Implementation of DevSecOps by Integrating Static and Dynamic Security Testing in CI/CD Pipelines

Agung Maulana Putra
Department of Cybersecurity Engineering
Politeknik Siber dan Sandi Negara
Bogor, Indonesia
agung.maulana@student.poltekssn.ac.id

Herman Kabetta
Department of Cryptographic Engineering
Politeknik Siber dan Sandi Negara
Bogor, Indonesia
herman.kabetta@poltekssn.ac.id

2022 IEEE International Conference of Computer Science and Information Technology (ICOSNIKOM) | 979-8-3503-9907-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICOSNIKOM56551.2022.10034883

***Abstract*— Problems at the build, test, and deploy stages are standard in the development lifecycle of systems with Agile. They are time-consuming and cause releases to fall behind schedule. DevSecOps provide the solution to this problem. This study delivers a build, test, and deployment automation solution for those working in an Agile SDLC environment. This study is an approach to implementing DevSecOps on an information system's Agile SDLC, a web-based software developed with the Node.js and Dart programming languages, Express.js, and Flutter frameworks. The process carried out in this study uses GitLab and Docker tools, consisting of five stages: continuous development, continuous testing, continuous integration, continuous deployment, and continuous monitoring. This approach shortens the time and streamlines the build, testing, and deployment, whereas previously, the process of system development was done manually. It took up to several hours to only take 3-4 minutes after automation was applied to the deployment process. In addition, we conduct a combination of automated static and dynamic security testing to help ensure the system's security by obtaining results related to vulnerabilities.**

Keywords—Build, Deploy, Docker, Test, DevSecOps, DevOps

I. INTRODUCTION

Most information systems use the Agile development life cycle in their development and maintenance. Agile is a relatively modern method because it emphasizes improvisation and adaptation. Even so, the Agile method workflow still applies the traditional systematic pattern. In Agile work environments, there are frequent issues with the build and deploy phases, which are time-consuming and lead to unscheduled release times.

DevOps (Development and Operations) is a new method that takes a collaborative and integrative approach between the development team (Dev) and the software operations team (Ops) in the process of developing and delivering software to infrastructure [1]. The DevOps method is proven to be able to reduce some of the development stages that exist in the old method. DevOps is able to shorten the time between software development and operation without compromising the quality of the software itself. DevOps is capable of delivering early-release software with a high frequency. The build, test, and deployment processes can detect problems earlier, so the code can be rolled back if there is an error during an update. The auto-deploy process can make earlier and more effective releases for software requiring a high update frequency. Figure 1 shows Agile's coding and testing process and how it compares with DevOps.

Technological advances such as Continuous Engineering, specifically DevOps, allow some organizations to gain a competitive advantage. However, security concerns have increased due to security breaches, such as large-scale data breaches, which have forced organizations worldwide to pay more attention to security threats [2]. Therefore there is a need for security integration as a need for security in DevOps, namely DevSecOps. DevSecOps aims to integrate security controls and processes into the DevOps software development life cycle with collaboration among the security, development, and operations teams.

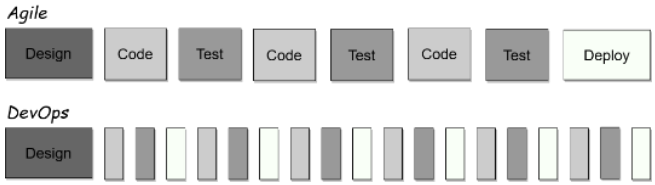

Fig. 1. Differences in Design, Coding, Testing, and Deploy in Agile and DevOps
Source: reprocessed from [1]

In this study, auto deploys to the server will be carried out to speed up and streamline the release process using GitLab CI/CD. It performs automated static and dynamic security automation testing before auto-deploy the system through the git commit mechanism. So the DevSecOps method is expected to be a solution for developing an information system.

II. RELATED WORKS

Shajadi, Abdollah [16] conducted the research on Automating Security Tests For Web Applications In Continuous Integration And Deployment Environments. In his study, the tools and process mechanisms were developed to implement automated security tests for web applications. The technology used is the Burp Suite Pro tool, the Python programming language, and the GitLab CI/CD tool. The results show that a Python script named Skinner performs automated security testing with Burp Suite Pro on the GitLab CI pipeline using the DevSecOps implementation procedure.

The second study was conducted by Tohirin et al. [1]. In his study, DevOps was implemented on the SDLC Agile Scrum web-based Covid-19 e-Screening application developed with the PHP programming language and the Laravel framework. This research results show that DevOps can be implemented well in the development process of e-Screening Covid-19 applications. Its code merging occurs quickly, daily builds are smooth, and code health and feasibility checks occur every time the developer commits and pushes. Tohirin et al. [1] use the DevSecOps method to the Dart and Node.js programming languages with the Flutter and Express.js frameworks, which are applied to an information system based on mobile and web applications.

The third research is conducted by Shama, Abriza Mahandis, and Dian W. Chandra [17]. In this research, a DevSecOps system was created for automation to speed up developer work and improve code quality. The result of this

research is the developer's work process at PT. Emporia Digital Raya is faster because it is assisted by the CI/CD process and improves the quality of the program code with the SAST process. The result of this research is the developer's work process at PT. Emporia Digital Raya is faster because it is assisted by the CI/CD process and improves the quality of the program code with the SAST process. In this study, the GitLab CI/CD tool is used as a system for auto deploy to the server to speed up and streamline the release process for an information system development.

The fourth research is conducted by Rangnau et al. [24]. In his study, a case study applies three different testing techniques in CI/CD. This will enable us to identify pitfalls, challenges, and shortcomings DevOps teams may encounter while automating security tests. Three dynamic application security testing techniques are integrated into a CI/CD pipeline. There are WebApplication Security Scanning (WAST) using Zed Attack Proxy (ZAP), Security API Scanning (SAS) with JMeter, and Behaviour Driven Security Testing (BDST) using SeleniumBase automation framework. The result of this research is to enable informed decisions when implementing DevSecOps practices in agile enterprise applications engineering processes and enterprise security.

TABLE I. RELATED WORKS COMPARISON

Factor	Related Works				This Study
	[1]	[16	[17]	[24]	
DAST		✓		✓	✓
SAST			✓	✓	✓
Open Source Testing Tools				✓	✓
Adopt in Agile Software Development	✓			✓	✓
Continuous Deployment					✓

III. RESEARCH METHODOLOGY

This research method performs several stages in planning, analysis, implementation, and testing, as seen in figure 2.

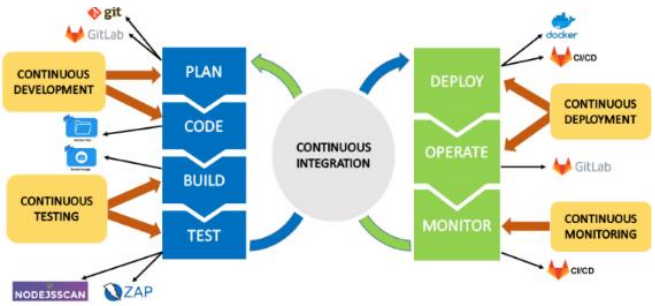

Fig. 2. Research Design
Source: reprocessed from [1]

- *Continuous Development*
  The plan is the stage where the planning process is carried out in a job [1]. At this stage, we use GitLab tools because CI/CD tools can do CI/CD storage and code management in the same place. Code is the stage where developers create, manage, and share code in the repository [1]. To do coding in software using git to maintain code (version control) to make it easier, so that system control can be done.
- *Continuous Testing*
  In this study, the build phase was carried out to automate and standardize the process for building and implementing code on information systems. The test is the stage of conducting unit testing of application features to check for vulnerabilities or check public endpoints to ensure they are accessible [16]. In the aspect of security testing, we perform static and dynamic tests automatically after the code is committed. Static testing is performed by automating the NJSSCAN tools, whereas dynamic testing is performed by automating the OWASP ZAP tools. This research focuses on the implementation of this stage.
- *Continuous Integration*
  In this study, GitLab is used to detect commits made by developers as early as possible and perform code integration which involves compiling and reviewing code. GitLab is connected to the server so the deployment process can be carried out.
- *Continuous Deployment*
  Deployment automation accelerates software deployment, ensures more consistency in the development process and deploys secure configurations for all systems and services. The Docker and GitLab CI/CD tools were used [17]. This stage is rejected when the automated security test in the continuous testing result failed.
- *Continuous Monitoring*
  In this study, the monitoring stage is carried out on software performance after the application is deployed on the server to identify network or server problems and root causes and maintain security [17].

IV. RESULT AND DISCUSSION

At this stage, the DevSecOps implementation is carried out to auto-deploy the system using GitLab CI/CD starting from the development stage to the deployment and monitoring process.

A. Continuous Development

The first step at this stage is to create a GitLab project and add the application files. Furthermore, cloning the repository locally to be able to run the application locally and perform testing on the application. Then a remote repository of GitLab is carried out to a local repository which aims to integrate the development process.

```
01-17-AGUNGMAUL+AgungMP@01-17-agungmaulana MINGW64 ~/Documents
$ git clone https://gitlab.com/agungmp1/TAkorpstar.git
Cloning into 'TAkorpstar'...
remote: Enumerating objects: 6358, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 6358 (delta 13), reused 0 (delta 0), pack-reused 6338
Receiving objects: 100% (6358/6358), 11.93 MiB | 3.89 MiB/s, done.
Resolving deltas: 100% (1272/1272), done.
Updating files: 100% (6279/6279), done.

01-17-AGUNGMAUL+AgungMP@01-17-agungmaulana MINGW64 ~/Documents
$ cd TAkorpstar

01-17-AGUNGMAUL+AgungMP@01-17-agungmaulana MINGW64 ~/Documents/TAkorpstar
$ git remote
origin
```

Fig. 3. GitLab cloning and remote process

Then the data in the local repository is committed and pushed to GitLab using git bash. All data in the local repository is automatically executed to the GitLab repository.

```
01-17-AGUNGMAUL+AgungMP@01-17-agungmaulana MINGW64 ~/Desktop/TAkorpstar
$ git add .

01-17-AGUNGMAUL+AgungMP@01-17-agungmaulana MINGW64 ~/Desktop/TAkorpstar
$ git commit -m "test commit"
[main 1f0008d] test commit
 1 file changed, 1 insertion(+), 1 deletion(-)

01-17-AGUNGMAUL+AgungMP@01-17-agungmaulana MINGW64 ~/Desktop/TAkorpstar
$ git push -u origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 292 bytes | 29.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
To https://gitlab.com/agungmp1/TAkorpstar.git
   e69e7ad..1f0008d  main -> main
branch 'main' set up to track 'origin/main'.
```

Fig. 4. Execute git commit and git push

To have a repository on the Docker Hub, you can sign up or register at the Docker Hub and create a Docker account in the free version. The image registry address in Docker Hub is used to push images built by the pipeline into that repository. GitLab requires the username and password credentials of the user's Docker Hub account to enter the registry before the push image is performed. The username and password credentials of the Docker Hub account are not included in the pipeline code because the pipeline is part of a publicly accessible repository. Therefore a project variable is created in the CI/CD pipeline configured in the GitLab project settings. Of course, it is more secure to protect confidential data or sensitive data.

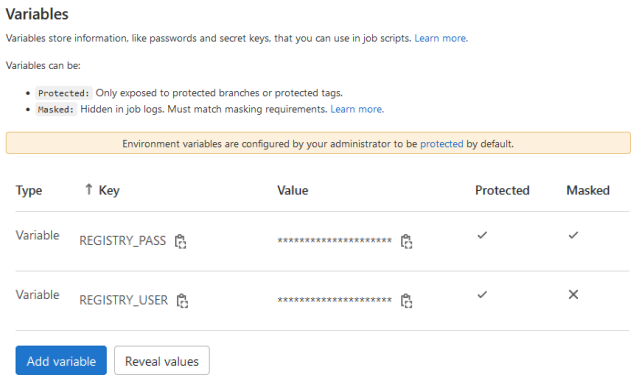

Fig. 5. Project variables that have been created

The code management process is carried out at the Code stage using the CI/CD pipeline on GitLab. All CI/CD configurations are written in YAML file format with the name ".gitlab-ci.yml". Tasks in the CI/CD pipeline are configured as jobs, such as running testing, building images, and deploying to the server. To build a Docker image from a Node.js application, a Dockerfile is used, which defines the base image of the application server and as a place to install all the dependencies required by the application.

A Dockerfile is a text document that contains a command to build an image. Using the Docker build command, the user can execute multiple command line instructions sequentially. The FROM instruction initializes a new build stage and sets the base image for the next instruction. Figure 6 shows the contents of the dockerfile configuration used

```
FROM ubuntu:20.04

RUN rm /bin/sh && ln -s /bin/bash /bin/sh

# update the repository sources list
# and install dependencies
RUN apt-get update \
    && apt-get install -y curl \
    && apt-get -y autoclean

# nvm environment variables
ENV NVM_DIR /root/.nvm
ENV NODE_VERSION 14.19.1

# install nvm
# https://github.com/creationix/nvm#install-script
RUN curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/

# install node and npm
RUN source $NVM_DIR/nvm.sh \
    && nvm install $NODE_VERSION \
    && nvm alias default $NODE_VERSION \
    && nvm use default
```

Fig. 6. Dockerfile Configuration

B. *Continuous Testing*

At the Build stage, the release pipeline builds for the Node.js application by executing application tests, building a docker image, and deploying it to the server. Here is the configuration of the image build process:

```
before_script:
  - docker login -u $REGISTRY_USER -p
$REGISTRY_PASS
Build:
  stage: build
  script:
    - docker build -t
$CONTAINER_RELEASE_IMAGE .
    - docker tag $CONTAINER_RELEASE_IMAGE
$CONTAINER_RELEASE_IMAGE
    - docker push $CONTAINER_RELEASE_IMAGE
  tags:
    - agung
```

Values in the pipeline, such as image tag names, can be extracted into custom variables. Using variables, among others, can store values you want to reuse and reduce code duplication [20].

```
variables:
  CONTAINER_RELEASE_IMAGE:
                    agungmp30/system:v1
```

Furthermore, security testing is carried out at this stage, including static and dynamic tests. Testing is done using Njsscan and OWASP-ZAP tools.

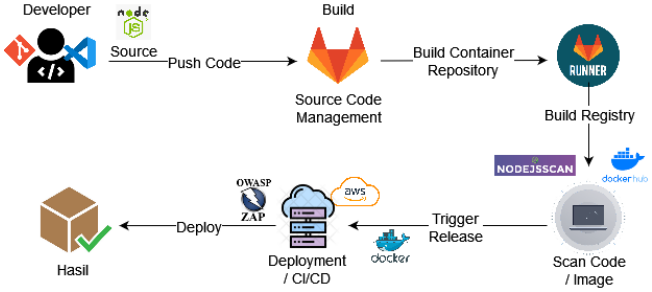

Fig. 7. Security testing integration scheme
Source: personal processing

The following is a static testing configuration using Njsscan.

```
SAST:
  stage: test1
  image: python
  script:
    - njsscan . --html -o report.html ||
        exit 0
  allow_failure: true
  artifacts:
    when: always
    untracked: false
    expire_in: 30 days
    paths:
      - report.html
  tags:
  - agung
```

The following is a dynamic testing configuration using OWASP-ZAP.

```
DAST:
  stage: test2
  image: owasp/zap2docker-stable:latest
  script:
    - docker run -i owasp/zap2docker-
      stable zap-cli quick-scan --self-
      contained --start-options '-config
      api.disablekey=true'
      http://$SERVER_ADDRESS:8080 >
      zapreport.html
  allow_failure: true
  artifacts:
    when: always
    untracked: false
    expire_in: 30 days
    paths:
       - zapreport.html
  tags:
  - agung
```

In running the testing process, the job pipeline also automatically generates artifacts. The artifact contains information related to the application code scanning test results regarding the vulnerabilities contained in the application.

TABLE II. RESULT OF SECURITY TESTING ARTIFACT

RULE ID	node_insecure_random_generator	
CWE	CWE-327: Use of a Broken or Risly Cryptographic Algorithm	
OWASP-WEB	A9: Using Components with Known Vulnerabilities	
DESCRIPTION	Crypto.pseudoRandomBytes()/Math.random() is a cryptographically weak random number generator.	
SEVERITY	WARNING	
FILES	File	app/utils/telebot.js
	Match Position	264 - 227
	Line Number(s)	24
	Match String	let message = '\n${name}

It can be seen in table II that there is a security gap based on CWE-327, namely the Use of a Broken or Risky Cryptographic Algorithm, which is an unnecessary risk that can result in the disclosure of sensitive information [18]. In addition, there are also security holes based on OWASP-WEB A9 regarding Using Components with Known Vulnerabilities, where components such as libraries, frameworks, websites, and other software modules run with the same privileges as the application are vulnerable to exploitation [19].

C. Continuous Integration

Continuous integration includes a configuration process coordinated by GitLab CI/CD with the deployment server. At this stage, register the GitLab runner on the GitLab project that has been created. GitLab Runner is registered on the server for deployment. In addition, the deployment server is needed to auto-deploy and as a place to run Docker applications on the server. The server is built using the Amazon Web Service (AWS) platform with the configuration requirements, which can be seen in table III as follows.

TABLE III. CONFIGURATION ON AMAZON MACHINE IMAGE

No	Type	Information
1	Software Image	Ubuntu Server 20.04 LTS (HVM), Canonical, amd64 focal image, 64-bit (x86)
2	Virtual Server/Instance	Tipe t2.medium, 2 vCPU, 4 GB Memory
3	Storage	SSD Volume 8 GB
4	Private Key	RSA type, .pem file format

The key pair allows connecting to the instance securely. When prompted, store the private key in a secure, accessible location on the local machine to connect to the instance. RSA encrypted private and public keys. The private key .pem file format is used with OpenSSH. The server is accessed remotely from the local machine using the SSH (Secure Shell) command by applying the SSH key added by the security settings on the server. SSH is used to access remote servers over the internet securely.

D. Continuous Deployment

The continuous deployment phase includes the deployment process, which is carried out with push images to servers operated via the GitLab CI/CD pipeline and Docker containers. The server is connected to the local machine by using the SSH command and the server IP address, which is the public IP address, and adding username credentials as authentication to connect to the server. After initialization, the configuration is done by installing Docker on the server to be able to run the Docker container in the process of deploying the Docker image to the server.

In the pipeline configuration, a new stage called "deploy" is added by creating a new job. The steps taken are using the SSH command locally, and GitLab Runner was starting the container for the deploy job. The "docker run" command is executed to run Docker on the deployment server. The application runs on port 8080, which is exposed on the host. Before executing the "docker run" command, stop and delete existing containers or stop containers running on port 8080 so that new containers can be created on each subsequent execution run.

```
- docker pull $CONTAINER_RELEASE_IMAGE
- docker stop $(docker ps -a -q) || true &&
  docker rm -f $(docker ps -a -q) || true
- docker run -d -p 8080:8080
  $CONTAINER_RELEASE_IMAGE
```

Deployment validation can be performed via the GitLab CI/CD pipeline, inspecting Docker containers running through the server and accessing the application via a web browser. Figure 8 shows the job status completed on CI/CD pipeline.

Fig. 8. Job status-completed on pipeline

E. *Continuous Monitoring*

In the continuous monitoring stage, the monitoring process is carried out on the GitLab CI/CD pipeline as a place to run deployment automation commands and on the server, which includes collection, reporting, CPU usage storage, memory usage, RAM allocation, and pipeline modifications used in system development. Server services must be accessible and run appropriately without experiencing interference, so server monitoring is needed so that the server's performance can run normally.

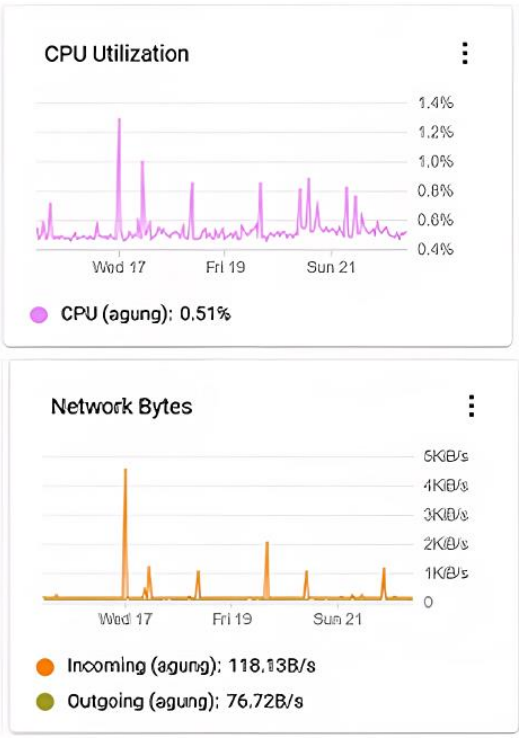

Fig. 9. Monitoring Dashboard

Monitoring activities help developers identify problems that can arise from system software. In addition, it can provide feedback that can be used for decision-making and improvement of the developed device by showing vulnerabilities and serving as material for the subsequent evaluation of the system development team.

F. *Analysis*

Based on the implementation that has been carried out, the results of the implementation of automation throughout the deployment chain can accelerate the work process for developers of an information system assisted by the CI/CD process and improve the quality of system security because there is a security testing process. The time required for this process is 3 minutes 18 seconds which can be seen in figure 16.

Fig. 10. Job time status on pipeline

Therefore, compared with the development process on an information system that still applies the Agile system development life cycle, the results of the implementation of DevSecOps in this study can meet the needs in the application of system automation and shorten the work process in an information system. A comparison of the work process in an information system with the system applied in this study can be seen in table IV.

TABLE IV. COMPARISON OF WORK PROCESS

No	The work process in an information system	The work process applied to the research
1	The process of deploying to the server manually	Deploy process to a server automatically with CI/CD system
2	The deployment work process takes 2-3 hours	The deployment process takes 3-4 minutes
3	There is no security testing process	There is a security testing process
4	Not deploying Docker containers on the system	Implementing a Docker container on the system

Based on the information above, it can be seen that the system automation carried out in the implementation of DevSecOps is able to increase efficiency in the aspect of the work process, whereas, in an information system, it takes up to several hours because the implementation process on the system is still done manually. While in this study, the process is conducted automatically with the CI/CD system. It uses Docker by making the application an image that is run in the Docker container to simplify and speed up the build, test, and deployment process. Security testing is carried out on the automation process to ensure system security.

V. CONCLUSION

DevSecOps has been successfully implemented using GitLab and Docker tools integrated with the deployment server. This implementation consists of 5 parts: continuous development, continuous testing, continuous integration, continuous deployment, and continuous monitoring. System automation carried out in the implementation of DevSecOps is able to increase efficiency in the aspect of the work process, whereas, in an information system, it takes up to several hours because the implementation process on the system is still done manually. While in this study, the process is carried out automatically with a CI/CD system where the deployment takes only 3-4 minutes using Docker, which renders the application as an image and runs in a Docker container to simplify and speed up the build, test, and deploy process.

The combination of automated static and dynamic security testing performed in a DevSecOps implementation can help ensure system security. Real-time test results can quickly notify developers about security holes or vulnerabilities in the program so that repairs can be made as soon as possible to improve system security.

REFERENCES

[1] T. Tohirin, S. F. Utami, S. R. Widianto, and W. A. Mauludyansah, "Implementasi DevOps Pada Pengembangan Aplikasi e-Skrining Covid-19," *MULTINETICS*, vol. 6, no. 1, pp. 15–20, May 2020.

[2] V. Gupta, P. K. Kapur, and D. Kumar, "Modeling and measuring attributes influencing DevOps implementation in an enterprise using structural equation modeling," *Inf. Softw. Technol.*, vol. 92, pp. 75–91, Dec. 2017.

[3] texnokot, "DevSecOps controls - Cloud Adoption Framework." https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/secure/devsecops-controls (accessed Jan. 11, 2022).

[4] B. A. Skurla, “DevOps Integration of Security Practices,” p. 55, 2020.

[5] A. Agarwal, S. Gupta, and T. Choudhury, “Continuous and Integrated Software Development using DevOps,” in *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, Paris, Jun. 2018, pp. 290–293.

[6] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps,” *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, May 2016.

[7] S. Sharma, “DevOps For Dummies,” *the knowledge*, p. 76, 2015.

[8] J. Van Baarsen, *GitLab Cookbook: over 60 hands-on recipes to efficiently self-host your own Git repository using GitLab*. Birmingham Mumbai: Packt Publishing, 2014.

[9] S. Chacon and B. Straub, “Pro Git Second Edition,” Apress.

[10] Anonymous, “How to convince leadership to adopt CI CD,” *GitLab.* https://page.gitlab.com/2021_eBook_leadershipCICD.html (accessed Dec. 28, 2021).

[11] A. Mardan, "Express.js Guide," *Leanpub*, 2014.

[12] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 228, 2017.

[13] I. Miell, A. Sayers, "Docker in practice," *Simon and Schuster*, 2019.

[14] D. Sagar, S. Kukreja, J. Brahma, S. Tyagi, and P. Jain, "Studying open source vulnerability scanners for vulnerabilities in web applications," *IIOAB JOURNAL*, vol. *9,* no.2, pp. 43-49, 2018.

[15] F. Holik, and S. Neradova, "Vulnerabilities of modern web applications," *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO),* pp. 1256-1261) IEEE, May 2017.

[16] A. Shajadi, “Automating Security Tests For Web Applications In Continuous Integration And Deployment Environment,” p. 64.

[17] “DevSecOps: Incorporate Security into DevOps to Reduce Software Risk,” *Agile Connection*. https://www.agileconnection.com/article/devsecops-incorporate-security-devops-reduce-software-risk (accessed Jan. 11, 2022).

[18] Y. He, R. S. Camacho, H. Soygazi, and C. Luo, “Attacking and Defence Pathways for Intelligent Medical Diagnosis System (IMDS),” *International Journal of Medical Informatics*, vol. 148, no. 104415, 2021.

[19] H. Tupsamudre, M. Sahu, K. Vidhani, and S. Lodha, “Fixing the Fixes: Assessing the Solutions of SAST Tools for Securing Password Storage,” *International Conference on Financial Cryptography and Data Security,* pp. 192–206, Springer, Cham, 2020.

[20] N. Vendor, B. Mosolygo, and P. Hegelus, “Comparing ML-Based Predictions and Static Analyzer Tools for Vulnerability Detection,” *International Conference on Computational Science and Its Applications*, pp. 92-105, Springer, Cham, 2022.

[21] D. Hariyadi, F. E. Nastiti, “Analisis Keamanan Sistem Informasi Menggunakan Sudomy dan OWASP ZAP di Universitas Duta Bangsa Surakarta,” *Jurnal Komtika (Komputasi dan Informatika),* vol. 5, no. 1, pp. 35–42, 2021.

[22] S. Chalishhafshejani and B. K. Pham, "Automated software security activities in a continuous delivery pipeline," *Faculty of Science and Technology*, Springer semester, 2021.

[23] Fathurrahman and Ester, "Automatic Scanner Tools Analysis As A Website Penetration Testing," *Jurnal Mantik*, vol. 4 no. 2, pp. 1138-1144, 2020.

[24] T. Rangnau, R. v. Buijtenen, F. Fransen and F. Turkmen, "Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines," *EDOC Conference, IEEE,* 2020.