Overview of DevSecOps frameworks for Software Development Lifecycle and its current limitations

Aleksandr Krasnov, Dr. Richard Maiti

Abstract:

We present the overview of DevSecOps frameworks for Software Development Lifecycle (SDLC) with their current limitations. At the end of the paper, we present a new framework of more precise assessment of DevSecOps maturity. Unlike other frameworks that leverage various components of governance, compliance, and risk-based assessment with convoluted parameters that can't be applied to a lifecycle across the board, a proposed framework simplifies the process that could apply to any system. It concentrates on five key questions, and the ability to answer those would dictate developmental areas and guide a software team on the work for several quarters ahead.

Keywords:

DevSecOps, Software Development, Maturity Assessment

I. *Introduction*

Over the last several years, DevSecOps has been gaining more popularity [1]. DevSecOps is short for Development, Security, and Operations. It is considered an extension of the broader DevOps [2] movement, which aims to break down silos between development (Dev) and operations (Ops) teams to improve collaboration, speed up software delivery, and enhance the overall quality of software systems. DevSecOps takes this a step further by emphasizing the importance of security at every stage of the software development lifecycle [3].

However, with implementing DevSecOps comes many challenges. DevSecOps dictates the software development life cycle's focus area to be broad, making it hard to measure engineering success correctly. According to Llanso [4], prioritization becomes the main pain point because of the high data points. Data saturation in cyber security provides a more significant challenge driving further engineering focus in light of security incidents, the growing number of vulnerabilities exploited, business impact, and security control costs [5].

The five essential questions need to have answers at any given incident: WHO (who is the threat vector), WHAT (what is happening: exploit, threat, Ransome, etc.), WHEN (when did the incident start, timestamps of any beneficial forensics activity, etc.), HOW (in what ways a

current incident came to be), and WHY (justifications for the incident: unpatched vulnerability that got exploited, ransomware, zero-day exploit, etc.). Once a team has clear answers to any incident, it will be up to an individual (Chief Information Security Officer (CISO), Head of Security, Senior Leadership, etc.) to prioritize security with the given data.

This paper dives into a new proposed matrix that is simple, digestible, and easier to navigate than most current standards.

II. *Literature Review*

Several frameworks aim to assess the maturity of one's security in the engineering space:

- DevSecOps Maturity Model (DSOMM) by Open Web Application Security Project (OWASP)
- Software Assurance Maturity Model (SAMM) by OWASP
- Building Security In Maturity Model by Synopsis

DSOMM represents assessment as a list of checks and balances in light of individual security activities [6]. It consists of 5 dimensions and 17 subdimensions:

- The "Build and Deployment" dimension has the subdimensions "Build," "Deployment," and "Patch Management."
- "Culture and Organization" is composed of the subdimensions "Design," "Education and Guidance," and "Process."
- The "Implementation" dimension has two subdimensions: "Application Hardening" and "Infrastructure Hardening."
- "Information Gathering" consists of "Logging" and "Monitoring" subdimensions
- "Test and Verification" has the most significant number of subdimensions, which are "Application tests," "Consolidation," "Dynamic depth for applications," "Dynamic depth for infrastructure," "Static depth for applications," "Static depth for infrastructure," and "Test-Intensity."

DSOMM misses the threat assessment covered in our proposed framework in the Pre-Commit Stage.

SAMM, specifically version 2.0, has five dimensions and 15 subdimensions, each of them covering different steps in the Software Development Life Cycle [7]:

- The "Governance" dimension contains subdimensions, such as "Strategy and Metrics," "Policy and Compliance," and "Education and Guidance."

- The “Design” dimension covers the following three dimensions: “Threat Assessment,” “Security Requirements,” and “Security Architecture.”
- The “Implementation” dimension contains “Secure Build,” “Secure Deployment,” and “Defect Management” subdimensions.
- The “Verification” dimension is composed of the subdimensions “Architecture Assessment,” “Requirements-Driven Testing,” and “Security Testing.”
- The “Operations” dimension contains the subdimensions “Incident Management,” “Environment Management,” and “Operational Management.”

SAMM has several aspects that seem inconsistent in SDLC:

The “Verification” dimension that has the “Architecture Assessment” sub-dimension should be a step covered in the design phase of SDLC (covered in the proposed matrix in the Pre-Commit Stage)

- The “Operations” dimension lacks logging and monitoring activities that are essential for successful observability (covered in the proposed matrix in the Post-Deployment Stage).

BSIMM is the framework developed by Synopsis and has four main dimensions as follows [8]:

- The “Governance” dimension contains “Strategy and Metrics,” “Compliance and Policy,” and “Training.”
- The “Intelligence” dimension contains “Attack Models,” “Security Features and design,” and “Standards and requirements.”
- The “SSDL Touchpoints” dimension contains “Architecture Analysis,” “Code Review,” and “Security Testing.”
- The “Deployment” dimension contains “Penetration Testing,” “Software Environment,” and “Configuration Management and Vulnerability Management.”

While the present model is essential to consider when assessing the security of the software products, it has its flaws:

- SSDL Touchpoints dimension covers a limited number of tests needed to ensure security in SDLC
- Monitoring and Logging are missing in BSIMM altogether.

III. Methodology

While this maturity matrix could be applied to any software development team, for simplicity, we assume that this is a web application team and omit CI/CD (continuous improvement and

continuous deployment stages) on a diagram. Traditional SDLC (software development life cycle) is shown in Figure 1 below.

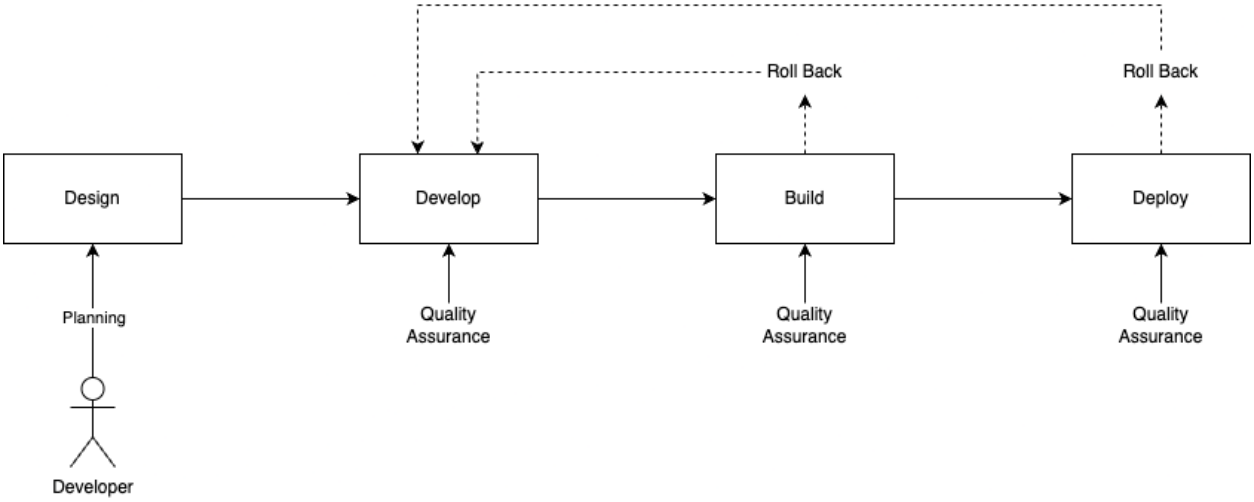

Figure 1. Software Development Life Cycle

As part of the research, we take the SDLC (software development life cycle) diagram and break it down into several stages:

1) Pre-Commit Stage. This stage covers the moment before a developer starts working on any given feature to commit.
2) Pre-Deployment Stage. This stage covers the moment after the pre-commit stage until the code is merged into the main branch.
3) Post-Deployment Stage. This stage covers the moment after a web application has been built and deployed.

The way those stages fit into the software development lifecycle can be described using the diagram in Figure 2.

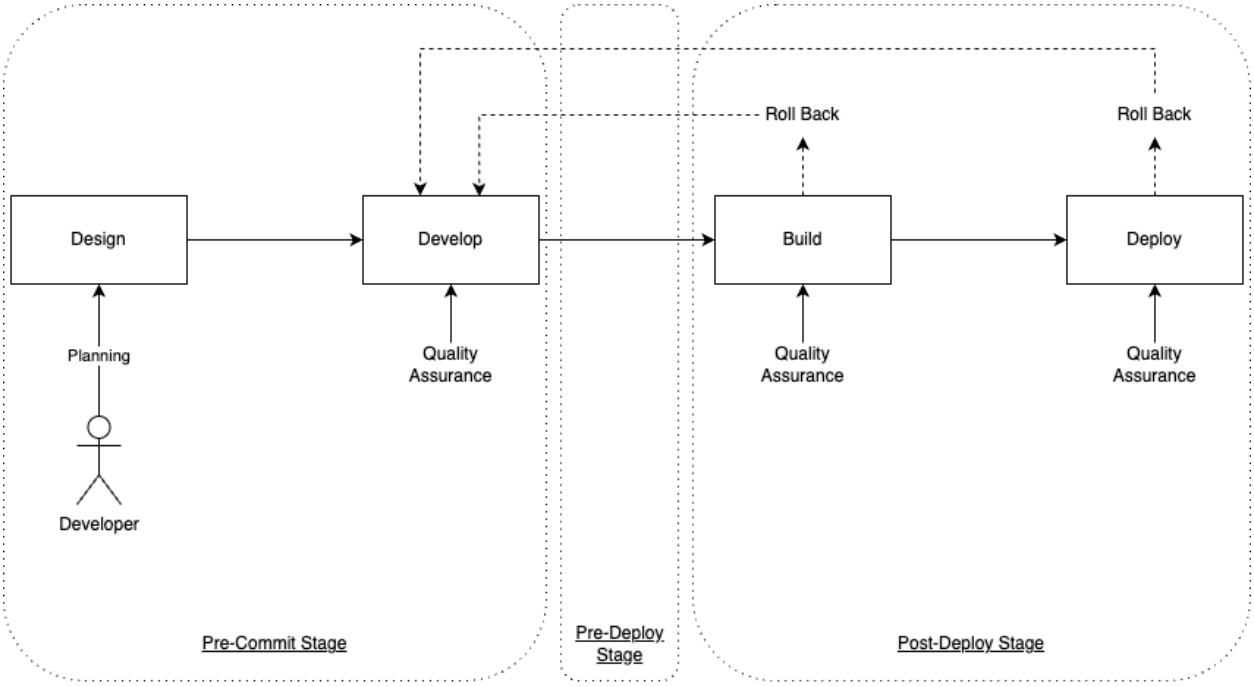

Figure 2. Pre-Commit, Pre-Deployment, and Post-Deployment stage

A developer plans to work on features in a team environment. After the design life cycle is over, software development starts. From the beginning to the time when the development cycle is over, we consider the Pre-Commit Stage. A time right after that, all the way to before changes made by a developer are deployed, will be the Pre-Deployment Stage. The last stage is the Post-Deployment Stage.

IV. *Research*

Security processes in the software development life cycle should decrease security risk by identifying vulnerabilities and threats [9].

The Pre-Commit Stage covers two processes in the diagram: Design and Development.

The design process includes a developer understanding the requirements for a feature they are working on. These aspects should be covered in that stage include threat modeling and baseline and access security controls.

The development process describes a developer working on a feature with clear requirements. Things that should be covered in that stage include SAST (static application security testing), SCA (software composition analysis), and source code review.

The Pre-Deploy Stage covers the process of committing the code a developer wrote. This includes SAST, SCA, and source code review because of possible discrepancies since a developer can bypass recommendations presented in the Development process of the Pre-Commit Stage.

The Post-Deploy stage includes two processes in the diagram: Build and Deploy.

The build process covers building the code the developer wrote, compiling, and integrating it into the appropriate architecture (containers, serverless, virtual machine, or physical server). Considerations that should be covered include containers and image scan, packages scan, Infrastructure as Code (IaC - infrastructure code security), and IAST (interactive application security testing).

The deployment process covers deploying a built code into the application. Things that should be covered here include RASP (runtime application security protection), DAST (dynamic application security testing), penetration testing, and ongoing system and network monitoring.

V. Matrix

The maturity matrix below in Table 1 represents levels of maturity based on the ability to answer any of the five questions (WHO, WHAT, WHEN, HOW, WHY):

	Pre-Commit Design (Threat Modeling, hardened environment, logging and monitoring enabled), Develop (SAST, SCA, source code review)	Pre-Deploy Assets scan (images, containers, serverless, packages), IaC, IAST	Post-Deploy Build (assets scan: images, containers, serverless, packages; IaC, IAST), Deploy (RASP, DAST, penetration testing, monitoring)
Expert	full visibility		
Advanced	WHO, WHAT, WHEN, WHY	WHO, WHAT, HOW	WHO, WHAT, WHEN, HOW
Needs Improvement	WHO, WHAT	WHO, WHAT	WHO, WHAT, WHEN
Lacking	< 2 of questions answered	< 2 of questions answered	< 3 of questions answered

Table 1. Maturity Matrix

This matrix is broken down into three stages that consist of 4 levels:

- Expert: complete visibility of all the data to be able to respond to all five questions
- Advanced: best practices in the field with some visibility limitations
- Needs Improvement: partial visibility to be able to answer urgent questions in light of an incident
- Lacking: limited to no visibility

Expert-level scoring at all three stages guarantees to pass DSOMM, SAMM, and BSIMM. While scoring high on the three frameworks discussed previously, Expert level scoring isn't guaranteed. For instance, having Maturity 4 on OWASP DSOMM in the "Build and deployment" dimension won't necessarily guarantee answering HOW in the Pre-Commit Stage. Similarly, having maturity level 3 in OWASP SAMM won't answer WHY in the Pre-Commit Stage. Lastly, missing monitoring and logging activities in the "Configuration Management & Vulnerability Management" sub-dimension in the dimension of Deployment won't answer WHY in the Post-Deploy Stage.

VI. Validation of the matrix

To claim that the current proposal of a matrix is better than the ones listed previously, we have run the assessment leveraging working industry professionals in using our maturity matrix. The key audience was security engineers and managers and directors of security departments in companies. As a result, we have gathered feedback from 12 individual contributors (either security engineers or software engineers), 8 engineering managers, and 3 directors of security departments. All of the participants came from a total of 24 companies with a profile of each company being a SAAS (Software as A Service) focused company with employers between 200-500 in size.

Main advantage of our matrix is its simplicity without jeopardizing accuracy and efficiency. After the assessment we have gathered the feedback from the participants and key findings were as follow:

- Engineering managers and directors found the matrix easy to follow
- Software engineers found ratings to be easily understood and areas proposed in SDLC made sense
- Most of the participants (18 out of 23) found that doing the matrix independently from the rest of the team would not change the results as opposed to completing the matrix altogether. That is meaningful positive feedback as most of the alternative frameworks discussed previously are intended to be done collectively within a company since results will vary per individual taking the assessment. However, the rest (5 out of 23) of the participants considered that the self-assessment would differ from person to person which

may cause discrepancy. A recommendation was suggested to elaborate each of the stages (Pre-Commit, Pre-Deploy, and Post-Deploy) with actionable key points

- The maturity matrix was received better by engineering managers and directors level participants than individual contributors for areas like simplicity and structural guidance on security planning.

VII. Conclusion and future work

In this paper, we have proposed a new DevSecOps Maturity Matrix. We have identified the gaps in other frameworks, such as OWASP DSOMM, OWASP SAMM, and BSIMM.

Gathering the results from the self-assessments represents quantifiable data regarding the security maturity of software products, which could be analyzed to make better strategic decisions, better allocate human resources, and always know where improvements need to be done in the software products. Naturally, there is space left for improvements as noted below. Thus, our proposed framework should be considered open for further development.

For future work, we have identified several areas:

- mapping practices proposed with other standards from the ISO/IEC 27001 family (such as ISO/IEC 27002/27003/27005)
- Proposing risk-based assessment to validate the current state of maturity, thus quantifying business impact per each step towards maturity
- Mapping the framework to be compatible with System and Organization Controls (SOC)
- Consider revision to the framework for industry specific software development (e.g. AI based development).

References

[1] H. Myrbakken and R. Colomo-Palacios, “DevSecOps: A Multivocal Literature Review,” *Communications in Computer and Information Science*, pp. 17–29, Sep. 2017. doi:10.1007/978-3-319-67383-7_2

[2] E. Freeman, “DevOps,” Amazon, https://aws.amazon.com/devops/what-is-devops/ (accessed Oct. 13, 2023).

[3] S. Radack, System development life cycle - NIST, https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=902622 (accessed Oct. 13, 2023).

[4] T. Llanso, “CIAM: A data-driven approach for selecting and prioritizing security controls,” *2012 IEEE International Systems Conference SysCon 2012*, Apr. 2012. doi:10.1109/syscon.2012.6189500

[5] Gitlab, “Gitlab DevSecOps 2021 survey results,” GitLab, https://learn.gitlab.com/c/2021-devsecops-report?x=u5RjB_ (accessed Oct. 14, 2023).

[6] DSOMM, https://dsomm.owasp.org/ (accessed Oct. 13, 2023).

[7] “OWASP SAMM,” OWASP SAMM | OWASP Foundation, https://owasp.org/www-project-samm/ (accessed Oct. 13, 2023).

[8] “Building security maturity model (BSIMM) consulting services,” Synopsys, https://www.synopsys.com/software-integrity/software-security-services/bsimm-maturity-model.html (accessed Oct. 13, 2023).

[9] R. Brasoveanu, Y. Karabulut, and I. Pashchenko, “Security maturity self-assessment framework for software development lifecycle,” *Proceedings of the 17th International Conference on Availability, Reliability and Security*, 2022. doi:10.1145/3538969.3543806