www.revistageintec.net
ISSN: 2237-0722
revista GEINTEC
Gestão, Inovação e Tecnologias

Effective DevSecOps Implementation: A Systematic Literature Review

Dhaval Anjaria[1]; Mugdha Kulkarni[2*]

[1]Symbiosis Center for Information Technology, Symbiosis International (Deemed University), Pune, Maharashtra, India.

[2*]Symbiosis Center for Information Technology, Symbiosis International (Deemed University), Pune, Maharashtra, India.

[2*]mugdha@scit.edu

Abstract

Adopting DevOps means increased collaboration between development and operations teams and faster release cycles through a shift to automation. Using Dev Ops brings with it several advantages in the development of software. Security, however, is often neglected in DevOps due to the fast release cycle. Therefore Dev Sec Ops has emerged as an extension to DevOps that attempts to integrate security with Dev Ops practices, which is not without its challenges. DevOps, and by extension Dev Sec Ops, represents a significant change in the culture, tooling, and processes used in software development. Therefore, when implementing DevSecOps, teams and their organizations need to be aware of the challenges it brings and how to address those challenges for a DevSecOps implementation to be effective. Literature on DevSecOps exists that outlines practices and principles to do this. This paper uses a grounded theory approach to do a systematic literature review of academic literature to find the factors that contribute to an effective DevSecOps implementation. It attempts to reconcile the challenges of DevSecOps with ways of mitigating them and the advantages that a DevSecOps implementation can bring. The paper thus outlines methods of effectively implementing DevSecOps as described in academic literature.

Key-words: DevOps, DevSecOps, Security, SecDevOps, Continuous Integration, Continuous Delivery.

1. Introduction

DevOps and its extension DevSecOps are portmanteaus of Development, Operations, and Security. DevOps and DevSecOps do not have strict academic definitions common practices that are part of DevOps are agreed upon. However, literature does refer to DevSecOps as collaboration between Development, Operations, and security. It represents a significant shift in an organization's processes. However, with such a shift, there can be challenges and impediments to an organization if it wishes to adopt DevOps [1].

DevSecOps focuses on the integration of security along with the existing aspects of DevOps. Implementing DevSecOps that uses automation very heavily and for which CI/CD is the core can be challenging for organizations. DevOps' cost to benefit relationship is not very well understood, and this is a cause for concern for management and those implementing DevOps alike. Thus, an attempt has to be made to understand the practices and principles that make for a DevSecOps implementation that is effective in the long run [2].

The contributions made with this paper are as follows: This paper uses an approach based on Grounded Theory to study existing literature about the elements of DevSecOps and how they should be implemented. A theory was developed to guide an effective DevSecOps implementation using an iterative and incremental practice of coding literature. This theory exists in the form of findings from the grounded theory-based literature review. It covers the benefits, pitfalls, challenges, and mitigations of those challenges necessary to be addressed for an effective DevSecOps implementation [3].

2. Methodology

2.1. Preliminary Research

A search string containing the keywords "DevOps," "DevSecOps," and "implementation" was conducted on IEEE Xplore and Scopus databases. Once a relevant subset was found, research notes were made for a small set of works. At first, the first author read seminal DevOps works, namely, "The Phoenix Project" (Kim 2014) and Google's SRE guidelines to create an understanding of DevOps. Once that study was completed, papers based on the search string were continued for preliminary research [4].

It was then discovered that through that study that Grounded theory would be the appropriate approach and could be suitably used for conducting the literature review. Therefore, research began on grounded theory practices and implementations [5].

2.2. Selection of Works for Grounded Theory

After the preliminary review was concluded, works were ready to be coded, shown in Figure 1.

The inclusion criteria were limited to the following keywords: "DevOps", "DevSecOps", "Challenges", "Implementation", and “Security". If a paper did not immediately relate to DevOps, Security, or an aspect of DevOps such as continuous integration, it was not considered for review [6].

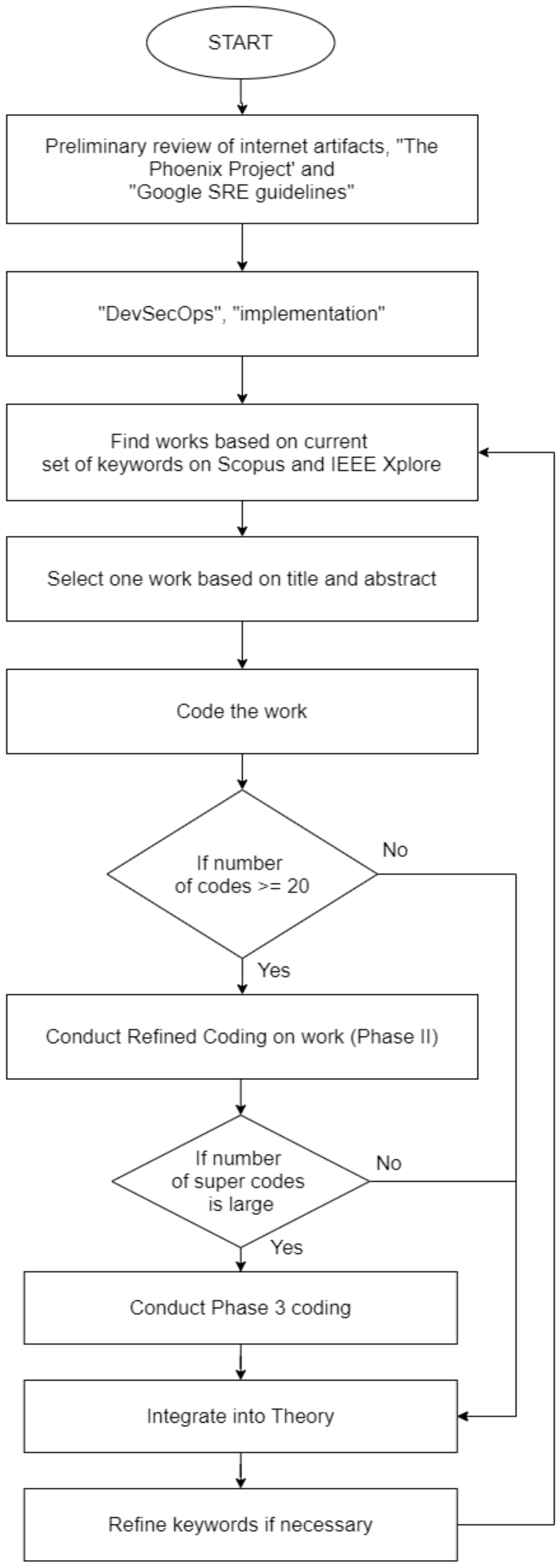
The collection of works for review also included literature such as Bird, 2016, which was used as the first work for coding. Subsequent works were coded and integrated into the theory incrementally. Thus, this paper uses 15 works that deal with DevSecOps and Security with DevOps from peer-reviewed journals in the application of grounded theory [7].

2.3. Implementation of Grounded Theory

Recommendations from Wolfswinkel et al., Stol, Ralph, & Fitzgerald, and Tie, Briks & Francis, were adopted into a customized, CAQDAS assisted coding process to form the literature review findings; hereby, referred to as “The Theory." It was found upon reviewing those works that the first step was to use "memoing" with heavy frequency throughout the process. Therefore to do this, memos were maintained in a single text file and were marked by timestamps. These memos contained observations made while reviewing individual works and notes on the methodology and how it was to be implemented.

The coding process for each work was conducted incrementally and iteratively. Each work was coded in three subsequent phases: Initial or Open Coding, Refined Coding, and Cohesive theory formation. The "constant comparison," which is an essential part of Grounded Theory approaches, was achieved by comparing codes with each other and the evolving cohesive theory that was being formed, which is explained in Figure 1. This process was repeated until "theoretical saturation" was achieved, whereby new works selected did not add to the existing theory.

Figure 1 - The Coding Process


START
Preliminary review of internet artifacts, "The Phoenix Project' and "Google SRE guidelines"
"DevSecOps", "implementation"
Find works based on current set of keywords on Scopus and IEEE Xplore
Select one work based on title and abstract
Code the work
If number of codes >= 20
No
Yes
Conduct Refined Coding on work (Phase II)
If number of super codes is large
No
Yes
Conduct Phase 3 coding
Integrate into Theory
Refine keywords if necessary

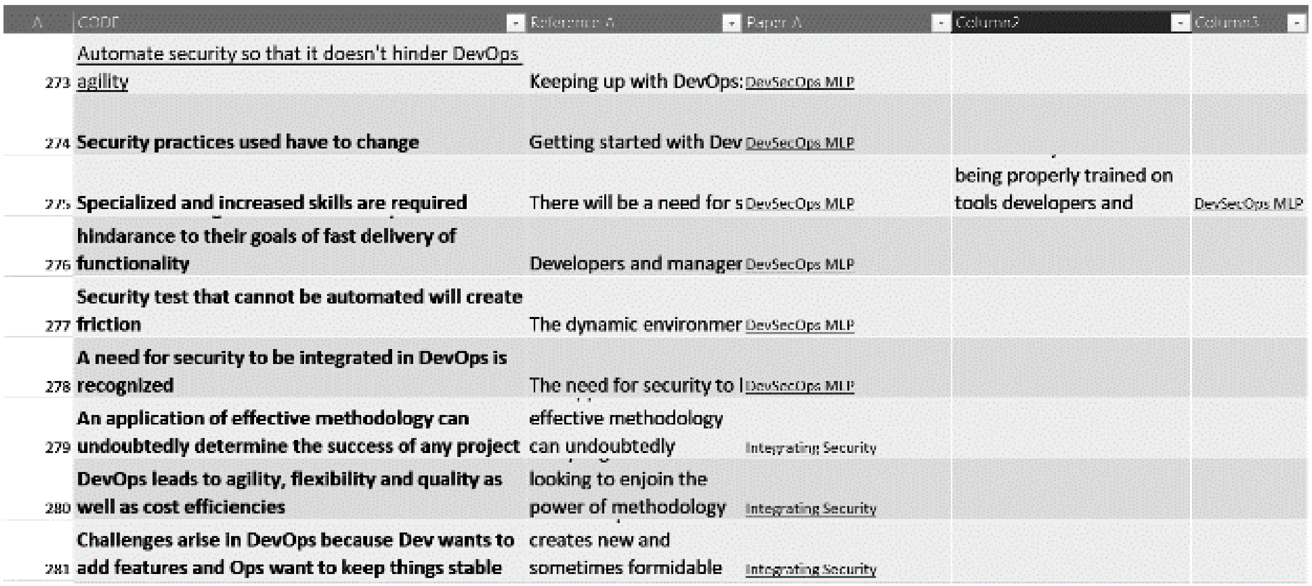
2.3.1. Initial Coding

For the initial or open coding process, each paragraph or logical grouping of sentences and paragraphs containing one cohesive idea was summarized into one sentence. These sentences were referred to as “raw codes” or “sub-codes” during the process. They were maintained in an Excel sheet with the reference text, i.e., the text that the code summarized and the work from which it was taken [8].

If an existing raw code was applied to more than one piece of text, then the text was added to that raw code rather than a new code. However, if there was even a small semantic difference between the code created and an existing one, a new raw code was assigned. For example, "Developers must understand security practices" and "Developers must appreciate security concerns" were treated as separate codes at this stage. The sections only referred to the methodology used in the paper, and those re-appearing in the conclusions were not coded. An example of this process is shown in Figure 2.

Codes for all works were maintained in a single excel sheet to achieve constant comparison. Once the entire work had been coded, a folder was created, and the codes were moved to the second phase of coding. In Figure 2, the first column represents a summary of the text in column 2. This text is raw text copied and pasted from a PDF file of the work. Column 3 represents a short name for the work, and subsequent columns house other paragraphs from other works, which can also be summarized in the same way or relate to the text in column 2 [9].

Figure 2 - Raw Codes



A	CODE	Reference A	Paper A	Column2	Column3
273	Automate security so that it doesn't hinder DevOps agility	Keeping up with DevOps:	DevSecOps MLP		
274	Security practices used have to change	Getting started with Dev	DevSecOps MLP		
275	Specialized and increased skills are required	There will be a need for s	DevSecOps MLP	being properly trained on tools developers and	DevSecOps MLP
276	hindarance to their goals of fast delivery of functionality	Developers and manager	DevSecOps MLP		
277	Security test that cannot be automated will create friction	The dynamic environmer	DevSecOps MLP		
278	A need for security to be integrated in DevOps is recognized	The need for security to I	DevSecOps MLP		
279	An application of effective methodology can undoubtedly determine the success of any project	effective methodology can undoubtedly	Integrating Security		
280	DevOps leads to agility, flexibility and quality as well as cost efficiencies	looking to enjoin the power of methodology	Integrating Security		
281	Challenges arise in DevOps because Dev wants to add features and Ops want to keep things stable	creates new and sometimes formidable	Integrating Security		

ISSN: 2237-0722
Vol. 11 No. 4 (2021)
Received: 19.07.2021 – Accepted: 18.08.2021

Usually, in grounded theory, this phase is referred to as either Selective coding or Axial coding (depending on the approach). For this paper, this phase is referred to as “Refined coding” since it draws from both approaches to reduce the number of codes used to form the theory ultimately. If works did not have enough raw codes, the raw codes were immediately used to compare with existing theory, thereby skipping subsequent phases.

2.3.2. Refined Coding

For refined coding, codes with more than one reference or are considered important (due to similarities with other codes in other works) were promoted to so-called "super-codes." They were referred to as super-codes because the remaining raw codes (referred to in this phase as sub-codes) with only one reference were to be consolidated into these super-codes.

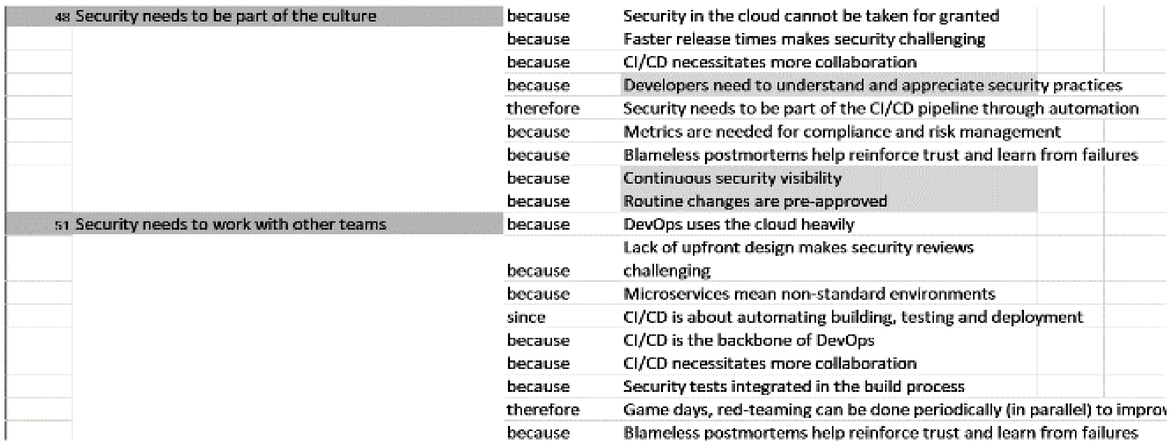
For each sub-code, a comparison was made to each super-code in order to find a relationship. This relationship was codified with words such as "because," "therefore," "given if," etc., that expressed some relationship between the two codes, which were essentially English language, sentences [10].

For example, if there was a super-code which said, "There is a trade-off between cost and added testing," where the sub-code and the relationship between them were expressed as, "because," "More complex CI/CD pipelines are harder to troubleshoot"

Relationships such as “and” and “however” were not considered since it was considered more suitable to form a new super-code because it was believed that within those relationships, the ideas expressed by sub-codes were not sufficiently explained or expressed in the existing super-codes [11].

An example is given in Figure 3.

Figure 3 - Super-codes and Sub-codes



48	Security needs to be part of the culture	because	Security in the cloud cannot be taken for granted
		because	Faster release times makes security challenging
		because	CI/CD necessitates more collaboration
		because	Developers need to understand and appreciate security practices
		therefore	Security needs to be part of the CI/CD pipeline through automation
		because	Metrics are needed for compliance and risk management
		because	Blameless postmortems help reinforce trust and learn from failures
		because	Continuous security visibility
		because	Routine changes are pre-approved
51	Security needs to work with other teams	because	DevOps uses the cloud heavily
		because	Lack of upfront design makes security reviews challenging
		because	Microservices mean non-standard environments
		since	CI/CD is about automating building, testing and deployment
		because	CI/CD is the backbone of DevOps
		because	CI/CD necessitates more collaboration
		because	Security tests integrated in the build process
		therefore	Game days, red-teaming can be done periodically (in parallel) to improv
		because	Blameless postmortems help reinforce trust and learn from failures

ISSN: 2237-0722
Vol. 11 No. 4 (2021)
Received: 19.07.2021 – Accepted: 18.08.2021

In Figure 3, column 1 represents a code with multiple references or could be considered important by the reviewer (super-codes). Column 3 contains the other codes that could relate to the super-code (column 1) somehow. Column 2 represents an expression of the "relationship" between the super-codes in column 1 and sub-codes in column 2.

These relationships did not have to be exact; that is, the words used to express the relationship did not have to form a grammatical sentence using the super-code and sub-code but was enough to express that some relationship exists. If the relationship was too ambiguous, remarks were added in.

If a sub-code did not apply to any super-code, it was promoted to a super-code. Sub-codes that also expressed a significant relationship with a super-code were highlighted. For example, for a super-code which said, "Access needs to be limited, one of the sub-codes and the relationship that was highlighted was, "because," and "Lack of upfront design makes security reviews challenging' [12].

2.3.3. Phase III Coding

The text that these codes referenced was referred to understand the nature and importance of the relationship. Once all the sub-codes were either promoted to or consolidated into super-codes, python was used to transform the data to be ingested into Power BI. Power BI was used to see various trends in the relationships, for example, which sub-codes applied to a large number of codes, and which super code absorbed the most sub-codes.

Python was also used to understand the commonality between super-codes: super-codes shared the most sub-codes and which super-codes were most related to other super-codes. For example, this process was not used for all works, for example works that did not have enough super-codes [13].

2.3.4. Theory Formation

Once these major super-codes were identified, the process of forming the theory was started. This theory was a prosaic amalgam of the final super-codes created thus far. Super-codes that remained after the above phases were completed were integrated into the existing prose in a way that could be cohesive and readable, which was also how ultimately constant comparison was conducted. Once the theory had been modified, other works were selected, and search strings were modified to include keywords, such as "security," "continuous integration," and “continuous delivery” [14].

3. Results and Discussion

This section contains the final cohesive theory that was formed. It follows thus:

3.1. Why DevOps?

The aim of implementing DevOps is to have more collaboration and create empathy. It is also to reduce times between releases to improve customer satisfaction. Therefore to do this, there has to be a culture of openness. DevOps' implementation can vary greatly; for example, a "DevOps team" can be a separate team altogether, responsible for some code entirely (from development to production. DevOps attempts to solve problems by improving sharing and collaboration between teams because collaboration can help fix problems faster since sharing means teams have to know about the problems other teams have and fix them.

DevOps provides agility, flexibility, and cost-effectiveness. DevOps does not require specialized tooling, and teams can be doing DevOps without explicitly saying so. Challenges arise in DevOps because the priorities of Dev and Ops do not always align [15].

3.2. Security and DevOps

The more unrestricted collaboration is, the more lapses in access control can occur. Security is often neglected in DevOps due to the fast nature of the release cycle. DevSecOps can be defined as integrating security protocols into DevOps. DevSecOps is about introducing security earlier in the software development lifecycle so that vulnerabilities in the product and the cost to fix them are reduced. Not doing so will cause delays arising from a need to fix vulnerabilities discover when security testing is ultimately done on a project, which is usually towards the end?

Adding security to DevOps can be challenging, and this process can be enabled using security monitoring [16].

3.3. Security Risks in DevOps

Using DevOps increases security risk because of the increased collaboration that takes place between teams. All the communication and tool they use cannot be audited every time they use them. In fact, because of increased collaboration and faster release cycles, security can be overlooked.

ISSN: 2237-0722
Vol. 11 No. 4 (2021)
Received: 19.07.2021 – Accepted: 18.08.2021

Access needs to be limited because certain security practices like manual audits, separation of duties, upfront reviews are difficult. These processes need to be done iteratively. Whichever processes can be automated must be added earlier in the coding cycle, like SAST practices, automated audits, etc. Whatever processes that cannot be automated must be done in parallel at various times parallel to the build process; not only does this help improve the security of the application, but at the same time, security issues can be communicated sooner in the lifecycle of an application, which means they can be fixed faster [17].

All routine changes should be automated. CI tools with more granular security options should be used to mitigate the amount of security exposure that collaboration brings. However, it is important to note that troubleshooting building without sufficient access can be harder if these granular options are not properly.

3.4. Reducing Risk through Automation

DevSecOps is about using security knowledge and tools to improve the development pipeline. Automating security and testing can contribute positively to a system's security. For example, automating security practices reduces mistakes, leading to vulnerabilities and reducing the amount of effort security engineers need to put in to add controls after the fact. Automating security also makes it fast and scalable, so security being overlooked to ensure rapid development can be addressed. Shifting security left in such a way makes it more cost-effective because it helps find bugs sooner helps at an earlier stage, making it easier to fix them.

Automating security should also be done so that the agility of DevOps is not sacrificed for security assurance. Automation ensures that at least some tests will be performed, which can solve security being neglected [18].

DAST and IAST can be used to look at components and their software while running. However, this is not being automated, and organizations are not aware of how much they lack automating security.

3.5. Accounting for Manual Processes

Manual security practices can be performed outside of deployment, and manual security checks should be minimized, as observed by Bird. Some security practices can be integrated before deployment, like manual code reviews, which can be done before deployments. Others like risk assessments and threat modeling, and red team exercises can be done iteratively. Manual checks can take time, but standardization reduces manual security work.

3.6. CI/CD

CI/CD is the core of DevOps, and it relies on automation. CI/CD also makes safer deployments through feature toggles, canary releases, and automated security practices. If something does slip the checks, fixes can be integrated a lot faster. DevOps is often implemented to reduce lead time to change [19].

However, it should be noted that as CI/CD pipelines get more and more complex, they can be harder to manage. The added testing, both in terms of security and otherwise, can significantly add to the complexity of the CI/CD pipeline and not increase build times. Security tests can increase build times even further.

Security tests still have to be integrated into the CI/CD pipeline. Otherwise, it can lead to more vulnerable software.

The security practices that have to be either automated or conducted in parallel and ops responsibilities that come with DevSecOps make developers feel that it creates more work for them.

3.7. Using CI/CD to Improve Collaboration

CI/CD necessitates automation. Automation can be monitored. This monitoring data is made visible to all, aligns the incentives of various teams, as observed by Mansfield-Devine, enabling collaboration.

3.8. The Advantages of More Frequent Releases

Fast integration also allows for faster feedback, which can improve productivity and lead to faster improvement. Finding errors faster can also make risk management easier and more cost-effective because fixing those errors can be cheaper if done early.

3.9. Addressing Security Concerns in the Fast Release Cycle

All of this automation means that risk assessments, compliance, etc., need to be done differently in DevSecOps. There is no large upfront design that can be assessed for security. Therefore, risk assessment and threat modeling should be done iteratively and incrementally.

Therefore, you need to keep track of the attack surface and risks with every new build; therefore, a dynamic tool pipeline is necessary because the code and its environment are constantly changing.

Security needs to be part of the coding cycle, and teams, especially developers, need to understand security needs. There has to be a continuous feedback system in terms of security to other teams and work with them to perform security tests and implement security policies. Security policies can be codified as codes.

3.10. Micro Services

DevOps and Micro services are well suited for each other. However, Micro services mean more security needs because a running system is more complex than a static list of Micro services. However, different Micro services teams still have to greatly communicate with each other and decide how their services will interface with each other.

3.11. Security Training

To implement DevSecOps, however, security needs to work with other teams. A lot of there has to be cross-training; developers and ops teams need security training. Not only is cross-training required, but personnel with "DevSecOps skills" that is an understanding of development, security, and operations are often required, and they can be hard to find. DevOps is a constant cultural process that an organization must practice. Training developers to code securely is easier than training security people to code.

3.12. Securing and Standardizing the Underlying Infrastructure

Security should very much be part of the entire development process. It begins with securing the underlying tools and infrastructure first. To better secure the underlying infrastructure and streamline the CI/CD process, the tools used have to be standardized across different servers.

Standardization has to be done for tools and processes, which CI/CD helps achieve. This process standardization ensures that at least some test coverage will be there, and fewer builds will be broken. Standardization will also help reduce manual security work.

Adding security tools in the process can be challenging but beneficial in the long run. Security tools and standards, such as SAST, OWASP, must be appropriate for the organization that uses them and the applications these organizations build, which can sometimes be a sizeable task.

3.13. Special Skills are Required

Managers and other departments also need training in handling DevOps teams that deploy fast and house a wider variety of skills. Since risk assessment, threat modeling, and security policies are implemented differently and earlier in the development cycle, some specialized skills, such as good monitoring and logging practices, security automation, etc., are required. A lack of experience with these tools and practices can be challenging.

3.14. Shifting Security Left

Security needs to "shift-left." Security needs to be part of the coding cycle, and teams, especially developers, need to understand security needs. There has to be a continuous feedback system in terms of security to other teams and work with them to perform security tests and implement security policies.

Shifting security left makes it easier to plan and execute security practices. SAST can assist in this process. Security assessments and issues will only be fixed if they are conveyed to the development teams. Shifting security left means starting from the non-functional requirements. It means discovering issues and bugs at an earlier stage which is more cost-effective.

3.15. Monitoring

Monitoring in organizations can be complex system data, although being granular and scalable can still have gaps. For example, cultural changes, configuration drift can often not be captured through system data alone. Data gathered from surveys help understand the real-world impact of the system and fill the gaps in system data. Therefore, monitoring requires a complementary approach using both system and survey data. Good monitoring, however, is necessary but insufficient. There needs to be cross-talk between teams in an organization. That said, in terms of monitoring, less useful data must be minimized because it could be that much harder to learn.

3.16. Teams Need to Work Together

For there to be buy-in in security among developers, operations, and management, security information should be conveyed to everyone. However, there needs to be a mapping between the security risks identified and the business risks they pose.

Teams, of course, need to work together to achieve common goals and to do this, data and information should be as visible as possible. Teams should not hide information from one another or even the customer. Conversely, customer satisfaction data should also be shared back to dev and ops teams.

Getting teams to work together with new tools and techniques can be challenging; so does integrate the tools and the accustomed processes of each team. For DevOps to be successful, the priorities and goals of various teams involved have to be aligned. A common security mindset can be created using metrics that are visible to everyone.

3.17. DevSecOps and Culture

A DevOps or DevSecOps team responsible for maintaining a system needs to have both the authority and responsibility for their system. The culture required for DevOps must promote shared ownership. The team has to be responsible for the application; teams need to be responsible, but they also have the authority to manage their service. There must be a culture that promotes learning from failure, which means that logs and monitoring data need to be visible and traceability of code needs to be put in, which improves compliance.

4. Threats to Validity

Some of the information may be lost in the process and not captured by the resultant theory. For example, one code, "State of DevOps report states that high performers deploy several times faster," appears in several works. However, since it is only referenced once per work and is usually absorbed by a super-code, it is not included in the theory. Secondly, the works chosen for coding were limited to the Scopus index and IEEE Xplore. Internet artifacts and Google Scholar were not used to source works for coding.

5. Conclusion

A grounded theory-based approach was used to build. The findings represent that DevSecOps involve changes that are both cultural and technical. The way organization builds software has to change, making significant use of automation. Any task related to security, compliance, and assurance can be automated and integrated into the CI/CD pipeline. Once the CI/CD pipeline is standardized, other organizational changes follow; for example, it gives developers faster feedback, which gives better results due to the changes being tested in running systems.

Monitoring of these systems also represents a challenge. It demands a complementary approach using both log data and survey data to get real-world feedback. The information generated through monitoring needs to be visible to all parties involved; doing so will provide visibility to all parties involved and might help align the incentives for the various teams in DevSecOps, which in turn helps collaboration.

However, these practices increase the security exposure of the system, which can be mitigated by securing the underlying tooling, standardizing the tooling, and doing manual checks parallel to the build pipelines. Access control must also be managed through the use of granular configuration in CI/CD security tools.

DevSecOps will require cross-training to be implemented properly since security needs to shift left, which means security has to be integrated into the coding cycle using SAST tools and into the CI/CD pipelines using DAST and IAST tools.

References

Ahmed, Z., & Francis, S. C. (2019, November). Integrating Security with DevSecOps: Techniques and Challenges. *In 2019 International Conference on Digitization (ICD; pp. 178-182). IEEE.*

Bass, L. (2017). The software architect and DevOps. *IEEE Software, 35*(1), 8-10.

Bird, J. (2016). *DevOpsSec: Securing software through continuous delivery.*

Chun Tie, Y., Birks, M., & Francis, K. (2019). Grounded theory research: A design framework for novice researchers. *SAGE open medicine, 7,* 2050312118822927.

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.

Erich, F. M. A., Amrit, C., & Daneva, M. (2017). A qualitative study of DevOps usage in practice. *Journal of Software: Evolution and Process, 29*(6), e1885.

Forsgren, N., & Kersten, M. (2018). DevOps metrics. *Communications of the ACM, 61*(4), 44-48.

Hilton, M., Nelson, N., Tunnell, T., Marinov, D., & Dig, D. (2017, August). Trade-offs in continuous integration: assurance, security, and flexibility. *In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (pp. 197-207).

Kim, G., Behr, K., & Spafford, K. (2014). *The phoenix project: A novel about IT, DevOps, and helping your business win.* IT Revolution.

Kromhout, B. (2018). Containers will not fix your broken culture (and other hard truths). *Communications of the ACM, 61*(4), 40-43.

Lee, J. S. (2018, October). The DevSecOps and agency theory. *In IEEE International Symposium on Software Reliability Engineering Workshops* (ISSREW; pp. 243-244). IEEE.

Mansfield-Devine, S. (2018). DevOps: finding room for security. Network Security, 2018(7), 15-20.

Mohan, V., & Othmane, L. B. (2016, August). SecDevOps: Is it a marketing buzzword?-mapping research on security in DevOps. *In 11th International Conference on Availability, Reliability, and Security* (ARES; pp. 542-547). IEEE.

Myrbakken, H., & Colomo-Palacios, R. (2017, October). DevSecOps: a multivocal literature review. *In International Conference on Software Process Improvement and Capability Determination* (pp. 17-29). Springer, Cham.

Prates, L., Faustino, J., Silva, M., & Pereira, R. (2019, September). DevSecOps Metrics. *In Euro Symposium on Systems Analysis and Design* (pp. 77-90). Springer, Cham.

Smeds, J., Nybom, K., & Porres, I. (2015, May). DevOps: a definition and perceived adoption impediments. *In International Conference on Agile Software Development* (pp. 166-177). Springer, Cham.

Rahman, A. A., & Williams, L. (2016). Software security in DevOps. *Proceedings of the International Workshop on Continuous Software Evolution and Delivery - CSED '16.* doi:10.1145/2896941.2896946

Stol, K. J., Ralph, P., & Fitzgerald, B. (2016, May). Grounded theory in software engineering research: a critical review and guidelines. *In Proceedings of the 38th International Conference on Software Engineering* (pp. 120-131).

Wolfswinkel, J. F., Furtmueller, E., & Wilderom, C. P. (2013). Using grounded theory as a method for rigorously reviewing the literature. *European journal of information systems, 22*(1), 45-55.