IBIMA Publishing
Communications of International Proceedings
https://ibimapublishing.com/p-articles/44SE/2024/4451824/
Vol. 2024 (24), Article ID 4451824

The DevSecOps: Tools, Techniques and Implementation Strategies Among Teams and Organizations*

Stanisław SKRZYPECKI

Military University of Technology, Faculty of Cybernetics, Warsaw, Poland

Correspondence should be addressed to: Stanisław SKRZYPECKI, stanislaw.skrzypecki@wat.edu.pl

* Presented at the 44th IBIMA International Conference, 27-28 November 2024 Granada, Spain

Abstract

This paper is a review of DevSecOps which is approach (related to DevOps) aimed to enhance the efficiency, speed, and quality of software delivery in IT process management. The paper is based on an analysis of the relevant academic literature, gray literature, industry publications, and product documentation, along with other key materials from leading IT industry vendors. Description, core principles, challenges and common implementation strategies of DevSecOps were presented. The paper provides foundational DevSecOps concepts and a comparison with the DevOps model. Moreover, various techniques and tools integral to DevSecOps practices and various strategies for implementing DevSecOps within teams and organizations are presented.

Keywords: DevSecOps, DevOps, shift-left, CI/CD, DevSecOps implementation, DevSecOps techniques

Introduction

Despite numerous global and regional disruptions in recent years, such as the COVID-19 pandemic and Russia's invasion of Ukraine, Israeli-Palestinian conflict or instability in Africa continent, the market of IT services and software has been growing steadily in recent years (although there are rumors its slowing down or is crashing) . At the same time, recent reports from cybersecurity institutions indicate a rising number of cyber incidents and an increasing number of software vulnerabilities. Given the need to enhance the security of IT products at every stage of their lifecycle, particularly during planning, development, and deployment phases, this study explores the integration of the DevSecOps approach among teams and organizations. DevSecOps combines the concepts of development, security, and operations into a unified framework within the software development and maintenance cycle. This paper aims to present what DevSecOps is, what tools and techniques can be used, which phases of the cycle and strategies to implement DevSecOps among teams and organizations.

DevSecOps and DevOps are two related approaches to IT process management that aim to enhance the efficiency, speed, and quality of software delivery. The DevOps approach emerged from the growing popularity of agile software development methodologies. It originated from the conflict between development (Dev) and operations (Ops) teams and from the need to increase the frequency of software deployments in the early 2000s. One notable industry presentation that illustrated these practices – emphasizing collaboration between development and operations teams as well as fostering an organizational culture aimed at achieving "10 deployments per day" on the Flickr platform – was delivered in 2009 [1]. DevOps focuses on integrating development and operations teams

Cite this Article as: Stanisław SKRZYPECKI, Vol. 2024 (24) "The DevSecOps: Tools, Techniques and Implementation Strategies Among Teams and Organizations " Communications of International Proceedings, Vol. 2024 (24), Article ID 4451824, https://doi.org/10.5171/2024.4451824

to streamline processes for continuous integration and continuous deployment (CI/CD) of software [7]. The DevSecOps aim to involve security teams into DevOps process. The workflows in both DevOps and DevSecOps are fundamentally similar and follow sequential stages within pipelines. These cycles comprise the following phases (as Atlassian vendor presenting [2], it was also illustrated in Figure 1):

- Discovery – researching, gathering ideas, and defining the project scope.
- Planning – dividing and tracking tasks, planning sprints.
- Building – allowing implementation to take place in environments consistent with production environments.
- Testing – automatically testing code to ensure its correctness and quality before deployment, including the use of staging environments.
- Deployment – frequently introducing new versions of software (new features, bugfixes and other changes) in an automated manner using CI/CD tools, employing various deployment techniques such as blue-green, canary, rolling, A/B, shadow, and recreate deployments.
- Operating – managing IT infrastructure and maintaining applications running in production, including configuration management and infrastructure maintenance.
- Observing – real-time monitoring of applications and infrastructure to quickly identify and resolve issues, as well as tracking service performance and availability.
- Continuous Feedback – evaluating each deployed version and collecting customer feedback to improve both the next version of the software and the entire process.

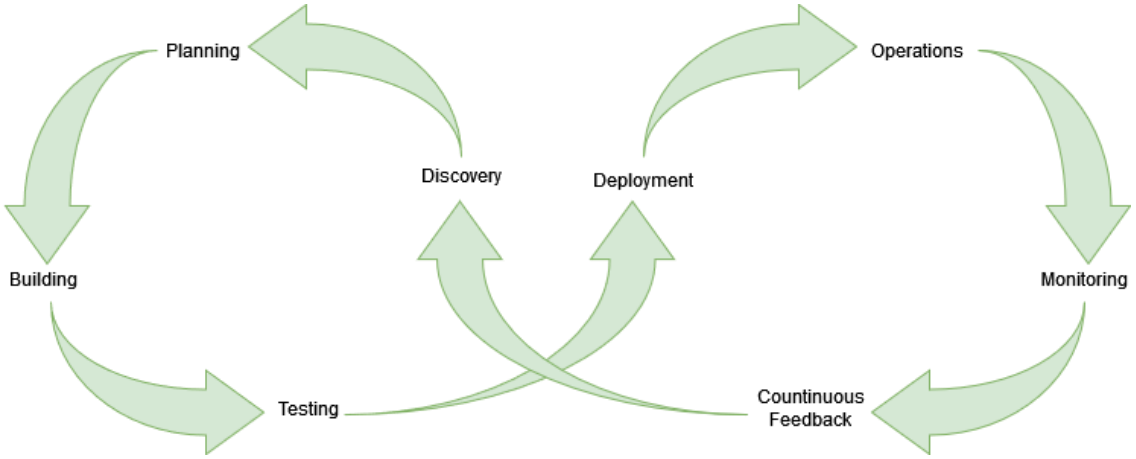

Fig. 1 DevOps Cycle: Discovery, Planning, Building, Testing, Deployment, Operations, Monitoring, Continuous Feedback.

DevSecOps represents an evolution of the DevOps approach by increasing the emphasis on security aspects through integrating security solutions at the earliest possible stages of the software development lifecycle. In traditional development models, security is often treated as a separate phase and addressed at a later stage in the process. This approach of enhancing security as early as possible is also referred to as "shift-left". To understand this concept the process may be visualized as an arrow starting from the left from programming, proceeding through the stages of building and deployment to production, and ending on the right side. Simultaneously, some vendors like Redhat emphasize that to maintain a high level of security, it is necessary not only to focus on "shift-left" but also to uphold "shift-right," which involves testing and monitoring the deployed software and production infrastructure [12].

A key aspect of DevSecOps is to introduce security in the most seamless and transparent manner possible, so that it does not diminish the agility and efficiency of developers, nor require significant changes in their prepared environments and utilized tools. Additionally, security should be implemented proactively (not merely in response to emerging problems) and holistically, as an integral part of the activities of development teams and ensuring their operational capabilities. This includes placing the greatest possible emphasis on collaboration between teams and maximizing the use of automation.

Introducing security planning and thinking, incorporating security tests (both static and dynamic), and providing rapid feedback on detected issues allows development teams to maintain efficiency while keeping security at a

high level that goes hand in hand with quality. A. Gupta said that for DevSecOps, the same as for DevOps, it is crucial to maintain the ability to effectively meet the requirements set by clients and the market through the capacity for rapid expansion of functionalities [8].

Pillars of DevSecOps

Various sources outline key concepts fundamental to the DevSecOps approach. Fo instance GitLab vendor identifies four main concepts: automation, collaboration, established security policy, and system state visibility [7]. Many of these concepts overlap, in principle, with the DevOps approach. However, for the purposes of this study, six pillars of DevSecOps have been adopted based on a framework published by the Cloud Security Alliance [4]. This framework places a significantly greater emphasis on the integration of security, distinguishing it from concepts typically associated with DevOps.

Pillar 1: Collective responsibility

The first pillar of DevSecOps is fostering a mindset within the organization (or team) that places security as a top priority and considers it a collective responsibility shared by all members. Every team member, from end-users to development, operations, and security teams, plays a role in maintaining security and is accountable for the organization's overall security posture. This shift requires treating security as an integral part of the development process, rather than an add-on after the software is completed. It is crucial for security to be seen not as a separate function, but as a core element of every business and development process.

Pillar 2: Collaboration and integration

Effective implementation of security in a DevSecOps approach requires collaboration among development, operations, and security teams, facilitating a better understanding of potential threats and rapid responses. A culture of collaboration and security awareness throughout the organization (or team) enables effective monitoring and reporting of anomalies. It is essential to recognize that human error is often the weakest link, with many security incidents arising from mistakes. Blame-seeking and confrontation within teams can negatively impact the disclosure of future security threats, underscoring the importance of a supportive, collaborative environment.

Pillar 3: Pragmatic implementation

DevSecOps implementation should be adapted to the specific needs and characteristics of the organization. Each organization varies in structure, maturity, and processes, meaning there is no universal approach or toolset suitable for every situation. Given the diversity of available software, implementation challenges may arise post-adoption, resulting in tools that do not yield expected benefits in security risk management. Many tools require complex implementation processes, which can be time-consuming and costly. Such tools may also lack compatibility with existing systems, diminishing their impact on security. For effective DevSecOps, it is crucial to adopt a holistic approach to security implementation and set precise goals. Choosing tools that integrate well with other software allows adaptation to current processes and facilitates future integrations. Pragmatism here means that actions should bring tangible benefits, be adapted to the organization's context, and maintain a forward-looking perspective.

Pillar 4: Bridging compliance and development

A major challenge is balancing security requirements with the need for rapid software development. Security teams should establish policies based on risk assessment; however, even with audits and controls, defining and evaluating metrics can be challenging. Identifying appropriate, measurable parameters for monitoring throughout the software lifecycle is essential. Automating these processes enhances risk management and regulatory compliance. DevSecOps teams must incorporate compliance requirements into their day-to-day activities, enabling prompt responses to deviations. This approach allows organizations to remain agile in delivering new functionalities while meeting regulatory requirements and minimizing security risks.

Pillar 5: Automation

Automation is a fundamental pillar of DevSecOps, reducing manual effort, accelerating testing, integration, and code deployment processes, thus enabling more frequent releases and error minimization. Frequent, automated, high-quality testing and continuous feedback improve software quality and streamline its operation. All feasible processes should be automated; others should be automated to the greatest extent possible, replaced or eliminated if necessary. It is important to recognize that automated processes, particularly those involving various types of security tests (such as vulnerability scanners, static, or dynamic analysis), may introduce build delays or errors. Such problematic processes can be resolved through workflow improvements or semi-automated approaches.

The main automation process in DevSecOps is the CI/CD (Continuous Integration/Continuous Deployment) pipeline, responsible for delivering a final product from implementation through code integration, automated testing, and deployment. Continuous deployment refers to fully automated application deployment to production, whereas processes requiring manual intervention (e.g., approver confirmation, scheduling a maintenance window) are known as continuous delivery. The CI/CD process, with both manual and automated aspects, is illustrated in Figure 2.

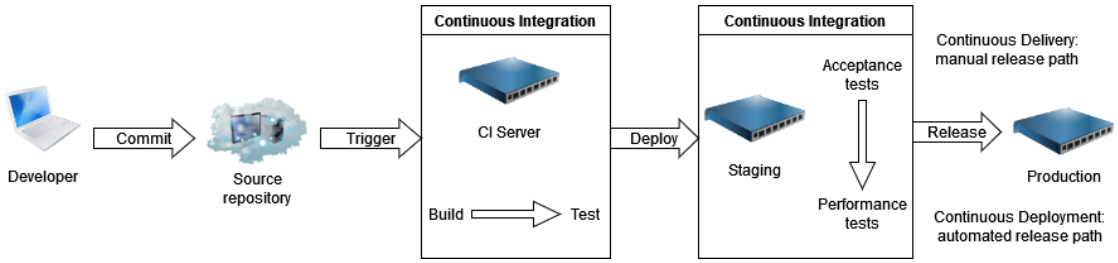

Fig. 2 CI/CD process with manual and automatic release paths.

Automation, particularly the inclusion of automated security tests within CI/CD processes, is one of the most critical pillars of the DevSecOps approach. Unfortunately, manual and unsystematic practices in coding, testing, and deployment often lead to issues caused by human error, reduce the likelihood of detecting security vulnerabilities before deployment, and can result in unstable software performance.

Additionally, in the context of process automation, version control systems play an essential role, as entire CI/CD processes should be defined as code. Both software and infrastructure configurations (IaaC - Infrastructure as Code) should be stored within such a system, with records of all changes and their authors. The version control system itself should serve as the central source of truth during deployments, ensuring traceability and consistency across the lifecycle.

Pillar 6: Measure, monitor, report and act

A core principle of DevSecOps is that effective management and improvement requires concrete metrics. Without robust monitoring and predefined metrics, it is impossible to determine if progress is being made toward established goals. Within the DevSecOps framework, commonly used metrics include deployment frequency, time to patch security vulnerabilities, percentage of code covered by automated testing, and the number of automated tests applied to a given application. These metrics enable teams to assess performance, identify areas for improvement, and respond proactively to security and operational issues.

Tools and techniques of different phases of DevSecOps

The following section highlights techniques and types of tools commonly used within the DevSecOps approach. This analysis is based on the *DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools* [6], a report publicly released in 2023 by the United States Department of Defense to introduce this area. It focuses on key techniques (classified as essential), detailing the phases of the process in which they should be implemented. The terminology and approach have been adapted for applications beyond the military context. It is essential to note, in line with the pillar of pragmatic implementation, that the selection of techniques and tools should be tailored to the specific organization, project, and the broader context. This includes considerations of readiness for change, organizational maturity and culture, timeline, and available human and financial resources.

Planning techniques

- **Change management planning**: Planning the change control process based on organizational policy and software development best practices. Tools: team collaboration systems, issue tracking systems.
- **Configuration identification**: Identifying or manually entering configuration items into the Configuration Management Database (CMDB) and establishing basic requirements for systems. Tools: CMDB, source code repository, artifact repository, team collaboration systems.
- **Configuration management planning**: Planning the configuration management process and identifying configuration items. Tools: team collaboration systems, issue tracking systems.
- **DevSecOps process design**: Designing project-specific workflows in line with the DevSecOps approach. Tools: team collaboration systems.
- **Documentation versioning**: Tracking design changes and storing all versions of documents. Tools: team collaboration systems.
- **Implementing Infrastructure as Code (IaaC)**: Setting up and configuring the environment using the IaaC approach. Tools: configuration automation tools.
- **Project (release) planning**: Managing project tasks and planning the release of new software versions. Tools: team collaboration systems, project management systems.
- **Project team onboarding planning**: Planning the onboarding process for the project team, communication protocols, and access control policies. Tools: team collaboration systems.
- **Risk management**: Assessing risk based on system architecture, supply chain information, and security risks. Tools: team collaboration systems.
- **Software requirements analysis**: Gathering requirements from stakeholders. Tools: requirements management tools, team collaboration systems, issue tracking systems.
- **System design**: Designing the system based on requirements. Tools: team collaboration systems, issue tracking systems, system design tools.
- **Test implementation**: Setting up the infrastructure and environment for testing. Tools: testing tools, team collaboration systems.
- **Test planning**: Planning tests, including acceptance criteria validation, and defining component and software tests. Tools: testing tools, team collaboration systems.
- **Threat modelling**: Identifying potential threats, weaknesses, and vulnerabilities, and defining a mitigation plan. Tools: threat modelling tools.

Design (development) phase

- **Programming**: Developing software by preparing source code. Tools: Integrated Development Environment (IDE).
- **Uploading source code**: Uploading source code to a source code repository (version control system). Tools: source code repository.
- **Logging code commits**: Logging successful code commits or analyzing rejected commits to track internal threats. Tools: commit logging tools.
- **Commit scanning**: Checking changes for sensitive information before pushing to the main repository. Tools: security plugins for the source code repository.
- **Database component testing**: Conducting closed testing of database components without considering code specifics. Tools: database testing tools, code coverage analysis tools.
- **Documentation**: Creating detailed implementation documentation. Tools: IDE, document editors, API documentation generation tools.
- **Static analysis**: Analyzing code to identify issues in logic and programming techniques. Tools: static code analysis tools.
- **Pre-commit code scanning**: Scanning and analyzing code during writing, notifying developers of weaknesses and providing corrective suggestions before committing. Tools: IDE security plugins.
- **Test creation**: Developing test procedures, test data, test scripts, and test scenario configurations. Tools: IDE, specific testing tools.
- **Unit testing**: Creating and executing unit test scripts. Tools: unit testing tools, test coverage tools.
- **Functional testing**: Testing code functionality according to predefined requirements. Tools: functional testing tools.
- **Preparing security scripts**: Preparing and configuring scripts to enforce security policies. Tools: IDE (IDE security plugins).

Building phase

- **API security testing**: Testing APIs for compliance with security requirements. Tools: testing toolset.

- **Compilation**: Compiling and linking (libraries) of source code and dependencies. Tools: build tool, linting tool, artifact repository.
- **Build configuration control and audit**: Tracking build results, SAST reports, and dependency verification. Tools: team collaboration systems, issue tracking systems, CI/CD orchestrator.
- **Component testing**: Conducting closed testing of program components without considering code specifics. Tools: testing toolset, test coverage tool.
- **Dependency vulnerability scanning**: Identifying vulnerabilities in libraries and dependent components (especially open-source). Tools: dependency scanning/BOM tool.
- **Functional testing**: Verifying that code operates according to specified requirements. Tools: testing tool.
- **Software integration testing**: Creating and executing test scripts to test larger portions of code, including interactions between software elements. Ensuring the new software integrates correctly with existing components without disrupting their functionality. Tools: testing tool.
- **Regression testing**: Re-running functional and non-functional tests to ensure the software works correctly after changes. Tools: test management tool.
- **Release packaging**: Packaging binary artifacts, infrastructure configuration scripts, test scripts, documentation, checksums, digital signatures, and release notes. Tools: release packaging tool.
- **Static Application Security Testing (SAST)**: Conducting static security testing of the entire software codebase. Tools: SAST tool.
- **Artifact storage**: Storing artifacts in an artifact repository. Tools: artifact repository.

Testing phase

- **API security testing**: Testing APIs for compliance with security requirements. Tools: testing toolset.
- **Compliance scanning**: Conducting compliance audits of artifacts, software instances, and system components. Tools: compliance scanning tools, software license checking tools, security compliance tools.
- **Manual security testing**: Performing penetration tests using tools and procedures to assess system security through simulated attacks. Tools: various tools and scripts (including network security testing tools).
- **Regression testing**: Re-running functional and non-functional tests to ensure the software operates correctly after changes. Tools: testing toolset.
- **Static Application Security Testing (SAST)**: Conducting static security testing for the entire software codebase. Tools: SAST tool.
- **Software integration testing**: Creating and executing test scripts to test larger code portions, including interactions between software elements, ensuring new software integrates correctly with existing components without disrupting functionality. Tools: testing tool.
- **System testing**: Testing the system as a whole, including code, configuration, and deployment. Tools: test management tool.

Release phase

- **Artifact replication**: Replicating released artifacts to regional artifact repositories. Tools: artifact repositories (release versions, regional versions).
- **Operational readiness testing**: Testing the suitability and effectiveness of the entire system. Tools: testing toolset, compliance scanning tools.
- **Go/No-Go release decision**: Deciding to release (or withhold) artifacts to the artifact repository for the production environment. Tools: CI/CD orchestrator.

Delivery phase

- **Configuration integration testing**: Testing the fully integrated system to ensure it meets requirements. Tools: testing tools.
- **Delivery of released artifacts**: Transferring released artifacts to the artifact repository. Tools: artifact repository.
- **Delivery results review**: Reviewing the release and all related artifacts, configuration test results, and recommendations. Tools: not specified.
- **Operational team acceptance**: Testing the delivered artifacts to ensure they meet operational requirements. Tools: not specified.

Deployment phase

- **Compliance testing**: Testing to verify that delivered items meet specification requirements, technical standards, contractual obligations, or regulations. Tools: test management tool.
- **Interoperability testing**: Testing communication and cooperation between different software and systems. Tools: test management tool.
- **Purpose-oriented testing**: Tests designed to evaluate and validate the performance of new or existing systems in real-world usage scenarios. Tools: test management tool.
- **Performance testing**: Testing the stability, speed, scalability, and responsiveness of an application or system under a specified load. Tools: test management tool.
- **Post-deployment security scanning**: Scanning the security of the system and infrastructure after deployment. Tools: security compliance tool.
- **User evaluation/feedback**: Collecting and analyzing user evaluations and feedback. Tools: survey and bug reporting tools.

Operations phase

- **Business operations**: Ongoing management of resource utilization and billing. Tools: operational dashboard.
- **Operational testing and cybersecurity assessment**: Evaluating operational effectiveness, suitability, resilience, and system capabilities under real-world cyber conditions. Tools: cybersecurity testing tools.
- **Event logging**: Recording system, network, user, and data-related events. Tools: event logging tools.
- **User evaluation/feedback**: Collecting and analyzing user evaluations and feedback. Tools: survey and bug reporting tools.

Monitoring phase

- **Asset inventory**: Inventorying IT assets within the system. Tools: inventory management tools.
- **Compliance monitoring of Commercial Off-The-Shelf (COTS) software**: Monitoring compliance of COTS software with selected policies (e.g., STIG). Tools: code compliance monitoring tools.
- **Resource and service compliance monitoring**: Monitoring the compliance status of deployed resources and cloud services with selected policies (e.g., NIST SP 800-53). Tools: service compliance monitoring tools.
- **Log analysis and audit**: Filtering, aggregating, analyzing, and correlating logs. Tools: log collection and processing tools.
- **System security monitoring**: Monitoring the security of all system components. Tools: InfoSec Continuous Monitoring (ICSM), issue tracking systems, notification and alert systems, operational dashboard.
- **User evaluation/feedback**: Collecting and analyzing user evaluations and feedback. Tools: survey and bug reporting tools.

Continuous feedback phase

- **User evaluation/feedback**: Collecting and analyzing user evaluations and feedback. Tools: survey and bug reporting tools.
- **Product backlog update**: Updating the product backlog with new features, improvements, bug fixes, vulnerability remediations, and performance enhancements based on collected metrics. Tools: requirements management tool.

Supplementary techniques for all above phases

In each phase presented in the previous section of this chapter, the following techniques are applied:

- **Test audit**: Tracking who performs which tests and recording the results of these tests. Tools: test management tool.
- **Mission-based cyber risk assessment**: Assessing risk based on the intended purpose of the system (or software). Tools: risk assessment tools.

Supplementary techniques for specific phases

In the testing, release, deployment, implementation, and operational phases, the following additional techniques are also applied.:

- **Software Bill of Materials (SBOM) analysis**: Collecting and analyzing information on the origin of all dependencies and third-party components used in the system. Tools: SBOM analysis tool.

- **Continuous risk monitoring in the software factory**: Monitoring controls within the software factory. Tools: monitoring toolset.

DevSecOps implementation strategies among teams and organizations

Integrated Approach to DevSecOps Implementation

One of the methods proposed in the literature for implementing DevSecOps is the integrated DevSecOps deployment approach developed by Akanksha Gupta [8]. This scientific article outlines six key actions that organizations must undertake to successfully implement DevSecOps without additional challenges.

I. Understanding the Need for DevSecOps

The first step is to gain a comprehensive understanding of what DevSecOps entails and why it is necessary. Organizations should assess the benefits DevSecOps can bring, analyze current issues, and identify areas for improvement.

II. Establishing DevSecOps ambassadors

From various departments within the organization, a group of DevSecOps ambassadors should be identified – individuals who demonstrate sufficient motivation and are willing to promote the DevSecOps initiative. Those ambassadors will play a key role in spreading awareness and supporting the implementation of DevSecOps across the organization. In specific cases, it may be necessary to engage ambassadors from outside the organization.

III. Developing a DevSecOps strategy

A DevSecOps strategy should be formulated and prioritize security, tasks automation, and implementation of continuous integration and deployment (CI/CD). Such strategy ensures that all activities will be carried out with security as a primary focus.

IV. Communicating with leadership

Communicating with management and securing their support is essential for successful DevSecOps implementation. Adoption strategy should be presented, including initial deployment costs and long-term benefits, and the initiative should be sponsored by a responsible individual who is overseeing all involved pillars.

V. Execution

The execution phase involves starting with small steps, continuously gathering feedback, and iterating repeatedly. This is also the stage for acquiring and implementing tools that will facilitate DevSecOps adoption (preferably starting with test deployments, or Proof of Concept).

VI. Audit and measurement of success

Ongoing monitoring and feedback collection are essential to refining the DevSecOps process. Establishing an alert and notification system, along with operational dashboards, enables proactive tracking of application performance. Additionally, surveys and other feedback systems help identify issues and set appropriate, measurable metrics, allowing organizations to pinpoint areas for improvement and assess the effectiveness of the DevSecOps implementation.

DevSecOps Maturity Model (DSOMM)

Another approach to implementing DevSecOps is the use of the DevSecOps Maturity Model (DSOMM), developed by the OWASP community [10]. DSOMM is designed to assess and improve DevSecOps practices within organizations (or teams) by integrating security throughout the DevSecOps lifecycle. The model is divided into five categories (dimensions) and eighteen subcategories (sub-dimensions) that define various aspects of an

organization's (or team's) maturity level within DevSecOps. These categories and subcategories are presented in Table 1.

Table 1: Categories and subcategories of DSOMM.

Categories	Subcategories	
	Name	**Description**
Building and Deployment	Building	Processes related to creating and testing artifacts.
	Deployment	Processes related to deploying and managing production environments.
	Patch Management	Processes for managing updates and security patches.
Culture and Organization	Design	Incorporating security in the design phase.
	Education and Advisory	Security training and consulting for teams.
	Processes	Standards and practices related to change management and continuity.
Implementation	Application Hardening	Practices related to securing applications.
	Development and Source Control	Practices related to version control and source code protection.
	Infrastructure Hardening	Practices related to securing IT infrastructure.
Information Gathering	Logging	Centralization and analysis of system and application logs.
	Monitoring	Metrics, alerting, and visualization of system and application status.
Testing and Verification	Application Testing	Security testing of application modules and integration.
	Consolidation	Management and visualization of security defects.
	Dynamic Depth for Applications	Testing dynamic components of applications.
	Dynamic Depth for Infrastructure	Testing dynamic components of infrastructure.
	Static Depth for Applications	Software composition analysis and static code analysis.
	Static Depth for Infrastructure	Analysis and testing of static infrastructure components.
	Test Intensity	Determining the intensity and scope of security testing.

For specific subcategories DSOMM defines actions to facilitate the implementation of the DevSecOps approach. Those actions are organized according to the recommended sequence of implementation and based on the

organization's (or team's) maturity level. Their order is determined by defining five maturity levels (as per the model's structure). These maturity levels include:

- Level 1: Basic understanding of security practices
- Level 2: Adoption of fundamental security practices
- Level 3: High level of security practice adoption
- Level 4: Very high level of security practice adoption
- Level 5: Advanced and large-scale implementation of security practices

DevSecOps Fundamentals Playbook

Another approach to implementing DevSecOps is presented in the DevSecOps Fundamentals Playbook, a non-classified guide released by the United States Department of Defense. This guide outlines ten steps, which are as follows [5]:

Step 1: Adopt a DevSecOps culture

DevSecOps integrates development, deployment, security, and operations, requiring commitment from all stakeholders and shifting from an "I" to a "we" approach. Success depends on collaboration among teams and the understanding that failure is a collective setback, not just the responsibility of one team.

Step 2: Adopt the "Infrastructure as Code" (IaaC) approach

The IaaC approach records infrastructure definitions and configurations in text files managed within a source code repository, eliminating configuration discrepancies across environments. IaaC enables repeatable and consistent automation of infrastructure management, ensuring environmental consistency.

Step 3: Adopt microservices and containerization

Containerization and microservices development align with modular architecture requirements, creating loosely coupled business services within containers. This approach enables flexible scaling, improves security, and facilitates easier change management. Microservices allow for independent development of components and faster responses to business needs.

Step 4: Adopt a capability model instead of a maturity model

The guide suggests that a capability model, rather than a traditional maturity model, is more effective in enhancing performance and adopting DevSecOps. Key metrics to define capabilities include deployment frequency, lead time from code to deployment, mean time to recovery, and failure rate.

Step 5: Implement continuous improvement through key capabilities

To sustain continuous improvement, twenty-four key capabilities have been identified to drive advancements in both DevSecOps teams and the organization. These capabilities are organized into five broad categories: 1) Continuous delivery, 2) Architecture, 3) Organizational culture, 4) Products and processes, and 5) Lean management and monitoring.

Step 6: Establish a software factory

All in-house software development activities should be carried out within an established software factory with DevSecOps processes in place. The simplest way to achieve this is to implement a pre-built commercial or open-source platform.

Step 7: Define complete and relevant DevSecOps lifecycles

Each software factory should implement DevSecOps lifecycles tailored to different applications, utilizing pipelines suited to specific processes and artifacts. Pipelines should consist of well-defined essential processes and scripts that work together within DevSecOps tools, establishing clear workflows and extensive automation.

Step 8: Adopt agile software acquisition policies

At the organizational level, the software acquisition model should be simplified to enable continuous integration and delivery, meeting end-user needs effectively.

Step 9: Strive for cyber resilience

Cyber resilience is the ability to anticipate, withstand, recover from, and adapt to adverse conditions, cyberattacks, or system compromises. The goal of DevSecOps is to embed cyber resilience into applications at every stage of their lifecycle.

Step 10: Shift security and operations left

Shifting security and operability testing and evaluation earlier in the DevSecOps lifecycle enables quick identification and resolution of issues, reducing risk, enhancing efficiency, and maintaining the delivery pace.

Conclusion

The analysis of DevSecOps in this study confirms that the DevSecOps approach enhances the cybersecurity of products resulting from IT projects within an organization. A detailed introduction highlights that DevSecOps is primarily characterized by collaboration among development, operations, and security teams. Such collaboration fosters an organizational culture focused on elevating security to a central role across all phases of the DevSecOps lifecycle. Additionally, the section on DevSecOps pillars discusses the importance of process automation (including automated security testing) for swiftly identifying and resolving issues while minimizing human error, such as mistakes in repetitive tasks. Continuous monitoring, reporting, and response, along with the development of appropriate metrics, are also emphasized as critical to maintaining security.

Another objective of this study – to explore how DevSecOps can improve security and how it can be implemented within a team (or organization) – was achieved by presenting various techniques used in different phases of DevSecOps, especially those essential to security. Three selected DevSecOps implementation approaches were then presented: an integrated approach comprising six defined actions, the DevSecOps Maturity Model, and the *DevSecOps Fundamentals Playbook*. By choosing an implementation strategy tailored to the team's (or organization's) needs and leveraging the presented techniques and tools, teams (or organizations) should be well-prepared to begin the process of integrating DevSecOps within their own structures.

References

- Allspaw, J. & Hammond, P., 2009. 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr. Velocity.
- Atlassian, What Is DevOps?, [Online]. Available at: https://www.atlassian.com/devops [Accessed 8 June 2024].
- CERT Polska, 2023. Raport roczny z działalności CERT POLSKA 2023, [Online]. Available at: https://cert.pl/uploads/docs/Raport_CP_2023.pdf [Accessed 2 June 2024].
- Cloud Security Alliance, Six Pillars of DevSecOps, [Online]. Available at: https://cloudsecurityalliance.org/artifacts/six-pillars-of-devsecops [Accessed 13 June 2024].
- Department of Defense, USA, 2021. DevSecOps Fundamentals Playbook. Version 2.0.
- Department of Defense, USA, 2023. DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools.

- GitLab, What is DevSecOps?, [Online]. Available at: https://about.gitlab.com/topics/devsecops/#what-is-dev-sec-ops [Accessed 8 June 2024].

- Gupta, A., 2022. An Integrated Framework for DevSecOps Adoption. International Journal of Computer Trends and Technology, 70(6), pp.19-23.

- NIST, National Vulnerability Database - Statistics, [Online]. Available at: https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&&results_type=statistics&&isCpeNameSearch=false [Accessed 2 June 2024].

- OWASP, DSOMM, [Online]. Available at: https://dsomm.owasp.org/ [Accessed 16 June 2024].

- Rajapakse, R.N., Roshan, M.Z.M.A.B.H.S., 2022. Challenges and solutions when adopting DevSecOps: A systematic review. Information and Software Technology, 141.

- Red Hat, Shift left vs. shift right, [Online]. Available at: https://www.redhat.com/en/topics/devops/shift-left-vs-shift-right [Accessed 10 June 2024].

- Główny Urząd Statystyczny, 2024. Dziedzinowe Bazy Wiedzy, Nakłady na oprogramowanie komputerowe przedsiębiorstw niefinansowych o liczbie pracujących 50 i więcej osób, prowadzących księgi rachunkowe. [Online]. Available at: https://dbw.stat.gov.pl/baza-danych [Accessed 2 June 2024].