Automating Security in a Continuous Integration Pipeline

Sohrab Chalishhafshejani[1], Bao Khanh Pham[1] and Martin Gilje Jaatun[12][a]
[1]*IDE, UiS, Stavanger, Norway*
[2]*SINTEF Digital, Trondheim, Norway*
{*s.chalishhafshejani, bk.pham, martin.g.jaatun*}*@[stud.]uis.no*

Keywords: DevSecOps, DevOps, Software Security, Cyber Security, Continuous Integration

Abstract: Traditional approaches to software security are based on manual methods, which tend to stall development, leading to inefficiency. To speed up a software development lifecycle, security needs to be integrated and automated into the development process. This paper will identify solutions for automating the security phase into a continuous software delivery process, integrating security tools into a Github repository by using Github Actions to create automated vulnerability scanning workflows for a software project.

1 INTRODUCTION

Aggressive competition in software companies has led to shortening applications' time-to-market intervals in order to receive feedback and evaluate the market faster before further development. As a result, agile software development has been introduced to bring the development team closer to business teams. This model tries to add new features on each iteration of the software lifecycle and build up features optimized based on end-user needs (Kumar and Goyal, 2020).

In recent years, software development has been shifting from delivering software as a product to a Software-as-a-Service principle. Hence, the software has been shifted from on-premises servers to cloud solutions. This enables providers to deploy new software features in short intervals without being concerned about backward compatibility issues (Myrbakken and Colomo-Palacios, 2017).

Traditionally, the development team was responsible for creating logic and core features of the program, and the operation teams handled deploying the app on the server/cloud. The two parties were normally only connected via ticketing systems, and problems along the way required sending and receiving several tickets between the development and operations team. The solution was to merge Development and Operations into the DevOps paradigm, where operations traditionally handled by the operations team such as building and testing are automated, resulting in shorter time to deploy changes, achieving continuous development and delivery. Companies like Google and Microsoft first stepped into the game and introduced pre-made DevOps utilities and software (Leite et al., 2019).

In contrast, there are several trade-offs to this approach of software delivery. DevOps engineers try to push code batches in short intervals and it will force security teams to review the newly generated code faster. Traditionally (Howard and Lipner, 2006) all security analysis procedures like vulnerability scanning and code review were done manually by the security team (Kumar and Goyal, 2020), resulting in either delayed releases or insufficient security if misconfigurations, hardcoded passwords, or other dangerous concerns slip through the controls.

There is a need for faster security checks and automated testing systems which can reduce the amount of manual work needed to be performed by security teams. Therefore, the DevSecOps concept was born to effectively combine DevOps and Security; integrating an automated continuous security model with a regular Continuous Integration and Continuous Delivery (CI/CD) pipeline through vulnerability scanning tools. These tools are added to the many phases of the software continuous delivery pipeline.

In this paper, we are going to look for possible ways to automate software security checks in the DevSecOps procedure by building up a workflow from security tools. This workflow has been tested with three open-source projects and one real-life project from the development organization that serves as our case study (referred to in the following by its pseudonym "ACME"). This will help compa-

[a] https://orcid.org/0000-0001-7127-6694

nies to scale up security in the whole DevOps process with a one-time effort by changing the pipeline and integrating security into it.

2 Motivation

Despite the successes of the Microsoft Security Development Lifecycle (Howard and Lipner, 2006) and the focus that has been afforded software security in the last decade and a half (McGraw, 2006; Williams et al., 2018), the average developer still does not possess the requisite skills for developing secure software (Oyetoyan et al., 2017). Traditionally, the need for strong software security measures has not been high enough on management's radar to convince them to invest more in it. Manual security procedures are prone to human error, and are also slowing the whole speed-to-market requirements of modern application development environments.

2.1 Problem Definition

The real question that arises here is what do we have to automate? There are several phases in software development that each can be a target by malicious actors. It is recommended that organizations look back at their whole infrastructure and try to understand where they can apply security measures in their DevOps environment. In addition, the infrastructure of applications is now more volatile since it is shifted more to Cloud Native apps and infrastructure as a service (IaaS) platform. These security measures must contain ways to ensure security. Containerized environments and microservice development also create new concerns which must be considered.

2.2 Use Cases/Examples

Despite some attention from the industry in recent years, the amount of academic research on DevSecOps is still meager (Jaatun et al., 2017; Rajapakse et al., 2021). However, there are quite a few practitioners who have researched, experimented, and published their gray literature reports (white papers, blogs, articles, etc.) (Myrbakken and Colomo-Palacios, 2017; Mao et al., 2020). Besides, there is a growing choice of tools for companies to build their own pipelines. All major service providers strive to build and deliver the best platforms and frameworks for their customers, and security companies try to create tools that can be integrated with those platforms. Major platforms include Github Actions, Amazon Web Services, Microsoft Azure, and Gitlab. Also, some companies choose to combine different tools and platforms or develop tools for internal use.

2.3 Challenges

Although promising to bring practical results, the application of automated security testing in the CI/CD pipeline in practice also faces certain difficulties. Below are the difficulties in applying DevSecOps mentioned by Myrbakken and Colomo-Palacios (Myrbakken and Colomo-Palacios, 2017).

- The new DevSecOps process must match the DevOps process; automated security checks have to be integrated into the existing CI/CD pipeline and have to be truly efficient at its speed.
- Volatility implies changes to:
  - Techniques – Security engineers must understand the DevOps process and developers must learn basic security skills and standards.
  - Process speed – Security scanning tools are time-consuming for each new build.
  - Culture – There will be a change in the culture of working at the company as security teams and product development teams combine. In addition, the understanding of security must also be disseminated to all other departments to get a truly secure process.
  - Standards – New security standards will be applied and updated continuously.
- Choosing the right tool for each platform is also a job that requires attention. Good tools are not necessarily suitable for the platform used by the company if they are not effectively integrated. Besides, not all tools are available to be integrated. Some tools have to be created or redeveloped from existing open-source code.

3 DEVELOPMENT LIFECYCLE

Software development lifecycle (SDLC) is the process of a software project being developed and operated. It usually involves major stages such as planning, implementation, testing, operation, and maintenance. Some of the oldest and most popular SDLC models include the waterfall model or the V-Model (Balaji and Murugaiyan, 2012). However, along with the development of technology, those old models were outdated and no longer suited to the needs of the software development industry. At that time, Agile was introduced as the preeminent software development model, and later, other models were gradually developed by researchers and engineers.

3.1 Agile Software Development

The Agile development story started when a group of software consultants signed the Manifesto for Agile Software Development in 2001 (Rehkopf, 2001). Traditional methods of software development like the waterfall method did not take into account the unpredictability aspect of the development environment (Rao et al., 2011), focusing on sequential work iterations and preparing requirements for each part of the design before moving to the next part. Testing teams were only involved in the final phase of development and problems were hidden until the testing phase. This method was not flexible to market and business. Any changes had to be completely researched and put in the next release of development (Balaji and Murugaiyan, 2012). On the other hand, the Agile method was introduced and it has been the leader for many years. Agile puts user feedback in front and welcomes changes from the market or users at any time during the development procedure.

Agile meant that software development cycles were reduced and finished products could be shipped faster. This is beneficial to companies, since products are exposed to users faster and feedback could be gathered to polish the current development process based on user needs. Recent years have seen the introduction of agile variants such as Kanban (Huang and Kusiak, 1996) and Scrum[1]. Kanban focuses more on maximizing efficiency and reducing work currently in progress by each team based on their capabilities. The Scrum method tries to create short development runs called Sprints to create software and evaluate it at each iteration (Rehkopf, 2020). These methods are the foundation of software delivery lifecycles and will play a big role in software development companies.

3.2 DevOps Methodology

The first Agile real-world implementations (Rehkopf, 2001) were mainly concerned about how it is possible to improve the overall development experience. However, developers were only focusing on code delivery at the time. The path for the codebase to reach clients was taken care of by the operation team. These two teams were incomplete with different pathways of delivery ideology. Development teams were based on swift actions and changes while the operations team only focused on stability and predictability of software changes. DevOps is a unique software delivery methodology that focuses on principles and practices to bridge the gap between development and operation teams, with continuous feedback and response pipelines which will also result in reduced software development cycle time (Jabbari et al., 2016).

The main difference between Agile and DevOps is that DevOps heavily focuses on collaboration between Development and Operations teams. In addition, it has been creating standards for pipelines and automated delivery methods which helps developers to publish their code into production with less hassle. This will lift the workforce from operation teams and make them available to monitor the product and work closely with the development team to fix problems along the way in production (Leite et al., 2019). Automation is one of the key roles in DevOps practices and should be applied whenever and wherever possible. There are several areas in which DevOps have been constantly focusing on improvement which are mentioned below.

3.3 DevOps Focus Areas

In the previous section, we have identified DevOps principles like increased deployment frequencies and reduced time to market. Achieving these goals requires fundamental changes in several areas in the software development environment.

3.3.1 Team Collaboration

In a software project, the collaboration between teams is a key to keep products stable since changes are implemented fast and on the go. Reducing delays and communication gaps is important, and can only be applied by new tools and cultural change in the whole software company. Collaboration software enables development and operation teams to go beyond emails, physical meetings, and regular talks and bring on a new level of connectivity (Hegde and Singh, 2020). Tools like Slack, Jira, Trello, and Codesourcer are examples of collaboration technologies that are widely used in industry to connect teams that are sometimes even geographically far away from each other. However, they should be accompanied by a reform in the mindset of teams to include cooperation and teamwork culture in the employees. Regular feeds from different phases of application development like unit testing and code analysis enable all parties to detect and solve issues faster.

3.3.2 Automation Wherever Possible

In practice, phases of an SDLC are often continuous. The lifecycle is repeated over and over again, and steps like planning, developing, building, testing, or deploying the software are continuous cycles (Virmani, 2015). When applying DevOps to those stages,

[1] https://www.scrum.org/resources/what-is-scrum

one of the most important things we need to pay attention to is optimizing the performance of the work. To do that, the automation of repetitive steps without the intervention of engineers is essential. Processes like building, testing, and deploying often repeat in large numbers, even with very small changes to the code. To handle these phases manually, the developers and operators will take a lot of time to complete. Therefore, DevOps encourages automation wherever possible. Automation saves more time during repetitive tasks, such as building and testing newly added code or modified old code (Ebert et al., 2016). By using automation, engineers can have more time for other stages that cannot be automated, such as planning, coding, or debugging.

3.3.3 Monitoring

Since automation is a critical DevOps goal, monitoring is indispensable for automation to follow its exact trajectory. Monitoring the system to make sure the system, pipeline, and tools are working as they should be. Once a problem occurs at any stage, it's easier and more efficient to resolve it with a carefully monitored system (Schlossnagle, 2018). In fact, software logs are often cumbersome and confusing, which makes it difficult for engineers to analyze and process them. However, quite a few engineers today still use purely manual debugging tools, which results in a significant reduction in productivity. With automated assistive tools, monitoring and measurement can be better accomplished (Ebert et al., 2016). System monitoring and automation are interrelated, where monitoring makes automation more accurate, while automation makes monitoring faster.

3.4 CI/CD Pipeline

The CI/CD pipeline, which has the basic components Continuous integration (CI) and Continuous Delivery (CD), is basically an Agile-based pipeline for SDLC optimization. The CI/CD pipeline is intricately constructed to ensure phases of software development can be continuous. In recent times, the CI/CD pipeline has gradually become an important component in software development, making SDLC more flexible, more efficient, and faster. Finally, with the rise of cloud solutions and big cloud providers releasing command-line tools for deploying applications, continuous deployment has been added to the SDLC as a final step on the pipeline. Here we focus on each of these phases in detail.

3.4.1 Continuous Integration

CI is the process that allows software developers to integrate new code into the original repository as well as share them throughout the workflow. Along with that, CI automation also allows detecting any error at an early stage to commit the problem to be solved immediately when it occurs (Virmani, 2015). When the new code is merged with the existing repository, a new version will be activated. After the build is completed, test runs are automatically performed against the build to ensure nothing goes wrong. The integration is continuous (making it to be the "C" in CI). The build automatically verifies the code every time the developer pushes their changes to the repository. Therefore, development teams may determine problems early and have time to come up with solutions.

3.4.2 Continuous Delivery

Inspired by distributors and deliveries, CD is a software engineering approach based on software production in short cycles, which makes it easy for publishers to test, build, and deploy regularly. At the same time, it also reduces costs and risks when changes occur. CD is considered as an extension of the CI, and is the regular code upgrade to ensure the quality assurance (QA) (Chen, 2015). The CD phase occurs at the end of the CI cycle and is responsible for the automatic distribution of the integrated code from the development stage to the production stage (Virmani, 2015). CD is not only tasked with automatically sending the integrated code, but also ensuring the code is sent with no errors or delays. This phase helps developers to incorporate new code into the main branch with a high degree of consistency. The CD part of the cycle is also responsible for checking code quality and performing checks to ensure a functional build can be released into the production environment.

3.4.3 Continuous Deployment

CI/CD process made massive changes in codebase manageable and possible during the daytime. Continuous deployment is another "CD" with a purpose beyond continuous delivery (Shahin et al., 2017). Continuous deployment tries to deliver these changes to end-users at a more accelerated speed. This approach tries to automate the deployment process and deliver up to hundreds of deploys in a day. Currently, tech giants like Facebook and Flickr have adopted this method. Software as service solutions and API-Driven(Goteti, 2015) software facilitates projects to have daily updates hidden from the end-users (Savor

et al., 2016). In conclusion, continuous delivery releases software to deployment as soon as the tests have passed in development. It will make features time to market even lower than before.

4 SECURITY IN SOFTWARE DEVELOPMENT

When it comes to software security, there are many possible approaches, one of them is the security of the product itself which may contain the issues that the code and builds carry full security features. To ensure this, engineers can implement a variety of methods such as periodically scanning for vulnerabilities in software, both static and dynamic code; constantly checking and updating libraries or dependencies; make sure not to use sensitive hard-coded variables; etc. Essentially, product security is to ensure that products that are deployed will not create vulnerabilities that could compromise the system on which the product is installed. On the other hand, security problems of the development process should also be considered. Since systems used to develop products may be subject to attacks from cyber criminals or adversaries, engineers must ensure thorough construction of a pipeline or SDLC. This system must be considered carefully in terms of security such as confidentiality, availability, and integrity. In general, the two views above are similar to the two sides of the coin, they must coexist and support each other so that the product can be considered secure (Assal and Chiasson, 2018; Talukder et al., 2009).

4.1 DevSecOps

In a traditional software development process, securing a product was often done independently by the security team, separate from the development team. However, a prerequisite for DevOps is speed, the combination of the development team and the operation team and leaving the security team to work separately does not meet the needs of the industry. Hence, in recent times, the DevSecOps concept was posed as an upgrade for DevOps, when it was aimed at integrating additional security into DevOps properties (Myrbakken and Colomo-Palacios, 2017).

DevSecOps arise with the requirement of secure output from the DevOps process. However, it is challenging in real-world implementations to introduce security to the DevOps process. Firstly, there are several toolsets available for DevSecOps. This brings inconsistency between each organization's implementation. Operations teams may be proficient in different programming languages and tools written based on them. There is currently a lack of standard DevSecOps implementation specifications. This leads to different opinions between operation teams with other parts of the organization. In addition, new DevOps security tools have to be tested themselves to ensure the overall security of the organization (Rajapakse et al., 2021).

4.2 Security Requirements Of The Organization

Security baselines are a group of pre-defined configurations and checks which ensure that the development environment complies with companies' overall security policies. These policies are established by well-known tech giants or even governmental organizations to keep businesses and companies safe from cyber threats. For instance, "Microsoft Windows security baselines"[2] is an example of security baselines provided by Microsoft corporation.Companies can have different definitions of security baseline in their environment. For instance, a company providing an online Application Programming Interface (API) for travel ticket booking may have a lower security baseline compared to a financial company handling sensitive transaction data. Security baselines can be broken into sections and applied as release gates. Release gates are critical checkpoints in an SDLC. These gates halt specific software release processes in case of reaching a security weakness threshold (Chung, 2018). Introducing several release gates at once will exhaust developers by lots of failed builds and releases. They have to be introduced gradually while the development team gets to know the new workflow of the pipeline. It is possible to integrate release gates in several SDLC stages like design, coding, testing, and deploying. As a result, these security checks can improve overall software quality and security in long-term usage.

4.3 Where To Start

Defining a security testing baseline starts by establishing minimum expectations from tests to be valid. Several industry-standard references have been already working on application security baselines in the past. The Open Web Application Security Project (OWASP) is the leading foundation supporting open source software security projects. The OWASP Application Security Verification Standard (ASVS)[3]

[2]https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-security-baselines

[3]https://owasp.org/asvs

project tries to document several in-depth verification steps to ensure application security at different security baselines. ASVS consists of 3 levels which makes it suitable for different companies with different security requirements. In this paper, we are trying to cover basic ASVS recommendations which can be covered by automated tests. This includes code scanning, static and dynamic security testing, and monitoring.

5 INSERTING SECURITY IN THE CI/CD PIPELINE

It is possible to insert automated security measurements to improve and assess the security of software before it reaches the users. The following sections present different parts of the pipeline which can be strengthened by security measures.

5.1 IDE Plugins And Linters

Text editor applications are being used every day by developers to produce and commit code. Error-checking tools created as a plugin for text editors are commonly called linters. E.g., the Visual Studio code editor maintains several linters like ESLint[4] and SonarLint[5]. When using linters, several bugs and security issues are prevented even before reaching the pipeline. In addition, it is possible to integrate linters in the pipeline process. Each alternative has advantages and disadvantages. Plugin linters will reduce the pipeline running time because it has already fixed parts of issues before reaching the pipeline, and it is less likely that the pipeline fails.

5.2 Static Code Analysis

Static Code Analysis (SCA)tries to discover vulnerabilities while the code is not in running mode. This type of analysis is often associated with white-box analysis due to access to the source code of the application. SCA uses methods like Taint Analysis (Kurniawan et al., 2018) and Data-Flow Analysis (Kronjee et al., 2018). Taint analysis tries to detect patterns related to injection vulnerabilities. It tries to identify tainted variables and traces them to possible vulnerable functions known as a 'sink'.

[4]https://eslint.org/docs/about/

[5]https://www.sonarlint.org/features/

5.3 Dynamic Security Testing

Dynamic Application Security Testing (DAST) tools try to have a black box view of the application. There is no prior knowledge about the codebase or database design for these tools while running. The goal is to simulate and assess conditions in real life when the software is exposed to the network. This type of tool is developed to be running on already deployed apps in a simulated production environment. Organizations normally deploy software in specific environments via containerization applications like Docker[6] and Kubernetes[7]. Containerization tools will give a suitable environment for the app to run over particular firewall and storage configurations. DAST tools try to access the application deployed on containers over ordinary network connection means like HTTP requests or database connection requests (Sönmez and Kiliç, 2021). Requests to the application can be tweaked to do a stress test on the whole application. Scenarios such as Denial of Service attacks and other known attacks can be simulated. Zed Attack Proxy (Bennetts, 2013) and Burpsuite[8] are examples of dynamic application security tools. DAST tools consist of mechanisms to bypass some security measures to go deeper into the scanning process. For instance, it is possible to bypass authentication by injecting authentication data in requests. As a result, organizations can estimate their software resistance to known attacks in case somebody acquires credentials and bypasses the authentication checkpoints.

6 PROPOSED SOLUTION

We aim to improve the software security pipeline in 3 main areas; development, pre-deployment, and post-deployment phase. The goal is a solution that has minimum configurations per repository, and can easily reproduce outcomes on different pipelines. This is important in the later adoption of the work since it can be implemented with a minimum of additional burden on the development team.

6.1 CI Tools

The first step is choosing the right CI tool. The first important feature to be considered is the ability to host the product on-premises or on the cloud. The second factor is integration and support. Finally, container-

[6]https://www.docker.com/

[7]https://kubernetes.io/

[8]https://portswigger.net/burp/enterprise

ization support is a must for the tool chosen for security testing. Containers enable developers to develop, test and deploy software more reliably by keeping the environment continuously the same. We have chosen Github Actions[9] as our tool since it meets all requirements mentioned above.

6.2 Security Tools

Github Actions combines features of a pipeline with Github source control and repository management capabilities. Individual software developers and companies active in cyber security can release tools based on the Github Actions platform. It brings new capabilities for security players in the market to integrate their tools with the platform.

6.3 Scanning & Dependency Checking

Codebases are always prone to include vulnerabilities that are easily detectable by machines. Although there could be possible logical vulnerabilities in applications, it is possible to prevent others by automated pipelines. There have been several comparisons between SCA tools (Mantere et al., 2009; Kaur and Nayyar, 2020). We have tried several tools like SonarQube, Snyk, and Fortify; here there is not a clear winner since each tool is presenting better features in different sectors. E.g., SonarQube provides extensive capabilities in detecting code smells and providing code review while Snyk is better at dependency scanning and security measures.

6.4 Open-port Scanning

We have used Github Actions to provide monitoring capabilities. A script will run the NMap[10] application at different time intervals during the day. NMap is a free open source software that is widely used by network administrators to perform network discovery and auditing. We have used NMap software in our automated pipeline to watch over the company's deployed apps and report which ports are currently open. Later on, the company can use this data to detect misconfigurations or internal mistakes which lead to open exposed ports in their network.

6.5 SSL/TLS Evaluation

We have been working on the pipeline to provide security checks to check for SSL/TLS version and configuration. This test will be part of the pipeline and can be run automatically in time intervals.

6.6 Integration and Automation

Github Actions use virtual machine runner instances to run pipelines. Each pipeline occupies portions of the runner's computational power and bandwidth usage. It is important that our automated pipeline is efficient, only running tests and builds whenever needed. It is possible to run tests on special conditions and Github runners will initiate the action whenever the trigger conditions are met. If all the pipeline tests run on every push to the repositories, this will exhaust the runners, and jobs will be in long queues before they could be completed. In addition, specific tests like SSL/TLS scans are time-consuming and will keep the runners busy for longer periods. We tried to develop smart pipelines that try to expand the runner's idle time by assigning pipeline triggers to persons committing the code. As an example, user experience and UI design teams' commits are ignored by runners. Developers' commits will trigger static analysis, code coverage, and code smell pipelines. SSL/TLS scans and port scans work in parallel in time intervals without attachment to commits. Finally, commits to master will run all possible runners to check before the new code is added to the system (Github, 2021).

7 CONCLUSION

In this paper we have studied challenges and identified solutions for integrating security in a DevOps Continuous Integration pipeline. We have constructed a DevSecOps pipeline which has been deployed in a software development organization, and preliminary feedback is positive. For further work, it would be interesting to assess the impact of the new pipeline in a longitudinal case study.

ACKNOWLEDGEMENTS

This paper is based on the two first authors' MSc thesis at the University of Stavanger.

REFERENCES

Assal, H. and Chiasson, S. (2018). Security in the software development lifecycle. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 281–296, Baltimore, MD. USENIX Association.

[9]https://docs.github.com/en/actions

[10]https://nmap.org/book/man.html

Balaji, S. and Murugaiyan, M. S. (2012). Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*, 2(1):26–30.

Bennetts, S. (2013). Owasp zed attack proxy. *AppSec USA*.

Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32.

Chung, A. (2018). How DevOps can use quality gates for security checks.

Ebert, C., Gallardo, G., Hernantes, J., and Serrano, N. (2016). DevOps. *IEEE Softw.*, 33(3):94–100.

Github (2021). Events that trigger workflows.

Goteti, H. (2015). API Driven development , bridging the gap between providers and consumers.

Hegde, V. and Singh, A. (2020). Team collaboration in DevOps: Accenture.

Howard, M. and Lipner, S. (2006). *The Security Development Lifecycle*. Microsoft Press.

Huang, C.-C. and Kusiak, A. (1996). Overview of Kanban systems. *International Journal of Computer Integrated Manufacturing*, 9(3):169–189.

Jaatun, M. G., Cruzes, D. S., and Luna, J. (2017). Devops for better software security in the cloud. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, pages 69:1–69:6, New York, NY, USA. ACM.

Jabbari, R., bin Ali, N., Petersen, K., and Tanveer, B. (2016). What is DevOps? a systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*, pages 1–11.

Kaur, A. and Nayyar, R. (2020). A comparative study of static code analysis tools for vulnerability detection in C/C++ and JAVA source code. *Procedia Computer Science*, 171:2023–2029. Third International Conference on Computing and Network Communications (CoCoNet'19).

Kronjee, J., Hommersom, A., and Vranken, H. (2018). Discovering software vulnerabilities using data-flow analysis and machine learning. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, New York, NY, USA. Association for Computing Machinery.

Kumar, R. and Goyal, R. (2020). Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (adoc). *Computers & Security*, 97:101967.

Kurniawan, A., Abbas, B. S., Trisetyarso, A., and Isa, S. M. (2018). Static taint analysis traversal with object oriented component for web file injection vulnerability pattern detection. *Procedia Computer Science*, 135:596–605.

Leite, L., Rocha, C., Kon, F., Milojicic, D., and Meirelles, P. (2019). A survey of DevOps concepts and challenges. *ACM Computing Surveys*, 52(6).

Mantere, M., Uusitalo, I., and Roning, J. (2009). Comparison of Static Code Analysis tools. In *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, pages 15–22.

Mao, R., Zhang, H., Dai, Q., Huang, H., Rong, G., Shen, H., Chen, L., and Lu, K. (2020). Preliminary findings about DevSecOps from Grey Literature. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pages 450–457.

McGraw, G. (2006). *Software Security: Building Security In*. Addison-Wesley.

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

Oyetoyan, T. D., Jaatun, M. G., and Cruzes, D. S. (2017). A lightweight measurement of software security skills, usage and training needs in agile teams. *International Journal of Secure Software Engineering*, 8(1):1–27.

Rajapakse, R., Zahedi, M., Ali Babar, M., and Shen, H. (2021). Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology*.

Rao, K. N., Naidu, G. K., and Chakka, P. (2011). A study of the Agile software development methods, applicability and implications in industry. *International Journal of Software Engineering and its applications*, 5(2):35–45.

Rehkopf, M. (2001). Manifesto for agile software development.

Rehkopf, M. (2020). Kanban vs. Scrum: which Agile are you?

Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., and Stumm, M. (2016). Continuous deployment at Facebook and OANDA. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 21–30.

Schlossnagle, T. (2018). Monitoring in a DevOps world. *Commun. ACM*, 61(3):58–61.

Shahin, M., Ali Babar, M., and Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943.

Sönmez, F. O. and Kiliç, B. G. (2021). Holistic web application security visualization for multi-project and multi-phase dynamic application security test results. *IEEE Access*, 9:25858–25884.

Talukder, A., Maurya, V., Santhosh, B., Jangam, E., Muni, S., Kp, J., Samanta, S., and Pais, A. (2009). Security-aware software development life cycle (SaSDLC) - processes and tools. In *2009 IFIP International Conference on Wireless and Optical Communications Networks*, pages 1 – 5.

Virmani, M. (2015). Understanding DevOps & bridging the gap from continuous integration to continuous delivery. In *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, pages 78–82, Piscataway, NJ. IEEE.

Williams, L., McGraw, G., and Migues, S. (2018). Engineering security vulnerability prevention, detection, and response. *IEEE Software*, 35(5):76–80.