input for future operations. This helps keep all group members aligned, promoting continuous communication with project stakeholders.

Criticality level

Based on the criticality assessment, this difficulty can be considered of medium level. The assessment of software security is important, particularly it needs to keep up with the pace of DevOps.

Solution

To overcome the limitations of traditional information gathering methods, innovative approaches [45] based on the integration of behavioural analysis techniques and Big Data solutions can be adopted in the DevSecOps context to leverage the advantages of these technologies. Such approaches enable a better understanding of the types of users who use the application, making it easier to obtain quick feedback from end-users. The adoption of these behavioural techniques provides three main benefits to the HCI group: the use of user profiles to monitor user behaviour, predictive analysis of trends based on user behaviour, and the creation of benchmarks to measure different variables.

Due to the speed of software releases in the DevOps paradigm, experts do not prioritize documentation and logging processes, as they are likely to slow down the pipeline. However, these processes are critical to software development and must be executed regardless of the release speed [21].

To easily detect and eliminate anomalies in the process, as well as use test results as evidence in case of verification, it is necessary to document automated distribution and related tests. To achieve this goal, specific tools can be employed for logging release logs and test activities, logging user access to data, and using document repositories. These practices enable the rapid processing of documents and adaptation to DevOps paradigms.

Defining software security metrics is a necessity, with particular attention to metrics that evaluate developer characteristics and their activities [14]. Measuring such metrics is important as teams have constant feedback on what they are doing regarding security. In this way, potential problems in the final stages of software development are mostly prevented.

The metrics are as follows: the number of developers who have undergone security training; the number of security errors found according to the known security classification, such as OWASP top 10; the time spent correcting errors in each category; the number of systems that undergo internal and external penetration testing; the number of commits in each time interval.

3.2.4 – C13: difficulties related to continuous security analysis.

Continuous security analysis is a recommended practice in DevSecOps [46], but the processes related to this practice are not widely adopted. Continuous security seeks to prioritize security as an important aspect throughout the development lifecycle, sometimes even beyond release; as a result, it is a fundamental practice, yet it has a lower priority compared to other aspects.

The practice of continuous vulnerability analysis, while important in DevSecOps [47], is often not adopted due to a lack of regular checks by developers and poor knowledge of the tools used to perform the process.

To properly implement this practice within DevSecOps, clear directives regarding specific sections of the pipeline that must include security measures are needed. Nevertheless, there is a lack of consensus (e.g., standardized methodology) on how security measures should be integrated into the DevOps pipeline [48]. For instance, some DevSecOps definitions emphasize the integration of security practices within a cloud system, while others emphasize the importance of involving developers and managers in concern for production time.

In general, the lack of standardization in integrating security measures in DevSecOps makes it difficult to adopt this practice uniformly and coherently.

Criticality level

Based on the criticality assessment, this difficulty can be considered of medium level. Continuous security practices are recommended in DevSecOps; therefore, organizations should still pay necessary attention even if they are not considered indispensable.

Solution

Continuous vulnerability analysis, which is an important continuous security practice, can be implemented in DevSecOps. Dedicated tools that constantly evaluate the presence of vulnerabilities in software [27] are used for this purpose. These tools enable continuous security vulnerability checks, especially whenever code is added to the main repository, to assist developers in recognizing and mitigating security flaws.

The shift-left security approach is essential for DevSecOps. Integrating security practices and activities from the initial stages is a great advantage, as it allows prioritizing security. In this regard, developers can identify vulnerabilities early on, promoting coverage of costs that would arise in later stages. This approach has a significant impact on resolving the difficulty at hand.

Finally, to support the above, a continuous security assessment practice can be implemented. According to this practice [46], security is transformed from a generic functional requirement to a key concept that must be present in all stages of the development life cycle, including post-release. This is supported by an intelligent and intuitive approach that identifies security vulnerabilities.

To integrate security into all stages of DevOps, a consensus must first be reached on the security practices to be adopted and where to associate them [48], along with a clear security goal to be achieved.

Subsequently, all necessary tools to implement these security practices must be involved in the process [49]. Among the security practices, continuous monitoring is widely used in highly regulated environments [50], where all settings, events, activities, logs, and notifications must be constantly monitored.

Continuous monitoring is also a fundamental practice in IoT systems, as it allows for constant and timely feedback from development and operations groups, promoting the implementation of security practices in DevOps. This practice involves detecting security threats and anomalies in activity logs, enabling developers to resolve issues more easily and quickly.

3.2.5 – C14: incompatibility between security and DevOps practices.

DevOps aims to speed up release cycles, however many security testing practices require human intervention. For example, penetration testing [44] is an activity that requires someone to configure, monitor and execute the testing tools. The complexity of these practices requires significant resources; therefore, the adoption of rapid releases could hinder the execution of thorough testing programs.

Furthermore, increasing complexity, vulnerabilities and dependencies on third-party components have made security assurance even more difficult [15]. IIn general, the use of third-party components, although advantageous, can introduce new vulnerabilities into the application. Therefore, it is necessary to ensure the security of all external components. Developers often face trade-offs between release speed and security integration [51], which can be identified as: assurance; security; flexibility. Assurance refers to the trade-off between the increased added value that additional tests provide and

the cost required to perform such tests; security refers to the trade-off between the increase in security measures and the ability to access and modify the CI system; finally, flexibility describes the trade-off between the need for powerful and highly configurable systems, and the need for simple and easy-to-use systems.

Unfortunately, many organizations are reluctant to transform their development and operations processes into DevSecOps [21] because they anticipate many incompatibilities between security and DevOps. However, these incompatibilities are resolved when the correct practices are adopted.

Criticality level

Based on the criticality assessment, this difficulty can be considered high-level. It is crucial that an organization be aware of the integration challenges of practices before adopting DevSecOps, otherwise, the imposition of the methodology would be unsuccessful.

Solution

The incompatibilities between security and DevOps can be resolved through the adoption of proper practices, with a particular emphasis on the shift-left security approach. To support the DevSecOps paradigm, continuous security assessment practices can be implemented, such as continuous monitoring.

To reduce incompatibilities between security and DevOps, security patches managed by DevOps practices can be utilized [44]. A crucial aspect to consider with this methodology is speed; software releases and development phases are rapid, resulting in the possibility of vulnerabilities in the production environment where the software is available for use.

Despite the unintentional presence of vulnerabilities, it is essential to address them as soon as possible through the application of security patches, which can be distributed quickly using DevOps practices.

To promote the adoption of DevSecOps, it is important to facilitate communication and collaboration within and between groups. It is incredibly useful to establish a multidisciplinary group [42] within the organization, comprising members with separate roles and characteristics but who share the same goal. Moreover, it is important to spread a passion for security among developers and involve them actively in security activities, including incident management tasks. This collaboration, in turn, enables the integration of security practices into every aspect of the continuous deployment pipeline and helps reduce vulnerability problems before software is released.

The security group can contribute to and improve collaboration with other groups by utilizing existing DevOps automation activities within the organization [17], particularly by modifying existing security tools to ensure a short feedback cycle between the security group and others.

Additionally, it is crucial to spread a passion for security among developers within the organization [52], as it would facilitate rapid resolution of problems in case of security incidents. However, it is necessary for such developers to participate in security activities to have tasks to perform, and it is essential to include them in the incident management lifecycle.

To ensure greater collaboration, appropriate controls and standards must be adopted [21]. For example, a clear separation of tasks among members of various groups should be established, and clear communication among members of diverse groups is important.

Manual communication methods such as email should not be abused as they may cause even more waste of time. Instead, automatic methods are ideal for informing group members of activities being performed or in progress in processes. For instance, stakeholders should be automatically notified by a system instead of manually by an administrator.

In highly regulated environments, it is necessary to continuously communicate risk information, assessment results, and logical process information to stakeholders [53]. To this end, it is crucial for all stakeholders to have a profile that enables their identification, and communication channels should be designed for the dissemination of such information.

3.3 – Challenges and solutions related to infrastructures

This section addresses the challenges of adopting DevSecOps in diverse types of complex infrastructures.

3.3.1 – C15: difficulties in adopting DevSecOps in highly regulated environments.

In highly regulated environments, such as the pharmaceutical industry, adopting continuous security practices like DevSecOps can be a complex challenge, as it may be difficult to integrate the continuous deployment of security patches for medical devices [54]. The increase in cyber-attacks in this industry has led to an increased demand for secure medical products, and there are laws and regulations requiring manufacturers to design secure products and accurately assess risks. Therefore, to continuously deploy security patches, manage vulnerabilities transparently, and integrate security into the software development life cycle, a cybersecurity strategy that includes integrated security controls and secure product distribution is necessary.

In this context, integrating DevSecOps continuous security practices in isolated production environments is challenging, as highlighted by [55]. This is due to the lack of appropriate DevOps solutions for such infrastructures, which makes it difficult to integrate security at all stages of the software development life cycle. Zero-trust security architectures, which characterize these environments, include separate environments, temporary access policies, and restricted communications with stakeholders, which pose a significant challenge for implementing DevSecOps security practices. Consequently, adopting continuous security measures in highly regulated or isolated environments is complex.

Criticality level

Based on the criticality assessment, this difficulty can be considered high-level. The described difficulty is relevant only for some organizations, namely those adopting highly regulated or isolated infrastructures. However, for companies intending to adopt DevSecOps on such infrastructures, it is essential to consider this problem.

Solution

In highly regulated environments, the adoption of DevSecOps can be complex but feasible using proper security practices. An effective partial solution is the implementation of continuous security assessment practices, particularly continuous monitoring, which is widely used in these environments.

In addition, an effective way to incentivize the adoption of DevSecOps in such environments is to facilitate collaboration and communication within and between groups, especially in highly regulated contexts where this is particularly important.

To ensure security in shared work pipelines, it is necessary to assign roles and configure access permissions, especially in highly regulated environments [55]. Limited access management policies are crucial in these cases, such as assigning minimal privileges based on the needs of individual developers who access highly regulated environments. Additionally, it is a good practice to automate changes in the production environment; therefore, it may be a clever idea to remove developer access to the production environment.

It is highly advisable to use Infrastructure as Code (IaC) in complex and highly regulated infrastructures. The use of IaC [53] allows for the infrastructure to be version-controlled, tested, built, and reliably deployed, often representing a preferred solution in complex environments with rigorous security requirements, such as physically isolated and secure environments. A development group can create a centralized automatic service that provides and distributes systems and software. Additionally, at the project's outset, it is essential for stakeholders to agree on the systems and

environments needed for project compilation and testing to resolve the software distribution problem through a limited number of centralized processes, which is a common challenge in highly regulated environments.

In highly regulated environments, it is crucial that vulnerability management activities are systematic and transparent to ensure the safety of products, such as medical devices [54]. It is important to design a risk-aware vulnerability assessment for the industry, such as designing a system with highly regulated configurations given the complexity of software, which often includes open-source components. Therefore, it is necessary to systematically evaluate the vulnerabilities of the components used, even in complex environments such as highly regulated ones.

Additionally, it is recommended to create simulation environments in these complex contexts, where developers can conduct testing practices using the simulation system as a temporary environment [53]. Subsequently, they can report the results to various stakeholders, and end-users can evaluate distributed versions even in simulated environments, providing additional feedback, even in isolated infrastructures, if necessary. Finally, to ensure system security, it is advisable to build limited access management policies, assigning roles and configuring access permissions to limit access based on needs, especially for developers who need to access the configurations of such environments.

The ENACT framework [56] for reliable IoT systems contains testing and simulation tools at one of its levels. These tools allow for the creation of test scenarios for applications based on programmed conditions with the aim of providing indications of performance, test resistance, and risk management. These tools aim to provide guidelines for evaluating the system's ability to address security issues.

3.3.2 – C16: difficulties in adopting DevSecOps in environments with limited resources.

Typically, when discussing environments with limited resources, two types of infrastructures are often referred to: Internet of Things (IoT) systems and embedded systems. IoT is defined as a network of objects, including sensors and other technologies that exchange information with other devices on the network. These objects can include household or industrial devices [57]. However, adopting DevSecOps on such a heterogeneous infrastructure is a challenge, as there is an elevated risk of cyber-attacks, and it is overly complex to maintain and update the infrastructure over time.

The complexity in implementing secure distribution pipelines and monitoring security events in IoT systems, due to their heterogeneous and distributed nature, represents a significant challenge [50]. This can hinder rapid development and make it difficult to operate and monitor production systems,

including cybersecurity events. Additionally, the infrastructure must be highly flexible to allow groups to configure and monitor security based on their specific criteria.

There are not many approaches and tools designed for IoT systems that adopt DevSecOps [56], due to the prevalence of Cloud systems that have more tools available.

Embedded systems represent another complex infrastructure to adopt DevSecOps [58]. These are made up of microprocessor-based electronic systems, designed for specific use and not reprogrammable by users. However, continuous integration (CI) in the software developed for these systems is difficult, due to the importance of compliance with standards, which does not focus on the speed of software release. Additionally, restricting access to information is a security-related issue in this sector. In summary, both infrastructures have difficulty adopting DevSecOps and some security practices.

Criticality level

Based on the criticality assessment, this difficulty can be considered high-level. The described difficulty is only relevant to some organizations, namely those that adopt IoT or embedded systems. However, for companies that intend to adopt DevSecOps on such infrastructures, it is essential to consider this problem.

Solution

Complex infrastructures such as IoT and embedded systems require continuous security assessment practices, while it is recommended to adopt IaC to manage this complexity. Furthermore, it is of fundamental importance to have simulation environments for testing activities.

The adoption of DevSecOps in IoT infrastructures can be supported by guided models [57]. Such models should be able to address the specific challenges of IoT infrastructures, such as heterogeneity, and support the definition of security and privacy requirements, as well as the automatic distribution and management of security features.

GENESIS is an engineering model that adopts this approach to support DevSecOps in the context of IoT. Additionally, the use of a suitable framework is fundamental for the development of intelligent IoT systems, which involve sensors and actuators and require elevated levels of security, privacy, resilience, reliability, and protection. ENACT [56], is a DevOps framework that offers an approach to adapting DevOps techniques to the specific needs of intelligent IoT systems.

3.3.3 – C17: difficulties in adopting DevSecOps in complex Cloud environments.

The principles and practices of DevSecOps are challenging to adopt in complex cloud environments. For instance, the rapid production of secure software is difficult in a cloud environment where the system is in a system-of-systems (SoS) form [59]. Such a system includes other systems (the constituent systems) that have managerial and operational independence. Integrating cloud applications and services within such a complex system entails several difficulties, especially concerning security. Therefore, it is crucial to define an adequate framework that plans and defines security requirements phases for an SoS using the DevSecOps approach.

Multi-cloud environments, which combine multiple and heterogeneous cloud services, represent another complex infrastructure [41]. Ensuring security in multi-cloud systems is not easy because it is necessary to control not only the components of a single system and the risks, they are exposed to but also those of cloud providers. Security, privacy, and data protection continue to represent the major barriers to the adoption of cloud infrastructures, especially when adopting the DevSecOps approach.

In multi-cloud environments, the use of microservices is common [60]. These can be defined as a collection of independent services that communicate with each other through simple internal communication mechanisms. Breaking down an application into microservices allows for rapid development and deployment of cloud applications. However, the intercommunication between these services exposes the system to security risks such as information espionage and other potential attacks. Therefore, adopting DevSecOps in this context is complex.

Another fundamental aspect concerning security in cloud infrastructures is data security [61]. Rapidly producing software while ensuring data security represents an overly complex task. Nonetheless, it is crucial that security is ensured in these environments.

Data security is essential not only during the execution of the cloud application but also during all other phases of the software lifecycle. Therefore, it is crucial that data are protected both from remote and local attackers, both when the system is running and when it is turned off.

Criticality level

Based on the criticality assessment, this difficulty can be considered of elevated level. The difficulty described is relevant to many organizations, i.e., those that adopt Cloud since it is the most widespread infrastructure. Therefore, companies intending to adopt DevSecOps on such infrastructures must take this problem into account.

Solution

Cloud environments can be overly complex, and as such, it is recommended to use Infrastructure as Code (IaC). Additionally, for the DevSecOps paradigm, it is essential to have facilitated communication and collaboration within and between groups.

An important aspect of cloud solutions is data security. This aspect is complex, as ensuring data security can be challenging, particularly in a methodology that aims to release software versions rapidly. To address this issue, it is necessary to focus on data security in parallel with the development cycle [61]. To this end, the Software as a Service (SaaS) security life cycle (SSLC) has been developed, which combines data security and software development lifecycles. This solution includes requirements gathering and analysis, design, implementation, testing, distribution, and maintenance.

Finally, it is also ideal to adopt appropriate DevSecOps frameworks for cloud environments. Several frameworks have been developed for cloud environments, as they are the most widely used infrastructure for the methodology.

For cloud environments that use System of Systems (SoS), a security-focused framework has been developed [59], namely SoS governance. This framework supports the integration of security practices, such as secure engineering, separation of duties, secure deployment and operations, availability and business continuity, training, and review.

The MUSA framework has been developed [62] to ensure security and privacy in multi-cloud systems, unifying the preventive and reactive approaches to security. In terms of prevention, the framework supports Security by Design practices during application development, as well as the integration of required security mechanisms in the application itself. Regarding reactive security, MUSA adopts monitoring practices to detect, notify, and mitigate security incidents, so that multi-cloud application providers are informed and ready to respond quickly to attacks and security issues.

Another security framework utilizes network function virtualization (NFV) [63] functions and microservices design patterns. This framework is used for heterogeneous and distributed cloud environments.

Finally, the Unicorn Framework is proposed [61], which aims to address the challenges related to the design of scalable and secure applications based on microservices architectures in multi-cloud environments. The framework provides a complete set of tools that help groups in developing, designing, and managing scalable and secure containerized applications in multi-cloud environments.

3.4 – Challenges and solutions related to personnel and organizational culture

This section addresses the difficulties related to knowledge, skills, and collaboration among members of different DevSecOps groups, as well as the challenges related to the organization's culture.

3.4.1 – C18: collaboration problems between groups.

Effective communication and collaboration among group members is a critical success factor in DevSecOps. However, proper collaboration among groups is not always practiced, and conflicts often arise between the development and security groups [14].

Typically, programmers spend a lot of time writing code that is evaluated by security professionals. This can lead to a loss of autonomy for developers, who may feel inferior and unable to make decisions about the code they have written, instead having to conform to the directives and advice of security professionals. In addition, the opinions of developers, even if competent in security matters, may be considered less valuable than those of the experts on the security team.

Historically, software engineering and software security have worked separately. These separations prevent communications and collaborations among stakeholders, including members of the security group. Other groups must communicate and collaborate frequently with members of the security group, and it is essential to establish shared responsibility, but this often does not happen.

Tools that facilitate collaboration with the security group [64] are necessary. In the early stages of the development process, the use of tools could be advantageous for managing and providing design models, allowing for collaborative understanding of the team's security or other group's design motivation.

Criticality level

Based on the assessment of criticality, this difficulty can be considered of medium level. Collaboration and communication within and between groups are important, but any divergences in the organization are unlikely to compromise software production. Nonetheless, the company must ensure a healthy and collaborative environment.

Solution

To improve collaboration and communication among groups within DevSecOps, it is necessary to have a "security champion" role that acts as a bridge between the security group and the others [14].

Members in this role are typically programmers who have received the best security training and are responsible for facilitating communication between programmers and other groups. They are the first point of contact when a security issue is identified externally, and they communicate with other members to ensure that security is integrated at all stages of the process.

Ideally, security champions should be integrated into the development group [44]. This can facilitate the acceptance of their recommendations and suggestions by developers, as security champions are part of the same group and have greater knowledge of security than developers.

To support the role of the security champion, appropriate norms and standards can be adopted to improve communication and collaboration between groups.

In addition, a cloud service such as CAIRIS [64] can be used to allow security experts and other groups to work together on design models, promoting internal collaboration between groups.

Finally, the fundamental approach of DevSecOps, namely shift-left security, is immensely helpful in improving internal collaboration and collaboration between groups, as members of the groups must collaborate and communicate from the initial stages to integrate security.

3.4.2 – C19: security knowledge differences.

It has been shown that DevSecOps supports developers in addressing security tasks; however, not all developers have the same security skills and knowledge, and often their knowledge and skills are not sufficient to perform certain tasks. In particular, the lack of training, education, and security skills among developers poses a significant challenge for the adoption of DevSecOps [14].

In the context of DevSecOps, the main problem is the separation between software engineering and software security education. Developers are typically not adequately trained in security, which leaves them ill-prepared in the field.

According to another theory [18], DevOps security is complex and requires highly skilled personnel to counter attacks. As DevOps practices are widespread in an organization, it may be difficult to find personnel with the required skills to manage all these practices. Therefore, it would be beneficial to simplify the security aspects of DevOps, making it easier for less specialized developers to work with.

Criticality level

Based on the assessment of criticality, this difficulty can be considered low-level. In an organization that adopts DevSecOps, knowledge differences among developers are not a significant problem, as it is common for group members to improve their skills over time and through practical experience.

Solution

Implementing security training and shared knowledge methods could improve the security skills of development team members, enabling them to perform important tasks. For example, a good understanding of security issues is essential for selecting and using appropriate tools [14]. If the development team uses static code analysis tools (SAST) to identify vulnerabilities, they must be aware that these tools can generate false positives, and it is up to team members to identify them.

However, it is not desirable to provide specialized security training to all developers [44]. Nevertheless, it is possible to train each of them sufficiently so that they can identify areas where expert support is required. It is believed that if developers acquired sufficient knowledge of software security, the number of security anomalies could decrease by 50%.

An effective strategy for mitigating security vulnerabilities is to offer security training [17], aimed at preparing developers to integrate security within DevOps organizations. This training process could include various activities, such as completing relevant online courses, participating in developer boot camps, and organizing company meetings.

Regarding the use of shared knowledge methods [14], it is common to conduct a retrospective, a learning review, to evaluate the results. It is important to note that the retrospective is blameless, meaning that everyone did their best based on available resources, their abilities, and the circumstances they were in, regardless of the results obtained.

3.4.3 – C20: difficulties in organizational culture.

To successfully adopt DevSecOps, as previously highlighted, it is necessary to make behavioural changes and promote a security culture. However, this process can lead to competition among group members [59], and fear of being replaced. On the contrary, this mechanism should represent an opportunity to gain experience and expand one's knowledge in security, without losing one's position. It is necessary for group members to collaborate and behave properly within their own group, as well as with other groups.

An even more significant problem that hinders the adoption of DevSecOps is reluctance to prioritize security within the organization [14]. Many developers and other group members do not want to take on security-related responsibilities, as they believe that the required effort is excessive, and the associated risk is too high.

Criticality level

Based on the assessment of criticality, this difficulty can be considered of medium level. In an organization that adopts DevSecOps, difficulties in organizational culture represent a significant challenge, since a security-oriented culture is fundamental to adopting such methodology.

Solution

One solution to facilitate the adoption of DevSecOps, leading to a security-oriented culture, consists of conducting security training and shared knowledge methods, whose benefits have already been examined in this section.

The cultural change required to adopt DevSecOps can be challenging for some people in the organization. To address this issue, it is recommended to conduct human resource management (HRM) programs in parallel with DevSecOps project transformations [59].

Security training programs should aim to address common feelings among group members, such as fear of losing control of their work or being replaced. It is essential that the environment is conducive to change, and DevSecOps should be perceived as an opportunity to improve one's skills.

3.4.4 – C21: insider threats.

In the context of DevOps, the role of the developer has changed and often involves access to production services. This expansion of permissions has increased the risk of insider threats, meaning users with access to the production environment who can use tools to damage the system or even take control of it [65].

Unrestricted collaboration between groups can lead to security issues such as inappropriate access to system resources [17]. In other words, when groups collaborate too closely, individual members may abuse their access power to resources, and accidentally or intentionally change system properties.

Criticality level

Based on the criticality assessment, this difficulty can be considered high. It is of fundamental importance for organizations adopting DevSecOps to consider this issue because the abuse of access permissions to system resources can cause serious security breaches.

Solution

In a perspective of mitigating internal threats, a framework for protecting data and system integrity has been proposed [65]. A set of integrity protection requirements was developed based on potential threats, and subsequently, a framework was developed targeting global integrity protection within microservices-based systems.

An example of a measure taken to counter internal threats is the use of a secure voting component, where the most dependable members of groups are called upon to vote whether to accept frequent changes made to artifacts (i.e., scripts and configuration files). This way, even if malicious insiders make such changes, they can still be controlled and discarded.

3.5 – Results

The analysis of challenges and solutions provides a comprehensive overview of the methodology. It has been highlighted that the most significant challenges in implementing the DevSecOps methodology concern the tools, which amount to a total of nine. In addition, five issues related to practices to be integrated, four challenges related to personnel and organizational culture, and three challenges concerning infrastructure have been identified. Each challenge identified during the analysis can have a negative impact on the correct use of the DevSecOps methodology. However, it is important to classify the difficulties according to their level of criticality and focus mainly on the most relevant ones. Therefore, for an organization that wants to adopt the methodology, resolving most or all the identified issues is of paramount importance. The proposed solutions are effective in achieving this goal. The difficulties, along with their level of criticality, and the corresponding solutions are presented in Table 2.

Challenge Numbers	Criticality levels	Challenges	Solutions
Security tools			
C1	medium	Difficulties in selecting tools.	- Standardization of security tools.
C2	low	Security issues related to tool complexity and integration	- Security-supported documentation.
C3	high	Tool configuration management issues.	- Security-supported documentation. - Use of best practices for the security tool's usage.
C4	low	Limitations of static analysis tools that affect rapid distribution cycles.	- Use of best practices for the security tool's usage. - Use of IAST tools. - Moving towards cloud solutions.
C5	low	Limited use of DAST tools in DevSecOps.	- Use of IAST tools. - Use of best practices for the security tool's usage.
C6	high	Inherent security limitations or vulnerabilities that affect the container ecosystem.	- Use of orchestration platforms. - Adoption of tools for continuous vulnerability assessment.
C7	high	Vulnerabilities related to *Continuous Integration* systems.	- Using a virtualization tool to encapsulate a part of the system.
C8	medium	Limitations of IaC or CaC scripts and tools.	- Static analysis of IaC scripts.
C9	high	Limits and security vulnerabilities affecting the CD pipeline	- Reusable design fragments. - Use of security threat analysis practices.

Practices			
C10	medium	Difficulties in automating traditional manual security practices.	- Adapting standards, policies, models, and service contracts based on testable criteria.
C11	medium	Inability to perform a rapid assessment of security requirements.	- Automated detection of vulnerabilities through requirements analysis.
C12	medium	Difficulties related to security measurement practices in DevOps.	- Use of Big Data and behavioral analysis techniques. - Effective processing of documentation and recording strategies. - Definition of security metrics.
C13	medium	Difficulties related to continuous security analysis.	- Adoption of tools for continuous vulnerability assessment. - Adopt the security *shift-left* approach. - Implement continuous security assessment practices.
C14	high	Incompatibility between security and DevOps practices.	- Adopt the security *shift-left* approach. - Management of security patches. - Implement continuous security assessment practices. - Facilitate communication and collaboration within and between groups.

Infrastructures			
C15	high	Difficulties in adopting DevSecOps in highly regulated environments.	- Implement continuous security assessment practices. - Facilitate communication and collaboration within and between groups. - Implement limited access management policies. - Adopt IaC. - Evaluate product-specific vulnerabilities. - Create duplicate or simulated environments for testing purposes.
C16	high	Difficulties in adopting DevSecOps in environments with limited resources	- Implement continuous security assessment practices. - Adopt IaC. - Model-driven engineering to support DevSecOps. - Create duplicate or simulated environments for testing purposes. - Adopt frameworks that support DevSecOps.
C17	high	Difficulties in adopting DevSecOps in complex Cloud environments	- Adopt IaC. - Facilitate communication and collaboration within and between groups. - Adopt frameworks that support DevSecOps. - Hybrid life cycles aimed at data security.

Personnel and organizational culture			
C18	medium	Collaboration problems between groups.	- Having security champions in the groups. - Facilitate communication and collaboration within and between groups. - Adopt the security *shift-left* approach. - Moving towards cloud solutions.
C19	low	Security knowledge differences.	- Implement security training and shared knowledge methods.
C20	medium	Difficulties in organizational culture.	- Implement security training and shared knowledge methods. - Carry out human resource management (HRM) programs in parallel.
C21	high	Insider threats.	- Integrity protection framework.

Table 2 : summary of challenges and solutions associated.

The analysis revealed that out of a total of twenty-one identified challenges, nine of them have a high level of criticality, eight have a medium level of criticality, and four have a low level of criticality. It is important to note that difficulties related to infrastructure are considered overly critical only if the organization uses such infrastructure. The distribution of criticality levels is depicted in Chart 1.

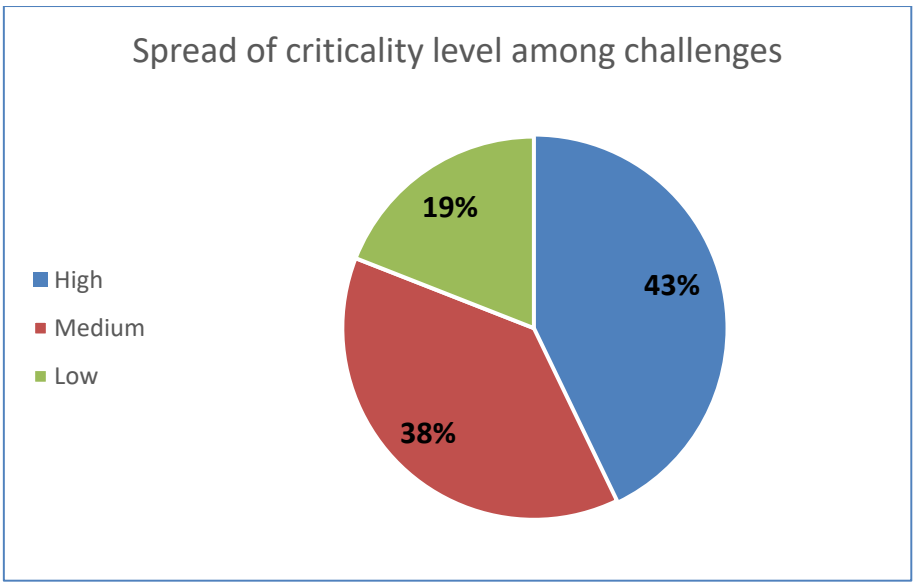

Chart 1: DevSecOps challenges – criticality level

3.6 – Research and improvements

Previously, all issues that may arise during the adoption of DevSecOps methodology were categorized and examined, and at least one solution and a level of criticality were associated with each of them. The analyses conducted revealed that DevSecOps represents the ideal approach for organizations that aim to frequently release secure software, despite the complexity of the methodology itself. Therefore, by following all or most of the proposed solutions, any interested organization can adopt the methodology appropriately. However, it is important to emphasize that there are still areas in the methodology that require further research. The following are all these areas:

1) Further research is needed on application security tools and methods that are specifically designed for DevSecOps: In the proposed solutions, SAST and DAST tools were discussed, which are often inadequate since they significantly slow down software releases. In this regard, a compromise has been found with IAST tools, which exist and are available, but further research on appropriate tools would be optimal.

2) More research is needed on how other application security testing methods can be provided as services (e.g., DAST): Only one study shows that if SAST tools are used and provided as services, they can solve the problem of slowdown.

3) Security tools focused on developers need to be developed: Most of the tools available to organizations are complicated for developers, who are often discouraged from using them.

4) Developers often do not know how to strike a balance between security and the speed of DevOps: When vulnerabilities are identified, it is necessary to strike the right balance to act without excessively slowing down DevOps processes, but without neglecting security.

5) More research is needed on consents for the shift-left security approach: There are only a few studies that offer solutions on how to adopt this approach and how to automate traditional manual practices. Therefore, more research is needed on consents and practices.

6) More research is needed on security metrics: There are few studies that have established security metrics adaptable to DevSecOps.

7) Clarify roles in DevSecOps: The decision-making roles of developers who deal with security tasks are not well defined and clear.

8) More research is needed on personnel difficulties in adopting DevSecOps: Personnel difficulties are few compared to those related to tools. Therefore, further socio-technical studies focusing on these difficulties and how they impact DevSecOps are needed.

9) Empirical validation of frameworks: Most of the proposed frameworks are case studies or solutions proposed by the same company that needed them. Therefore, more studies are needed to understand the applicability of these solutions, especially in different contexts.

Chapter 4 – Conclusions

From the analysis of difficulties, solutions and obtained results, it can be concluded that DevSecOps is a valid methodology for all organizations that want to produce secure and fast software. Security tools represent a fundamental element for the success of DevSecOps. While several issues have been identified, all of them can be effectively managed through tool standardization, adoption of IAST tools, provision of tool documentation, use of cloud solutions, and IaC tools.

Regarding security practices, it has been observed how difficult it can be to implement them in this methodology. However, they can also be managed effectively using security metrics, analysis of security requirements, facilitation of collaboration and communication within and between groups, and the use of the security shift-left approach.

Another examined topic is infrastructure, specifically the difficulty of adopting DevSecOps methodology for some infrastructures such as complex cloud environments (multi-cloud, SoS, etc.), highly regulated environments, and environments with limited resources. Nevertheless, solutions exist for each of these infrastructures, particularly the need for a suitable framework, facilitation of collaboration and communication within and between groups, and the use of IaC.

Finally, an aspect of fundamental importance for the DevSecOps methodology, concerning personnel and organizational culture, was examined. Difficulties in collaboration among team members, differences in security knowledge, problems in implementing a security-oriented culture, and possible internal threats were taken into consideration. However, these challenges can also be addressed through the adoption of solutions that improve personnel security skills, promote effective communication among team members, reduce differences in security knowledge, manage internal threats, and promote cultural change towards a security-focused approach.

DevSecOps represents a complex methodology that requires further research in some areas. However, given the increase in cyber-attacks over the last year, it is becoming increasingly important for organizations to integrate the concept of security, which is often overlooked or addressed superficially. An organization that produces secure and fast software has a competitive advantage in the market and ensures the security of end users as well as the company itself, reducing exposure to possible cyber-attacks.

References

[1] Jabbari, R., Ali, N. bin, Petersen, K., & Tanveer, B. (2016). What is DevOps? A systematic mapping study on definitions and practices. *ACM International Conference Proceeding Series*, *24-May-2016*.

[2] Zaheeruddin Ahmed, & Shoba. C. Francis. (2019). *Integrating Security with DevSecOps: Techniques and Challenges*. Proceeding of 2019 International Conference on Digitization: Landscaping Artificial Intelligence.

[3] Török, M., & Pataki, N. (2020). *DevOps Dashboard with Heatmap*. Proceedings of the 11th International Conference on Applied Informatics, Eger, Hungary, January 29–31, 2020.

[4] Stoyanova, M. (2019). SMART CONCEPT FOR PROJECT MANAGEMENT – TRANSITION TO DevOps. In *KNOWLEDGE-International Journal* (Vol. 34).

[5] Sánchez-Gordón, M., & Colomo-Palacios, R. (2020). *Security as Culture A Systematic Literature Review of DevSecOps*.

[6] Myrbakken, H., & Colomo-Palacios, R. (2017). DevSecOps: A multivocal literature review. *Communications in Computer and Information Science*, *770*, 17–29.

[7] Nandgaonkar, S., & Khatavkar, V. (2022). CI-CD Pipeline For Content Releases. *2022 IEEE 3rd Global Conference for Advancement in Technology, GCAT 2022*.

[8] Sethi, F. (2020). Automating Software Code Deployment Using Continuous Integration and Continuous Delivery Pipeline for Business Intelligence Solution. *International Journal of Innovation Scientific Research and Review*, *02*(10), 445–449.

[9] Dupont, S., Ginis, G., Malacario, M., Porretti, C., Maunero, N., Ponsard, C., & Massonet, P. (2021). Incremental Common Criteria Certification Processes using DevSecOps Practices. *Proceedings - 2021 IEEE European Symposium on Security and Privacy Workshops, Euro S and PW 2021*, 12–23.

[10] Yang, Y., Shen, W., Ruan, B., Liu, W., & Ren, K. (2021). Security Challenges in the Container Cloud. *Proceedings - 2021 3rd IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications, TPS-ISA 2021*, 137–145.

[11] Chen, S. J., Pan, Y. C., Ma, Y. W., & Chiang, C. M. (2022). The Impact of the Practical Security Test during the Software Development Lifecycle. *International Conference on Advanced Communication Technology, ICACT*, *2022-February* 313–316.

[12] Jiang, Y., & Adams, B. (2015). Co-evolution of infrastructure and source code - An empirical study. *IEEE International Working Conference on Mining Software Repositories*, *2015-August*, 45–55.

[13] Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. In *Information and Software Technology* (Vol. 141). Elsevier B.V.

[14] Tomas, N., Li, J., & Huang, H. (2019, June 1). An empirical study on culture, automation, measurement, and sharing of DevSecOps. *2019 International Conference on Cyber Security and Protection of Digital Services, Cyber Security 2019*.

[15] Yasar, H., & Kontostathis, K. (2017). Where to Integrate Security Practices on DevOps Platform. *International Journal of Secure Software Engineering*, *7*(4), 39–50.

[16] Soenen, T., van Rossem, S., Tavernier, et al. (2018). Insights from SONATA: Implementing and integrating a microservice-based NFV service platform with a DevOps methodology. *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, 1–6.

[17] Rahman, A. A. U., & Williams, L. (2016). Software security in DevOps: Synthesizing practitioners' perceptions and practices. *Proceedings - International Workshop on Continuous Software Evolution and Delivery, CSED 2016*, 70–76.

[18] Wilde, N., Eddy, B., Patel, K., Cooper, N., Gamboa, V., Mishra, B., & Shah, K. (2016). Security for Devops Deployment Processes: Defenses, Risks, Research Directions. *International Journal of Software Engineering & Applications*, *7*(6), 01–16.

[19] Rafi, S., Yu, W., & Akbar, M. A. (2020). Towards a Hypothetical Framework to Secure DevOps Adoption: Grounded Theory Approach. *ACM International Conference Proceeding Series*, 457–462.

[20] Rahman, A., Parnin, C., & Williams, L. (2019). The Seven Sins: Security Smells in Infrastructure as Code Scripts. *Proceedings - International Conference on Software Engineering*, *2019-May*, 164–175.

[21] Mohan, V., ben Othmane, L., & Kres, A. (2018). BP: Security concerns and best practices for automation of software deployment processes: An industrial case study. *Proceedings - 2018 IEEE Cybersecurity Development Conference, SecDev 2018*, 21–28.

[22] Steffens, A., Lichter, H., & Moscher, M. (n.d.). *Towards Data-driven Continuous Compliance Testing*. (2018). CSE 2018: 3rd Workshop on Continuous Software Engineering.

[23] Kritikos, K., Papoutsakis, M., Ioannidis, S., & Magoutis, K. (2019). Towards configurable cloud application security. *Proceedings - 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2019*, 684–689.

[24] Zampetti, F., Scalabrino, S., Oliveto, R., Canfora, G., & di Penta, M. (2017). How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines. *IEEE International Working Conference on Mining Software Repositories*, 334–344.

[25] Kupsch, J. A., Miller, B. P., Basupalli, V., & Burger, J. (n.d.). *From Continuous Integration to Continuous Assurance*. 2017 IEEE 28th Annual Software Technology Conference (STC).

[26] Siewruk, G., Mazurczyk, W., & Karpiński, A. (2019). Security assurance in Devops methodologies and related environments. *International Journal of Electronics and Telecommunications*, *65*(2), 211–216.

[27] Lescisin, M., Mahmoud, Q. H., & Cioraca, A. (2019). Design and implementation of SFCI: A tool for security focused continuous integration. *Computers*, *8*(4).

[28] Brady, K., Moon, S., Nguyen, T., & Coffman, J. (2020). Docker Container Security in Cloud Computing. *2020 10th Annual Computing and Communication Workshop and Conference, CCWC 2020*, 975–980.

[29] Bjørgeengen, J.(2019). A Multitenant Container Platform with OKD, Harbor Registry and ELK. ISC High Performance 2019: High Performance Computing pp 69–79.

[30] Combe, T., Martin, A., & di Pietro, R. (2016). To Docker or Not to Docker: A Security Perspective. *IEEE Cloud Computing*, *3*(5), 54–62.

[31] Martin, A., Raponi, S., Combe, T., & di Pietro, R. (2018). Docker ecosystem – Vulnerability Analysis. *Computer Communications*, *122*, 30–43.

[32] Khan, A. (2017). *Key Characteristics of a Container Orchestration Platform to Enable a Modern Application*. IEEE Cloud Computing Vol.4 Issue 5.

[33] Gruhn, V., Hannebauer, C., & John, C. (2013). *Security of Public Continuous Integration Services*. WikiSym '13: Proceedings of the 9th International Symposium on Open Collaboration.

[34] Rahman, A., Partho, A., Morrison, P., & Williams, L. (2018). What questions do programmers ask about configuration as code? *Proceedings - International Conference on Software Engineering*, 16–22.

[35] Guerriero, M., Marconi, F., et al., (2018). Towards DevOps for Privacy-by-Design in Data-Intensive Applications: A Research Roadmap. ICPE '17 Companion: Proceedings of the 8th ACM/SPEC.

[36] Rimba, P., Zhu, L., Bass, L., Kuz, I., & Reeves, S. (2016). Composing Patterns to Construct Secure Systems. *Proceedings - 2015 11th European Dependable Computing Conference, EDCC 2015*, 213–224.

[37] Ullah, F., Johannes Raft, A., Shahin, M., Zahedi, M., and Babar, M.A (2017) Security Support in Continuous Deployment Pipeline, *Proceedings of 12th International Conference on Evaluation of Novel*

Approaches to Software Engineering

[38] Paule, C., Dullmann, T. F., & van Hoorn, A. (2019). Vulnerabilities in Continuous Delivery Pipelines? A Case Study. *Proceedings - 2019 IEEE International Conference on Software Architecture - Companion, ICSA-C 2019*, 102–108

[39] Casola, V., de Benedictis, A., Rak, M., & Villano, U. (2020). A novel Security-by-Design methodology: Modeling and assessing security by SLAs with a quantitative approach. *Journal of Systems and Software*, *163*.

[40] Jaatun, M. G. (2019). Architectural risk analysis in agile development of cloud software. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, *2019-December*, 295–300.

[41] Rios, E., Iturbe, E., et al., (2019). Service level agreement-based GDPR compliance and security assurance in (multi)Cloud-based systems. *IET Software*, *13*(3), 213–222.

[42] Abrahams, M., Langerman, J., (2018). Compliance at Velocity within a DevOps Environment**.** 2018 Thirteenth International Conference on Digital Information Management (ICDIM).

[43] Atighetchi, M., Simidchieva, B., & Olejnik, K. (2019). Security Requirements Analysis - A Vision for an Automated Toolchain. *Proceedings - Companion of the 19th IEEE International Conference on Software Quality, Reliability and Security, QRS-C 2019*, 97–104.

[44] Jaatun, M. G., Cruzes, D. S., & Luna, J. (2017). DevOps for better software security in the cloud. *ACM International Conference Proceeding Series*, *Part F130521*.

[45] Brewer, J., Joyce, G., & Dutta, S. (2017). Converging data with design within agile and continuous delivery environments. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10288 LNCS*, 533–542.

[46] Fitzgerald, B., & Stol, K. J. (2014). Continuous software engineering and beyond: Trends and challenges. *1st International Workshop on Rapid Continuous Software Engineering, RCoSE 2014 - Proceedings*, 1–9.

[47] Nassereddine, M. (2020). DevSecOps practices for an agile and secure it service management. In *Journal of Management Information and Decision Sciences* (Vol. 23, Issue 2).

[48] Larrucea, X., Berreteaga, A., & Santamaria, I. (2019). Dealing with Security in a Real DevOps Environment. *Communications in Computer and Information Science*, *1060*, 453–464.

[49] Vijayakumar, K., & Arun, C. (2019). Continuous security assessment of cloud based applications using

distributed hashing algorithm in SDLC. *Cluster Computing*, *22*, 10789–10800.

[50] Díaz, J., Pérez, J. E., Lopez-Peña, M. A., Mena, G. A., & Yagüe, A. (2019). Self-service cybersecurity monitoring as enabler for DevSecops. *IEEE Access*, *7*, 100283–100295.

[51] Hilton, M., Nelson, N., Tunnell, T., Marinov, D., & Dig, D. (2017). Trade-offs in continuous integration: Assurance, security, and flexibility. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, *Part F130154*, 197–207.

[52] Gilje Jaatun, M. (2018). *Software Security Activities that Support Incident Management in Secure DevOps*. ARES 2018: Proceedings of the 13th International Conference on Availability, Reliability and Security.

[53] Morales, A. R. F. J. A. A., Yasar, H., & Volkman, A. (2018). *Implementing DevOps Practices in Highly Regulated Environments* (Vol. 4). ACM.

[54] Stockhausen, H. M. von, & Rose, M. (2020). Continuous security patch delivery and risk management for medical devices. *Proceedings - 2020 IEEE International Conference on Software Architecture Companion, ICSA-C 2020*, 204–209.

[55] Zheng, E. (2018). *Building a Virtually Air-gapped Secure Environment in AWS: with principles of devops security program and secure software delivery. HotSoS'18.*

[56] Lavirotte, S., Metzger, A., et al., (2020). *Development and Operation of Trustworthy Smart IoT Systems: The ENACT Framework.* DEVOPS 2019: Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment pp 121–138.

[57] Ferry, N., Nguyen, P. H., Song, H., Rios, E., Iturbe, E., Martinez, S., & Rego, A. (2020). Continuous deployment of trustworthy smart IoT systems. *Journal of Object Technology*, *19*(2), 1–23.

[58] Mårtensson, T., Ståhl, D., & Bosch, J. (2016). Continuous integration applied to software-intensive embedded systems – Problems and experiences. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10027 LNCS*, 448–457.

[59] Carturan, S. B. O. G., & Goya, D. H. (2019). A systems-of-systems security framework for requirements definition in cloud environment. *ACM International Conference Proceeding Series*, *2*, 235–240.

[60] Pallis, G., Trihinas, D., Tryfonos, A., & Dikaiakos, M. (2018). DevOps as a Service: Pushing the Boundaries of Microservice Adoption. *IEEE Internet Computing*, *22*(3), 65–71.

[61] Weber, I., Nepal, S., & Zhu, L. (2016). Developing Dependable and Secure Cloud Applications. *IEEE*

Internet Computing, *20*(3), 74–79.

[62] Rios, E., Iturbe, E., & Palacios, M. C. (2017). Self-healing multi-cloud application modelling. *ACM International Conference Proceeding Series*, *Part F130521*.

[63] Thanh, T. Q., Covaci, S., Magedanz, T., Gouvas, P., & Zafeiropoulos, A. (2016). Embedding security and privacy into the development and operation of cloud applications and services. *2016 17th International Telecommunications Network Strategy and Planning Symposium, Networks 2016 - Conference Proceedings*, 31–36.

[64] Faily, S., & Iacob, C. (2017). Design as code: Facilitating collaboration between usability and security engineers using CAIRIS. *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017*, 76–82.

[65] Ahmadvand, M., Pretschner, A., Ball, K., & Eyring, D. (2018). Integrity protection against insiders in microservice-based infrastructures: From threats to a security framework. *Lecture Notes in Computer Science, 11176 LNCS*, 573–588.