TRIANZ℠

DEVEOPMENT SECURITY OPERATIONS (DEVSECOPS) PROCEDURE

TRIANZ INTERNAL

trianz.com

Statement of Confidentiality

Information Classification

☐	Public
☒	Internal
☐	Confidential
☐	Restricted

Table of Contents

1. Introduction

DevSecOps is an organizational software engineering culture and practice that aims at unifying software development (Dev), Security (Sec) and operations (Ops). The main characteristic of DevSecOps is to automate, monitor, and apply security at all phases CI/CD pipeline: plan, develop, build, test, release, deliver, deploy, operate, and monitor. In DevSecOps, testing and security are shifted to the left through automated unit, functional, integration, and security testing - this is a key DevSecOps differentiator since security and functional capabilities are tested and built simultaneously.

DevSecOps integrates application and infrastructure security seamlessly into Agile and DevOps processes and tools. It addresses security issues as they emerge, when they're easier, faster, and less expensive to fix (and before they are put into production). Additionally, DevSecOps makes application and infrastructure security a shared responsibility of development, security, and IT operations teams, rather than the sole responsibility of a security silo. It enables “software, safer, sooner”—the DevSecOps motto–by automating the delivery of secure software without slowing the software development cycle.

2. Objectives

The objectives of this procedure are to:

- Increase secure application deployment - new release can be deployed into the production environment securely with the help of security tool automation and integration during CI/CD.
- Controlled risk characterization, monitoring, and mitigation across the application development lifecycle
- Reduce mean-time to production securely - the average time it takes from when new software features are required until they are running in production.

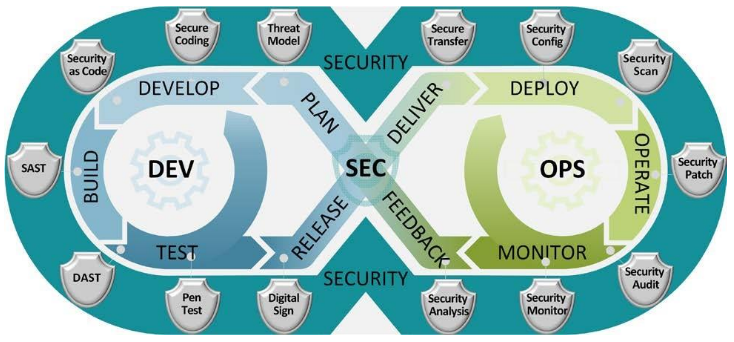
3. Scope

The scope includes all internal and applicable vendor applications that are owned, operated, maintained, and controlled by Trianz internally and externally either on premises or on cloud. DevSecOps also enable organization to bridge the gap between developers, security team, and operations team.

The organization should monitor real world threats and up-to-date advice and information on software vulnerabilities to guide the organization's secure coding principles through continual improvement and learning. This can help with ensuring effective secure coding practices are implemented to combat the fast-changing threat landscape.

4. Process Description (Procedure)

The goal of DevSecOps is to create an environment where building, testing, and deploying software can occur rapidly, frequently, and securely. The DevSecOps software lifecycle phases are: plan, develop, build, test, release, deliver, deploy, operate, and monitor. Where security is embedded within each phase as illustrated in the below figure.



Following best practices should be followed for implementing DevSecOps in Trianz managed CI/CD environment.

4.1 Plan

Secure coding principles should be used both for new developments and in reuse scenarios. These principles should be applied to development activities both within the organization and for products and services supplied by the organization to others. Planning and prerequisites before coding should include

- Organization-specific expectations and approved principles for secure coding to be used for both in-house and outsourced code developments.

- Common and historical coding practices and defects that lead to information security vulnerabilities;
- Configuring development tools, such as integrated development environments (IDE). to help enforce the creation of secure code
- Execute security analysis and create a test plan to determine scenarios for where, how, and when testing will be done for the application.
- Ensure all security requirements are captured with respect to CIAP (confidentiality, integrity, availability, and privacy) and AAA (authentication, authorization, and accountability)
- Ensure potential threats and mitigations are detected using Threat modeling tool during the plan.
- Development team shall undergo application code security and compliance training.
- Establish acceptance test criteria, user designs, and threat models.
- Following guidance of development tools and execution environments as applicable:
- Maintenance and use of updated development tools (e.g. compilers);
    - Qualification of developers in writing secure code.
    - Secure design and architecture, including threat modelling.
    - Secure coding standards and where relevant mandating their use.
    - Use of controlled environments for development.

4.2 Develop

- All the code generated during development must be committed to the source code repository and thus version controlled.
- Deploy linting tools and Git controls to secure passwords and API keys.
- Any custom code being written should be scanned for possible security vulnerabilities during development.
- Ensure developers are adhering to secure coding practices to ensure confidentiality, integrity, and availability.
- Considerations during coding should include:
    - *Secure coding practices specific to the programming languages and techniques being used.*

- *using secure programming techniques, such as pair programming, refactoring, peer review, security iterations and test-driven development.*
- *Using structured programming techniques.*
- *Documenting code and removing programming defects, which can allow information security vulnerabilities to be exploited.*
- *Prohibiting the use of insecure design techniques (e.g. the use of hard-coded passwords, unapproved code samples and unauthenticated web services)*

4.3 Build

- Access to source code and other critical system resources during development, testing or production must be limited to authorized personnel as per Access Control Policy.
- While designing & developing the application, incorporate static application security testing (SAST) tools (Ex:SonarQube), Dynamic Application Security Testing (DAST) tools (Ex: Burpsuite) before deploying to production.
- Ensure code validation is conducted for the changes that are pushed out of a Git repository on a developer's workstation should be scanned using tools like Talisman / Git Secret for ensuring no keys are uploaded inadvertently into the repository
- Conduct scanning using tools like Twistlock tool for all the images in the registry, which scans images during the build and deploy process, and also continuously monitors any vulnerability changes in your running containers and add image signature
- Conduct infrastructure code scan which adds a Build Step for scanning Docker images for security vulnerabilities, using the API provided by Aqua Security.
- The build script should also include targets for running automated unit tests.

4.4 Test

- Ensure the Controls and remediation against the OWASP top 10 vulnerabilities that would be applicable as per technologies are in place.
- Use dynamic application security testing (DAST) tools like Burpsuite, Aqua Enforcer tool to test your application while in runtime. These tools can detect

errors associated with user authentication, authorization, SQL injection, and API-related endpoints.

- Conduct the VA scan on the application hosted systems, images, virtual machines and containers in development using Nessus Tool
- In addition to system testing, ensure testing includes DAST (Dynamic Analysis Security Testing) and IAST (Interactive application security testing) and System composition Analysis (SCA) tools (Ex: Snyk, Whitesource) to test the vulnerabilities, code-smells, dependencies, use of open source libraries.
- Testing should be conducted during and after development. Static application security testing (SAST) processes can identify security vulnerabilities in software. Before software is made operational, the following should be evaluated:
    - *Attack surface and the principle of least privilege.*
    - *Conducting an analysis of the most common programming errors and documenting that these have been mitigated.*

4.5 Release

- Ensure before releasing the application, employ security analysis tools to perform thorough penetration testing and vulnerability scanning.
- Ensure all the identified gaps and vulnerabilities are remediated and code is compliant to security requirement and design.
- Ensure access restrictions are compliant as per below controls :
    - *Restrict the access to the application is only via like user identity specific IPs and block the application accessing from public IPs.*
    - *Application is authenticated via Azure SSO or AD integration with MFA enabled.*
    - *Deploy Web Application Firewall and harden the rules to protection mode.*

4.6 Deploy and Monitor

- After completing all the necessary tests in runtime, send a secure build to production for final deployment.
- Ensure the application is being monitored for any security incidents and alerted.

- Ensure the security incidents are logged and incident management procedure is followed.
- Ensure conducting application security audits and VAPT scan
- After code has been made operational:
    - *Updates should be securely packaged and deployed.*
    - *Reported information security vulnerabilities should be handled (see 8B);*
    - *Errors and suspected attacks should be logged, and logs regularly reviewed to make adjustments to the code as necessary.*
    - *Source code should be protected against unauthorized access and tampering (e.g. by using configuration management tools, which typically provide features such as access control and version control).*
- If using external tools and libraries, the organization should consider:
    - *Ensuring that external libraries are managed (e.g. by maintaining an inventory of libraries used and their versions) and regularly updated with release cycles.*
    - *Selection, authorization and reuse of well-vetted components, particularly authentication and cryptographic components.*
    - *License, security and history of external components.*
    - *Ensuring that software is Unmaintainable, tracked and originates from proven, reputable sources.*
    - *Sufficiently long-term availability of development resources and artefacts.*
- Where a software package needs to be modified the following points should be considered:
    - *Risk of built-in controls and integrity processes being compromised;*
    - *Whether to obtain the consent of the vendor;*
    - *Possibility of obtaining the required changes from the vendor as standard program updates;*
    - *Impact if the organization becomes responsible for the future maintenance of the software as a result of changes;*
    - *Compatibility with other software in use.*

5. Exit Criteria and Output

- Exit Criteria: All reviews are passed

- Output: Secure application deployed to production

6. Roles and Responsibilities

Roles	Responsibilities	External/Interna
Developer	• Developers are responsible to follow secure coding guidelines and need to ensure that the do all the quality test	Internal
DevOps Engineer/Security Architect	• To perform risk assessment, threat modeling and ensure to hardening the security, after evaluating the identified risk and the build process.	Internal
Internal Apps Team	• To conduct the necessary scans and provide the reports • To provide all the Internal and external application details related to design, implementation and operation of the Trianz Applications, Client Applications (wherever applicable) • Remediate the identified vulnerabilities as per the Assurance Report	Internal
IS Operations / Devops team	• Perform Vulnerability scan on Application and provide reports	Internal
InfoSec Team	• Conduct the end to end application security assessment • Review the VA scan and Code Review report • Track the closure report	Internal
CIO/CISO	• Review and approve the final assurance report	Internal

7. Conclusion

DevSecOps aims for a goal of having security checks and controls applied transparently and automatically within a rapid-development automated DevOps pipeline. Shifting security left to start at development makes sure that DevSecOps is effective and it follows the journey of service throughout its lifecycle.

A guiding principle is to ensure security-relevant code is invoked when necessary and is tamper resistant. Programs installed from compiled binary code also have these properties but only for data held within the application. For interpreted languages, the concept only works when the code is executed on a server that is otherwise inaccessible by the users and processes that use it, and that its data is held in a similarly protected database. For example, the interpreted code can be run on a cloud service where access to the code itself requires administrator privileges. Such administrator access should be protected by security mechanisms such as just-in-time administration principles and strong authentication. If the application owner can access scripts by direct remote access to the server, so in principle can an attacker. Webservers should be configured to prevent directory browsing in such cases.

Application code is best designed on the assumption that it is always subject to attack, through error or malicious action. In addition, critical applications can be designed to be tolerant of internal faults. For example, the output from a complex algorithm can be checked to ensure that it lies within safe bounds before the data is used in an application such as a safety or financial critical application. The code that performs the boundary checks is simple and therefore much easier to prove correctness.

Some web applications are susceptible to a variety of vulnerabilities that are introduced by poor design and coding, such as database injection and cross-site scripting attacks. In these attacks, requests can be manipulated to abuse the webserver functionality.

8. References

DevSecOps: A Complete Guide to What, Why, and How - Plutora

What is DevSecOps? | IBM

DevSecOps Implementation Process and Road Map – Security at Every Step - DevOps.com

9. ISO Control Mapping

Category of Control	**ISO 27001:2022 Control**	**Document Name as per ISO 27001:2022**
Technological	8.26 Application security requirements Control Information security requirements shall be identified, specified and approved when developing or acquiring applications	DEV SEC OPS Procedure
Technological	8.31 Separation of development, test, and production environments	
Technological	8.33 Test information	

Document Control

Owner:	CISO	**Release ID:**	DEVSECOP_PROC_0133

For Trianz Process Improvement Group (TPIG) Purpose Only

Version History

Ver. No.	**Date**	**Author**	**Reviewer**	**Approver**	**Introduction/Reason for Change**	**Change Descriptio**
0.1	29- Oct-20	Rajesh B	Vijaya Rajeswari	Phani Krishna	Initial Release	Initial Draft
1.0	30-Oct-20	Rajesh B	Vijaya Rajeswari	Phani Krishna	Reviewed and approved	Approved and Baselined
1.1	3-May-21	Divya	Vijaya Rajeswari	Phani Krishna	For Review	Updated with new information classification
2.0	3-May-21	Divya	Vijaya Rajeswari	Phani Krishna	For Approval	Approved and Baselined
2.1	3-Jan-22	Karthik N			For Review	Information classification updated from “Trianz Confidential” to “Trianz Internal”
3.0	3-Jan-22	Karthik N	Siva N	Siva N	For Approval	Reviewed, Approved and Baselined
3.1	13-Mar-22	Sanjana	Karthik		For Review	Updated the roles and responsibilities

4.0	18-Mar-2022	Sanjana	Siva N	Siva N	For Approval	Reviewed, Approved and Baselined
4.1	21-Mar-2023	Vijaya and Asho Veeramau	Srikanth		For Review	Updated Introduction section and Section 2.1 New template change
5.0	8-May-2023	Vijaya	Srikanth	Srikanth	For Approval	Approved and Baselined
5.1	23-Jan-2024	Rakesh Vijendra	Vijaya		For Review	Requirement from ISO27001:2022 of Secure coding has been addressed
6.0	23-Feb-24	Rakesh Vijendra	Vijaya	Srikanth	For Approval	Approved and Baselined
6.1	30-Apr-25	Kruti	Vijaya R		For Annual Review	Migrated to a new Template and Yearly Review
7.0	29-May-25	Kruti	Vijaya R	Srikanth	For Approval	Approved and Baselined

TRIANZ℠

Contact Information

Name

Email

Phone

Thank You

infosec@trianz.com