

National  
College of  
Ireland

# Automation of Remediation of Configuration Vulnerabilities Reported by the DAST Scanning Procedure

MSc Industrial Internship  
MSc Cyber Security

Vladyslav Bril  
Student ID: x21102872

School of Computing  
National College of Ireland

Supervisor: Vikas Sahni

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Vladyslav Brill  
**Student ID:** x21102872  
**Programme:** MSc Cybersecurity **Year:** 2022  
**Module:** MSc Internship  
**Supervisor:** Vikas Sahni  
**Submission Due Date:** 6<sup>th</sup> of January  
**Project Title:** Automation of Remediation of Configuration Vulnerabilities Reported by the DAST Scanning Procedure  
**Word Count:** 6996 **Page Count:** 20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:**



**Date:** 06.01.2023

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

Attach a completed copy of this sheet to each project (including multiple copies)	<input checked="" type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission,</b> to each project (including multiple copies).	<input checked="" type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project,</b> both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input checked="" type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Automation of Remediation of Configuration Vulnerabilities Reported by the DAST Scanning Procedure

Vladyslav Bril  
x21102872

## Abstract

Modern security requirements have affected approaches to building a DevOps model, stimulating the transition to DevSecOps paradigm with the addition of elements of checking the product for compliance with security criteria. In most cases, the vulnerabilities found during product testing, described in the generated reports by dynamic testing tools (DAST), need to be fixed manually which can require a lot of effort from developers who may not deal with the aspects of secure product creation.

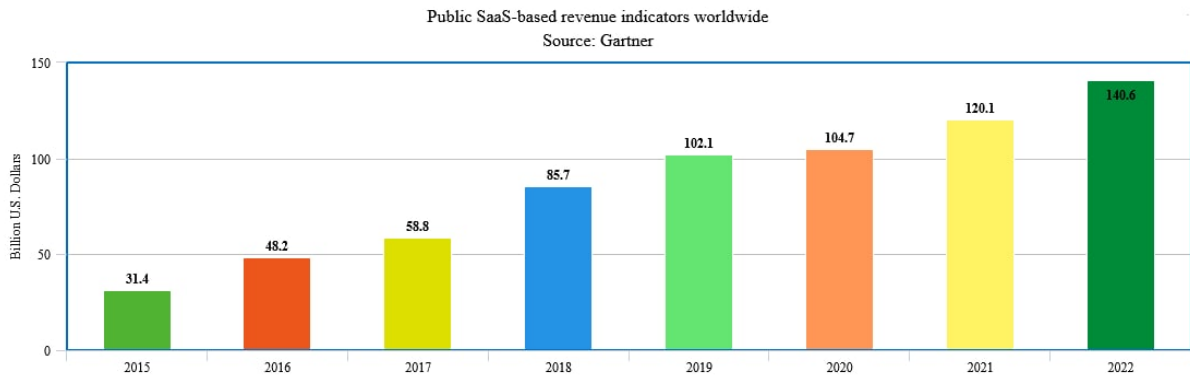
A solution to this problem is a separate module that can automate the process of fixing vulnerabilities detected, as well as having the ability to be integrated into the CI/CD pipeline. The concept of dedicating remediation procedures to the pre-defined scenarios is significant to enhance the overall product security level, as well as release the developers from the burden of regular vulnerabilities fixes. This work analysed current trends in building automated DevOps and DevSecOps factories, delivered a software component that aims to automate the remediation activities after conducting the DAST operations, and also proposed an optimal DevSecOps scheme for which it is possible to introduce such software.

## 1 Introduction

Over the past ten years, companies developing their own digital product have begun to realize that the volume and intensity of work is growing with the expansion of new technologies. The on-premises scenario that existed at that time, when the consumer received software directly to his device, began to hamper the ability of developers to release regular technical updates and optimization patches, and project managers were forced to resort to stricter prioritization of released features (Al Hayek and Abu Odeh, 2020). In order to solve this problem, with the development of cloud technologies, the Software-as-a-Service business model developed, which involved deploying a product in a cloud infrastructure with further provision of access to users through a single-entry point (e.g., web browsers) (Mell and Grance, 2011). This approach to implementation has become a sensation, and according to the Gartner research, by the end of 2022, revenue for SaaS products will reach the peak<sup>1</sup> at 140.6 billion U.S. dollars (Figure 1).

---

<sup>1</sup> <https://www.statista.com/statistics/505243/worldwide-software-as-a-service-revenue/>



**Figure 1: Public SaaS-based revenue indicators worldwide**

The transition to the as-a-Service model freed the hands of developers, who were able to regularly release updates and control that users are working with the most up-to-date version of their product (Rangnau *et al.*, 2020). An additional component of success was the popularization of the DevOps approach, as an agile method of developing and delivering applications to the end user. Its main concept is to automate the process of building, testing, deploying, and assessing the compliance of the application according to previously identified metrics (Ahmed and Francis, 2019). Over time, it was necessary to introduce elements into the classic DevOps loop that could analyze and accompany all stages of product development from a security point of view. Formerly established classical approaches could not match the pace with which the application went through the stages of the DevOps cycle. Based on this, a new approach was developed that combines elements of DevOps and Security – DevSecOps (Rangnau *et al.*, 2020; Sen, 2021). From now on, each phase of DevOps has been accompanied by a characteristic security-related phase, whether it was threat planning at the initial stage; static and dynamic testing at the assembly stage; or conducting more generic penetration testing techniques, as well as monitoring the occurrence of incidents in the system.

According to a study published by Forbes<sup>2</sup>, in 2021 the average number of attacks and attempts to compromise the system increased by 15% compared to 2020. In addition, as the authors of the article mentioned: "The main causes of these attacks will come from misconfigurations, human error, poor maintenance, and unknown assets.". The previously mentioned testing approaches are intended to minimize the occurrence of risks due to the above conditions, especially when it comes to misconfiguration or human error. Testing tools such as SAST and DAST have already been analysed in research papers quite a lot, including for their operational potential (Pinconschi, Abreu and Adao, 2021; Huang *et al.*, 2022). However, until now, according to the author of this work, the issue of automation of error correction, according to the results of testing, remains open. Often, developers are forced to manually fix the found misconfiguration vulnerabilities themselves in order to ensure the safe operation of the product. In this regard, is it possible to analyze how it is feasible to automate the process of fixing such vulnerabilities found as a result of scanning a software product for errors?

<sup>2</sup> <https://www.forbes.com/sites/chuckbrooks/2022/06/03/alarming-cyber-statistics-for-mid-year-2022-that-you-need-to-know/>

As a subject of research, approaches for conducting dynamic testing (DAST) at the pre-production stage will be further analysed. In addition, a supplementary goal of the work is to build a model of the factory based on the DevSecOps approach, into which subsequent developments could be integrated without loss in output quality.

The remainder of the work is structured as follows: Section 2 provides additional details regarding dynamic testing approaches, and also demonstrates a literature analysis of existing solutions regarding building DevSecOps models. Section 3 contains a description of the author's approach to the study of the task, and also creates the basis for further experiment described in Sections 4 and 5. In addition, Section 5 contains the results obtained during the development of an auxiliary software product. Section 6 conducts an analytical comparison of the main characteristics of the development, according to the previously stated criteria. The last Section 7 summarizes all the work done, and also indicates possible next steps to develop the affected problem in order to find a more optimal and optimized solution.

## 2 Related Work

The following section contains the analysis of the current state-of-art of the research related topics, such as different approaches to security testing that are being incorporated into the DevSecOps pipeline. In addition, DevOps similar to DevSecOps does not dictate any particular strict realization and the whole CI/CD factory building process is in the hands of system engineers. With that being said, an exhaustive analysis of the most common and widely used DevSecOps models should be conducted in order to determine whether it is possible to integrate the after-testing issue resolving components without breaking the core principle of continuous integration and delivery.

### 2.1 Security Testing Approach in DevSecOps

As mentioned earlier, the infosystem is evolving from a single, straightforward, independent structure to a complete approach with a complicated architecture based on microservices, and the scale of software is growing, making the challenge of proper quality assurance thought-provoking (Sun *et al.*, 2021). On the other hand, such cloud-based technologies as IaaS, PaaS, and SaaS are rapidly advancing, changing the way traditional system architectural design is thought of while also introducing new security concerns (Chen and Suo, 2022). Since the beginning of the development cycle, a DevSecOps model has implemented network security practices. To find security flaws, the code is examined, audited, scanned, and tested throughout the development cycle (Simonjan, Taurer and Dieber, 2020).

According to [Zech \(2011\)](#), whenever an software product meant to be launched, an exhaustive testing should be conducted in order to cover both functional and non-functional requirements that come from formerly composed testing scenarios. However, when the application is delivered under Software-as-a-Service (or any other as-a-Service type), security considerations should play a major role in product delivery chain.

Observing the core concepts of DevSecOps model, security testing (in terms of code & application as a whole) is mainly conducted on stages when developers try to integrate their solutions into the existing application. Due to the fact that vulnerabilities can arise at any stage of the traditional DevOps model, different number of security testing took place in the

DevSecOps model, including SAST (static application software testing) and DAST (dynamic application software testing). Both these approaches have also adapted to the transition into more Agile SDLC models. For example, Faber (2020) justifies how different testing approaches combined in the single paradigm, including their scope and responsibilities, may enhance the overall security of the product released through the latter DevSecOps model.

In another paper, Rahman and Williams (2016) pinpoint to the fact that under rapid delivery model that DevOps offers to the organization to deliver their service to the customers, a big security related issues may arise, since there is a possibility of ignoring such problems in favour of releasing the product on time and meeting the clients demands. Speaking more on the aforementioned topic, Mohan and Othmane (2016) makes a statement that the lack of adequate security testing and absence of security team in the software delivery chain would affect the overall security factors of the product delivered. However, such activities as automated code review, static and dynamic testing, as well as automated monitoring may enhance the overall level of application's security. Moreover, such an attitude should not interfere the Agile model utilized in the company.

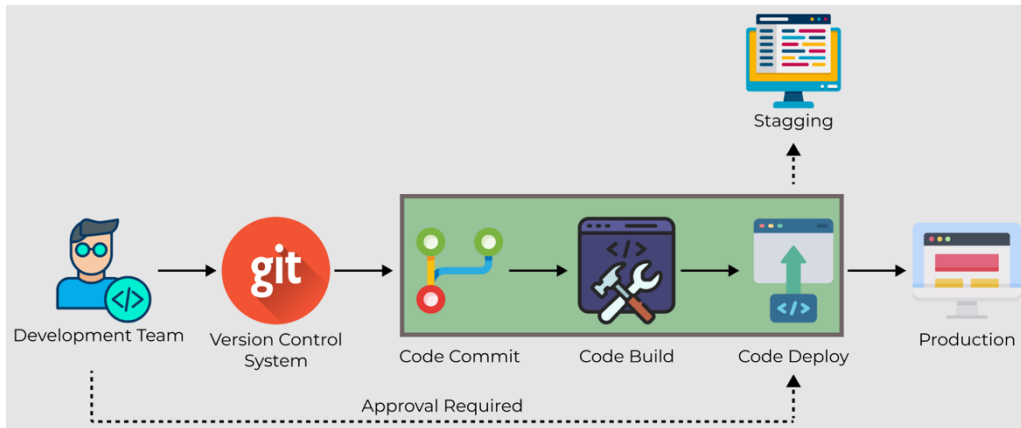
Chen et al. (2022) conducted research analysing the majority of the stages that occur in the SDLC and attempted to implement security related scenarios in each of them to enhance the overall security of the delivered product. The group of authors showed that such phase as “Coding and Unit Testing” can incorporate different security testing techniques, namely software composition analysis (SCA) and static application security testing (SAST). Similar to the latter stage, the “Testing” phase in the majority of cases involves types of security testing that can be conducted on stage environments, when there hasn't been any public release done so far, but the application is already running for being checked by the Quality Assurance team. Such tests involve dynamic application security testing (DAST), or a hands-on penetration testing conducted by a third-party or an authorized insider.

There are also a certain number of papers (Arnold and Qu, 2020) that analysed the approach to conduct white-box testing (static testing) and black-box testing (dynamic testing) in a traditional and DevSecOps models. According to the previously mentioned authors, even though such testing as a black box is preferred to be conducted in a manual way, the certain requirements that a DevOps model demands from the system engineers make it necessary to transfer the manual testing workflow into the automated alternative.

## **2.2 Existing DevSecOps Model**

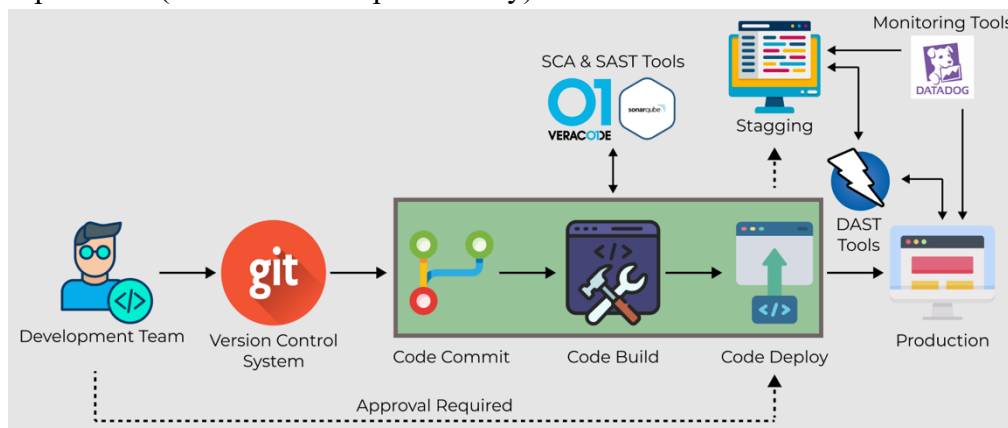
Approach to Agile development maintains the effort on replying to user requests and comprises self-organization and adaptation to specific demands.

A more recent strategy called DevOps tries to integrate both aspects of software development and operations. DevOps' primary characteristic is the automation of numerous software testing and integration processes, which enables businesses to build and deliver new software releases quickly and smoothly. Although the software development industry has benefited from both Agile and DevOps approaches, security is sometimes overlooked in either strategy (Lee, 2018). Figure 2 indicates a simplified version of the build & release phases of the aforementioned model.



**Figure 2: Simplified schematic version of the DevOps model (build & release phases)**

Based on the previous statement, DevSecOps tries to integrate security related aspects into each standard DevOps phase with the aim to enhance the overall safety of the product delivered. DevSecOps offers the ability to boost the velocity of the process of integrating alterations into production while maintaining high standards for both quality and security. Collaborations across various IT work groups or disciplines (development, security, and operations/implementation) must be given more attention if DevSecOps is to be effectively incorporated into the existing work cycle (Lee, 2018; Ibrahim, Yousef and Medhat, 2022). Figure 3 describes a simplified version of the previously showed DevOps model with addition of extra security related components that enables the transition from DevOps to DevSecOps model (build & release phases only).



**Figure 3: Simplified schematic version of the DevSecOps model (build & release phases)**

In general, the demand for quick deployment of safe and secure software outputs is driving up interest in DevSecOps in both business and academics (Rajapakse *et al.*, 2022). The expanding corpus of formal literature in this field demonstrates that a big number of literature sources emphasizes the necessity of adopting the DevSecOps concept into the SDLC model (Anjaria and Kulkarni, 2021; Almeida, Simões and Lopes, 2022). Furthermore, such big domains as cloud migration and integrating IoT devices have already been brought to the light in terms of existing academic resources (Carturan and Goya, 2019; Sojan, Rajan and Kuvaja, 2021). However, as it was stated in Section 2.1, there are no separate works related to the adaptation of the existing model that would facilitate the automation of the

security operations inside the DevSecOps model, for instance DAST scanning results remediation in particular.

### 2.3 Literature Analysis Results

Table 1 down below indicates the analysis of some of the different literature sources that were mentioned earlier in the previous sections. The main aim of such investigation is to determine the drawbacks or flaws of different kind in the examined papers and how the further proposed solution could facilitate the overall process of releasing a secure software and build a reliable DevSecOps model that includes new aspects in it.

**Table 1: Literature analysis results per source**

<b>Author</b>	<b>Title</b>	<b>Approach Discussed</b>	<b>Improvements Required</b>
Ahmed, Z. and Francis, Shoba. C. (2019)	<i>Integrating Security with DevSecOps: Techniques and Challenges</i>	The paper discusses the practical approach towards implementing the DevOps and DevSecOps models into the development lifecycle. The result demonstrates that by implementing different security aspects from the earliest stage of software development, it is possible to increase the total level of security of the application on each phase of development.	Even though the testing procedure was discussed in the course of the work, a more detailed approach is required to verify the necessity of implementing the SAST/DAST tools into the DevSecOps model, as well as how these tools could be automated to maintain the velocity of the product being delivered. Furthermore, additional attention should be pointed out to the possibility of vulnerability remediation by utilizing an automated approach.
Chen, T. and Suo, H. (2022)	<i>Design and Practice of Security Architecture via DevSecOps Technology</i>	The authors describe the advantages and key points on DevSecOps model and what benefits can it deliver to both the developing team and the product released. As a result, it was mentioned that the DevSecOps has the highest percentage of issue findings in a production environment comparing to other models and approaches.	While discussing the testing phase of the DevSecOps model, it would be beneficial to demonstrate how different testing approaches (e.g., SAST or DAST) can alter the overall workflow of the development paradigm, as well as how the idea of automating everything in the work phase can be transferred to testing & after-testing remediation procedures.



Sun, X. <i>et al.</i> (2021)	<i>Design and Implementation of Security Test Pipeline based on DevSecOps</i>	The paper demonstrates the realisation of a security testing pipeline that aims on source code testing for verifying the overall reliability of the software released.	While the work mainly discusses the approach to a source code testing in a dedicated pipeline, extra research is required to identify how to implement a dynamic or a runtime application testing into the developed model that would facilitate the overall quality of the software being released.
Simonjan, J., Taurer, S. and Dieber, B. (2020)	<i>A Generalized Threat Model for Visual Sensor Networks</i>	The paper shows a strategical threat model for the visual sensor networks (VSN) while examining some of the most impactful threats aimed onto it. It was mentioned how to deduct and proceed with the threats once they have been found.	As the outcome of the work mentions, a DevSecOps approach should be considered for further research, since its capabilities are matching the expectations for continuous security implementation in all phases of development, including the testing phase.
Faber (2020)	<i>Testing in DevOps</i>	Different testing techniques and approaches that can be used under the DevOps paradigm to maintain the secure release of the software components; the main motive says that DevOps should implement as much automation as possible.	Even though the role of testing automation is discussed throughout the work, there is no idea delivered regarding the transition to the DevSecOps model (at least partial), plus the automation is not considered for the test findings & remediation procedures.

As a conclusion on the Subsection 2.1, it can be said that all the previously mentioned works in Table 1 present different approaches to the aspects of testing that need to be implemented in the DevSecOps model in order to achieve high product security criteria. However, none of the aforementioned articles, as well as other publications that were analyzed but not included in this section, did not contain a study whose purpose would be to provide a solution or a practical realization with which developers could not only analyze the results of scans and security tests, but still capable of automating the process of fixing detected problems during the testing phase (in particular, during DAST, provided that the application has already been compiled).

Similar to Subsection 2.1, while conducting the DevSecOps model analysis, it was clear that there are no separate works related to the adaptation of the existing model that would

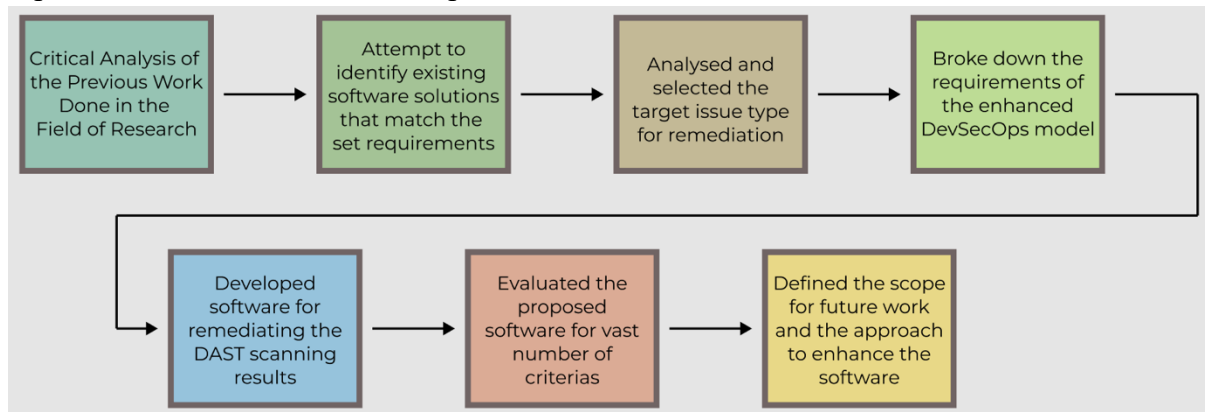
facilitate the automation of the security operations inside the DevSecOps model, for instance DAST scanning results remediation in particular.

### 3 Research Methodology

Based on the preceding literature review, it could be expected that a solution that could automate methods for repairing problems after dynamic testing is required. This solution should also be able to be included into the DevSecOps paradigm without going against its tenets. Similar to the previous section, the research methodology is divided into two sections. The first part describes the DAST tool and how its results will be automatically extracted, examined, and adjusted. The second section evaluates the DevSecOps model in order to adapt it to the integration capabilities of the solution created in the first paragraph of the approach, in line with the previously specified goals of the work.

#### 3.1 General Approach to the Research

The research methodology for creating a solution that can automate the process of repairing vulnerabilities discovered by the use of a DAST scanner and verifying the possibility of its integration into the DevSecOps model is shown in Figure 4. The main phases of the latter consisted of the following steps: analysis of the previously done research in the field; identify (if possible) existing software-based solutions to the established research aim; select the core issue type that was being focused throughout the rest of the research cycle; breakdown of the DevSecOps requirements; actual realisation of the software component for automating remediation phase in CI/CD; evaluation of the developed product and further additional aspects related to future work and potential enhancement of the software delivered.



**Figure 4: Defining stages of the research methodology**

The following section contains the descriptive details regarding Stages 1–4, while Stage 5, 6 and 7 are thoroughly described in the later parts of the work. While Stage 1 was covered in the previous sections of the work (as part of the literature review), Stage 2 of the research process began with an analysis and determination of the existing alternatives to the proposed scheme of the software that could potentially automate the remediation procedure of the DAST scanning. As it was mentioned in the latest section of the literature review, the lack of

existing solutions that fall under these requirements made it reasonable to assume the fact that this field hasn't been properly researched and no solutions have been developed so far.

### **3.2 Identifying the Target Issue Type**

A web-product may be examined for more than 150 vulnerabilities<sup>3</sup> of varied severity and complexity using one of the commonly<sup>4</sup> used DAST scanner called OWASP ZAP. It was required to identify the kind of vulnerability that would subsequently be goal to verification and automatic remediation in order to reduce and focus the subject area of this work. Since 2021, according to data provided by the OWASP resource known as *OWASP Top-10*<sup>5</sup>, the following vulnerability categories have occupied the top five positions: access control issues (A01), cryptographic failures (A02), injection problems (A03), insecure design concerns (A04), and security configuration issues (A05). Since the majority of the vulnerabilities discovered using OWASP ZAP fall into the A05 category, the latter is most suited for the primary objective of this paper. Therefore, in the further development of this work, the main focus was on vulnerabilities of category A05: Security Misconfiguration, which include configuration errors for the secure use of web services, the use of outdated or insecure components, the use of standard passwords and security certificates, non-closed ports, as well as displaying errors with sensitive information for the end user.

### **3.3 Analysing the Format for Presenting Data**

According to the preferences of the author, the OWASP ZAP dynamic scanner has a function that allows a user to create a report on the scanning that has been done. HTML and JSON files are two of the most popular file types that the application standard supports<sup>6</sup>. At the same time, HTML files are created as local web pages, where the scanner creates a large list of sections in which it indicates things like the vulnerabilities found, their severity and risk for the application, options for resolving them, links to helpful sources for fixing them, constructed graphs and diagrams according to the vulnerabilities found etc. Additionally, customers have the option to modify the generated report utilizing a space designated for unique add-ons. While working on the project, it was determined to further examine reports in the common HTML format and choose technological tools that could access the data in the web files produced by OWASP ZAP.

### **3.4 Breakdown of the Enhanced DevSecOps Model's Requirements**

The outcomes of the performed literary analysis were examined for the notion of the DevSecOps model in addition to creating a software component capable of automating the remedy of discovered configuration vulnerabilities in web applications. In order to ensure the possibility of the developed product's implementation in the pipeline in the event of potential model integration, it was essential to consider the specifics of building the CI/CD model during the design stage. These considerations include taking into account the interaction of different components presented, offering flexibility in settings, and optimizing operating time

---

<sup>3</sup> <https://www.zaproxy.org/docs/alerts/>

<sup>4</sup> <https://brightsec.com/blog/owasp-zap/#owasp-zap-tutorial>

<sup>5</sup> <https://owasp.org/www-project-top-ten/>

<sup>6</sup> <https://www.zaproxy.org/docs/desktop/addons/report-generation/templates/>

for this software. In light of the aforementioned primary requirements, it was identified that the DevSecOps concept can include an additional module compatible with DAST scanning. This unit would have the ability analyze any vulnerabilities discovered, attempt to fix them, and initiate a subsequent DAST check to ensure that there are no new vulnerabilities. Additionally, it was essential to structurally allow for manual confirmation of vulnerabilities by users, as well as the capability to turn off this functionality in the event that the developer is accountable for omitting a product version with a known vulnerability (for example, low priority and severity).

The suggested model idea is further described under Section 4.2, which also takes into consideration the integration of the created software module for automating the repair of configuration problems.

### **3.5 Defining the Evaluation Criteria for Software**

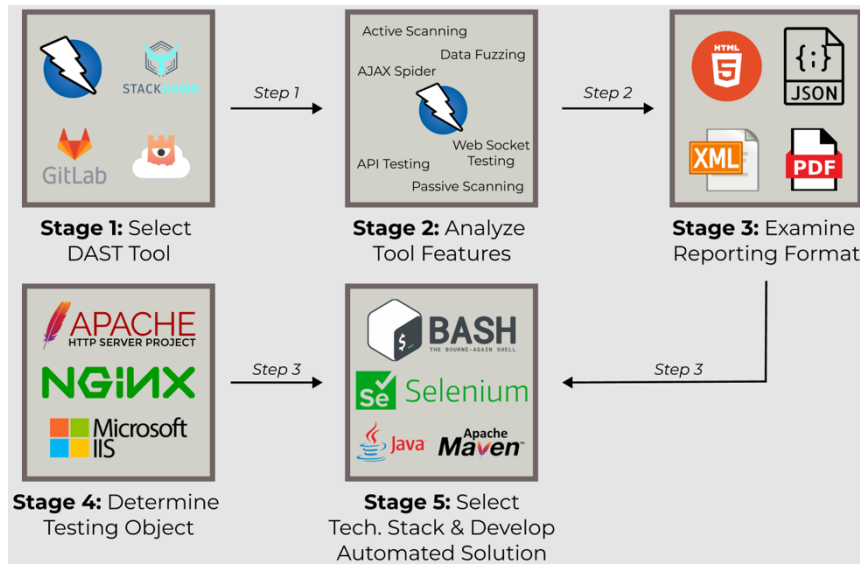
The techniques for assessing the generated program for conformity with different standards, measuring technical qualities, and performing comparative analysis are one of the last steps of the research approach. It is not feasible to conduct a fair comparison analysis because no analogues for the produced software component were discovered throughout the course of the literature review. However, the essential technical aspects that may be utilized for a comparison assessment with a more sophisticated solution in the future were represented in the performance evaluation stage of the developed program. The program's speed, precision, versatility, adaptation to different systems and launch settings, resource consumption, and other factors were among the primary suggestive qualities found at the stage of establishing a study methodology. Section 6 provides more information regarding the outcomes of the program's performance review.

## **4 Design Specification**

This section describes how a developed software can automatically assess live scan results for a web product, interpret them for future usage, conduct adjustments to fix misconfiguration issues, and then retest the changes implemented into the configurational files. Additional information about the roadmap workflow that was followed during the development of the software component is provided in Section 4.1, and the architecture of the DevSecOps model is described in Section 4.2, with the option of incorporating the component from Section 4.1 into its design scheme.

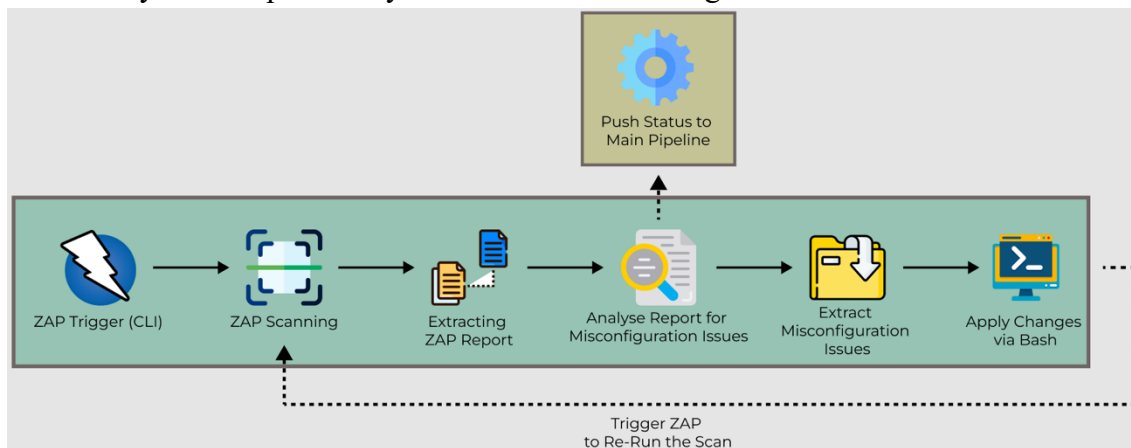
### **4.1 Automated Remediation Tool Design**

In order to narrow down the research scope and define the exact implementation features, a workflow roadmap was created that is shown on Figure 5. The main steps of the research route involved selecting a DAST tool from the available options, evaluating its features and the format in which results were presented, choosing an object for testing, picking a technology stack for putting a software solution into practice, and considering metrics that could be used to demonstrate the benefits of the developed approach.



**Figure 5: The overall roadmap for specifying the workflow of the application developed**

While the majority of the aforementioned stages have been discussed earlier in the previous sections of this work, a separate accent was made to analyse the capabilities of different frameworks that provide a toolset to work with different data formats supported by OWASP ZAP (such as .html or .xml), as well as the server-side stack that was aimed to get changed while the ZAP scanner detects any misconfiguration issues. As a result, the final architecture is described on Figure 6: a separate CI/CD pipeline component is created to run the DAST scan using the OWASP ZAP tool and analyse results of the generated report. The rest of the work relies on a set of scripts that extract the misconfiguration issues into a separate file, apply additional changes (validation of the findings) and pass the valid options to a separate module that applies new configuration settings through the Apache server using both Bash and Java capabilities. After that, the ZAP scanning procedure is re-initiated in order to verify that the previously found issues are no longer exist in the new scan.



**Figure 6: Architecture of the proposed Java and Selenium based solution to automate the remediation of the OWASP ZAP scanning results**

In addition, every time the report is being analysed for the presence of misconfiguration issues, it pushes a status code to the main pipeline, indicating whether the current scanning

result obtains any flaws or if it is free to proceed with the application deployment in the DevSecOps model.

## 4.2 Enhanced DevSecOps Model Design

Speaking of the DevSecOps model, Figure 7 shows an improved scheme on how a previously mentioned software could fit with its own architecture and the defined workflow into the existing standards of the DevSecOps loop. For instance, the settings could be set in such a way to permit OWASP ZAP to start scanning the application after the job is being passed to the Code Deployment phase. As it was mentioned previously, by referring to the report analysis status, the pipeline decides whether it needs to proceed with the release or wait until the software conducts the remediation procedure to ensure that the build has no flaws documented.

Furthermore, the proposed design of the enhanced DevSecOps model brings the idea to manually skip the report checking phase and proceeding with the release, when such actions need to take place in the pipeline (e.g., in case of hotfixes or continuous loops being made in the remediation software part of the pipeline).

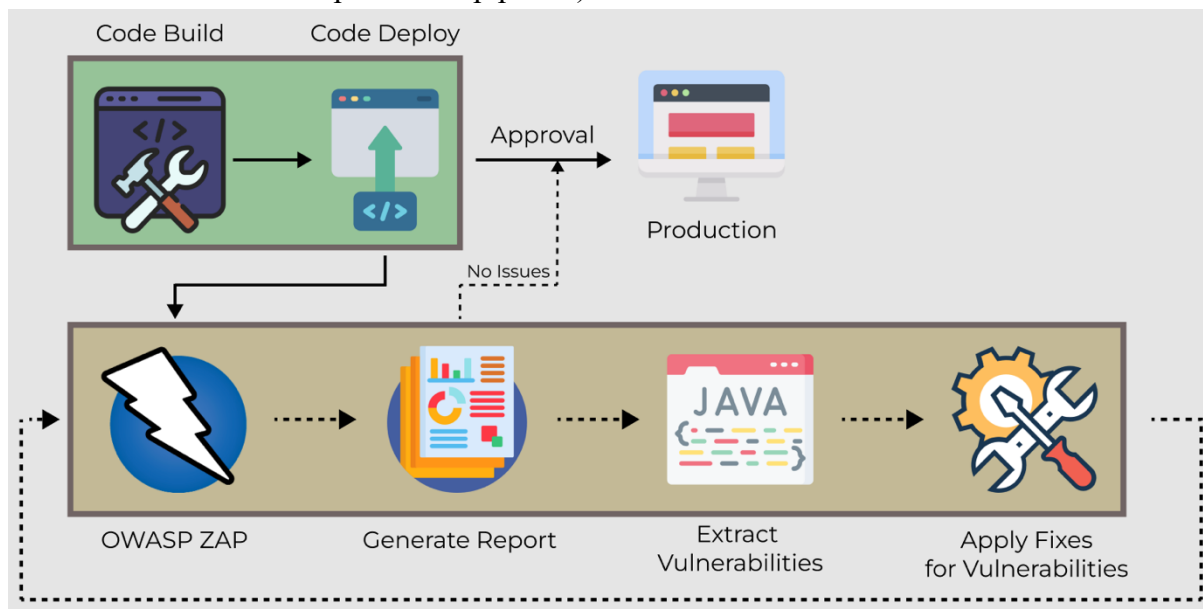


Figure 7: Proposed DevSecOps model including vulnerability remediation component

## 5 Implementation

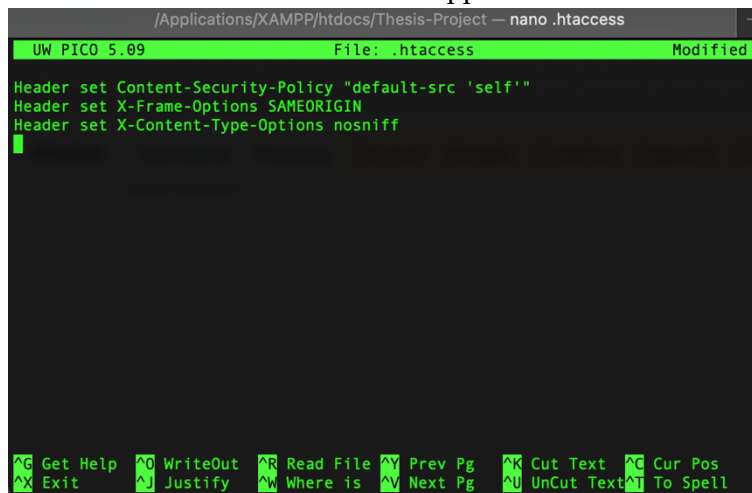
The implementation phase of software for the automatic remediation of vulnerabilities and configuration errors discovered using the OWASP ZAP scanner are shown in this section. Based on the previously built architecture and logical model, a list of main tools and approaches is stated in the following roster:

- Selenium framework and Chrome WebDriver are two of the primary technologies used to efficiently extract information from a report in the .html format. The application may analyze a previously created OWASP ZAP report for vulnerabilities linked to missing response headers by combining the skills of this tool with the web

driver. For further processing, the results of this function are written to a different file stored in the same directory, as the main program;

- The produced file is analysed by a second Java-based module, which is also in charge of removing any instances of the headers that were extracted in the previous phase that were repeated. With this strategy, the developers have the chance to have access to a special set of headers that must be added to the web server configuration that hosts the web application. The present realization entails executing a bash-oriented line of code that alphabetically sorts the file and eliminates any duplicate rows, leaving just one instance of each header missing;
- The information on the available headers and their initial default settings is being stored via the final Java-based module. The next component of the software ensures that all pertinent headers are added to the `.htaccess` file (Apache web server configuration file) and saves the modified settings when a file with unique headers is produced in the previous stage. The `BufferReader` and `BufferWrite` methods, which are included in the Java language, constitute the foundation for all file-related operations under this step;
- The application comes with an optional add-on that may be activated if necessary to start the scanning process both before and after changing the configuration file (by using the OWASP ZAP CLI command set).

While combining all the aforementioned libraries and tools under the single module, an independent application was written that addresses the issues related to setting the configuration headers in the Apache web-server settings (Figure 8). The OWASP ZAP scanner was initialized, scanning settings were established, and the address to be scanned was supplied using a command that was defined in the application code.



```
Applications/XAMPP/htdocs/Thesis-Project — nano .htaccess
UW PICO 5.09 File: .htaccess Modified
Header set Content-Security-Policy "default-src 'self'"
Header set X-Frame-Options SAMEORIGIN
Header set X-Content-Type-Options nosniff
^G Get Help ^O WriteOut ^R Read File ^Y Prev Pg ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where is ^V Next Pg ^U UnCut Text ^T To Spell
```

**Figure 8: Realization of automatic addition of missing headers to solve the misconfiguration issues raised by the OWASP ZAP scanner**

## 6 Evaluation

The findings of the critical analysis performed about the creation of a software solution that may automate the process of correcting misconfiguration issues discovered by the OWASP ZAP scanner are presented in this section. It was already indicated earlier in the course of the

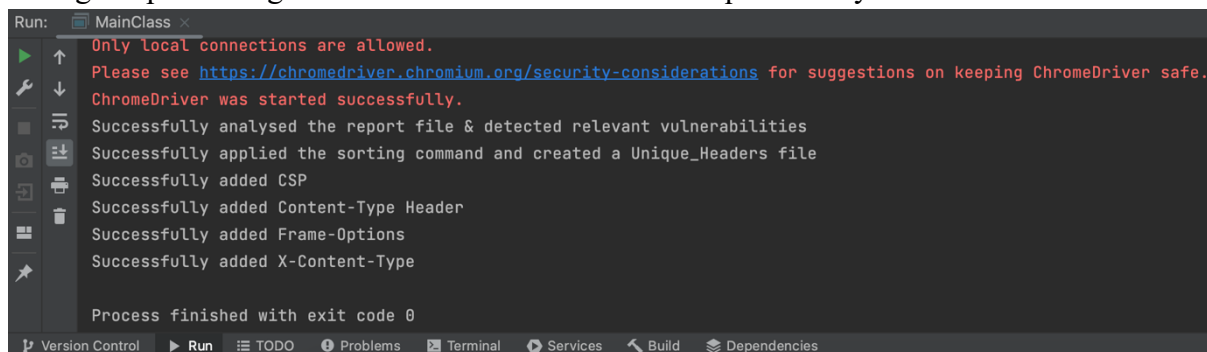


study that existing solutions covering this area and addressing previously given issues were not identified when assessing existing realisations. As a result, it is impossible to carry out a practical comparison study. In this case, a single produced software was examined by the author of this work in order to determine whether this product complied with numerous pre-established features.

## 6.1 Case Study 1: Technical-Oriented Features

Analyzing the technical characteristics, one can express an assessment of such parameters as performance, speed, the load on system resources, and the accuracy of the execution of given commands. Since the developed program is written mostly in the Java programming language, all the main limitations on the performance of this program depend on the Java virtual machine (JVM), which interprets the bytecode with the written program. In addition, the program itself performs only simple operations with input and output streams, working with files at the level of reading and writing information, as well as working with the Selenium library and the Chrome web driver, which does not imply the occurrence of critical errors associated with the program environment.

As for the coverage of the previously set condition for the program to work, the main emphasis in the work of the component was on fixing misconfiguration errors associated with missing headers in the Apache web server settings. Other vulnerabilities found during the operation of the OWASP ZAP scanner are ignored because they do not have an implementation for their remediation. Figure 9 displays the outcome of the program's successful execution, with an emphasis on the primary text prompts that were produced during the processing of each distinct function in the sequence they were invoked.



```
Run: MainClass x
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
Successfully analysed the report file & detected relevant vulnerabilities
Successfully applied the sorting command and created a Unique_Headers file
Successfully added CSP
Successfully added Content-Type Header
Successfully added Frame-Options
Successfully added X-Content-Type

Process finished with exit code 0
```

Figure 9: Example of successful software execution and vulnerability remediation

Despite the fact that the coverage of fixing the found vulnerabilities comes down to problems with setting headers in the Apache configuration file, the program detects missing headers with absolute accuracy, organizes them in a separate file, and also controls the absence of duplicates in the web server configuration file, which means that it can an error occurs on startup.

## 6.2 Case Study 2: Usability and Versatility

One of the important characteristics is the overall usability and user friendliness of the developed application. Since the current development assumed its subsequent integration into the DevSecOps model, there was no native UI interface, and all work was carried out through



the command line and the development environment console. In fact, the initial parameters were set in the source code, since when this software is connected to the CI/CD pipeline, different values are transferred to these fields each time a new build is generated.

Speaking of the platforms supported by the developed application, one of the main prerequisite checks requires to verify the origin of the operation systems that hosts the program. As per initial design, a UNIX-based platform (namely macOS) was used while developing and testing the aforementioned application. Due to the fact that other OS (such as Windows) implement altered shell syntax, it was decided that the main aim would be focused on the UNIX-based OS because of the potential integration into the CI/CD pipeline.

### **6.3 Research Discussion**

During the critical evaluation of the software component, it was proved that this implementation covers the gap found in the current DevSecOps development trend, namely the lack of a solution for automating error correction after DAST scanning, as part of the CI/CD architectural solution.

Despite the relative ease of implementation of the program, as well as its clear focus on one of the types of vulnerabilities - misconfiguration issues, this approach has a number of limitations. For example, the implementation of a program made through bash scripts implies the execution of this program only on UNIX-like systems. In addition, the parameter values for the web server response headers, like other parameters, are set manually when working with the HashMap collection.

Notwithstanding these limitations, the current implementation is able to improve the designated DevSecOps model not only by increasing the level of security of the released web application, but also by relieving the developer's task of fixing the vulnerabilities found manually, but only to audit and control the operation of automatic error correction and, if necessary, to make their own manual changes.

## **7 Conclusion and Future Work**

To put it concisely, it can be implied that the designed, programmed and evaluated solution is able to increase the percentage of implementation of secure software by correcting a category of misconfiguration errors related to the absence of the necessary headers in the responses of the web server to the client. Moreover, according to the previously set tasks and research questions at the beginning of this work, the option of developing a DevSecOps model was considered, which could implement the realized software product, thereby increasing the level of security in delivery architecture by totally solving the issues related to response headers misconfiguration.

Speaking of the future work aspects, the application could be adapted for usage for different platforms as a standalone program with its own user interface, or practically implemented into the existing DevSecOps model. Moreover, another subcategory of misconfiguration issues could be addressed in order to increase the coverage of the software capabilities.

## References

- Ahmed, Z. and Francis, Shoba.C. (2019) ‘Integrating Security with DevSecOps: Techniques and Challenges’, in *2019 International Conference on Digitization (ICD)*. Sharjah, United Arab Emirates: IEEE, pp. 178–182. Available at: <https://doi.org/10.1109/ICD47981.2019.9105789>.
- Al Hayek, W.Y. and Abu Odeh, R.A. (2020) ‘Cloud ERP VS On-Premise ERP’, *International Journal of Applied Science and Technology*, 10(4). Available at: <https://doi.org/10.30845/ijast.v10n4p7>.
- Almeida, F., Simões, J. and Lopes, S. (2022) ‘Exploring the Benefits of Combining DevOps and Agile’, *Future Internet*, 14(2), p. 63. Available at: <https://doi.org/10.3390/fi14020063>.
- Anjaria, D. and Kulkarni, M. (2021) ‘Effective DevSecOps Implementation: A Systematic Literature Review’, *Revista Gestão Inovação e Tecnologias*, 11(4), pp. 4931–4945. Available at: <https://doi.org/10.47059/revistageintec.v11i4.2514>.
- Arnold, B. and Qu, Y. (2020) ‘Detecting Software Security Vulnerability during an Agile Development by Testing the Changes to the Security Posture of Software Systems’, in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*. Las Vegas, NV, USA: IEEE, pp. 1743–1748. Available at: <https://doi.org/10.1109/CSCI51800.2020.00323>.
- Carturan, S. and Goya, D. (2019) ‘Major Challenges of Systems-of-Systems with Cloud and DevOps – A Financial Experience Report’, in *2019 IEEE/ACM 7th International Workshop on Software Engineering for Systems-of-Systems (SESoS) and 13th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (WDES)*. Montreal, QC, Canada: IEEE, pp. 10–17. Available at: <https://doi.org/10.1109/SESoS/WDES.2019.00010>.
- Chen, S.-J. *et al.* (2022) ‘The Impact of the Practical Security Test during the Software Development Lifecycle’, in *2022 24th International Conference on Advanced Communication Technology (ICACT)*. PyeongChang Kwangwoon\_Do, Korea, Republic of: IEEE, pp. 313–316. Available at: <https://doi.org/10.23919/ICACT53585.2022.9728868>.
- Chen, T. and Suo, H. (2022) ‘Design and Practice of Security Architecture via DevSecOps Technology’, in *2022 IEEE 13th International Conference on Software Engineering and Service Science (ICSESS)*. Beijing, China: IEEE, pp. 310–313. Available at: <https://doi.org/10.1109/ICSESS54813.2022.9930212>.
- Faber, F. (2020) ‘Testing in DevOps’, in S. Goericke (ed.) *The Future of Software Quality Assurance*. Cham: Springer International Publishing, pp. 27–38. Available at: [https://doi.org/10.1007/978-3-030-29509-7\\_3](https://doi.org/10.1007/978-3-030-29509-7_3).
- Grishina, A. (2022) ‘Enabling automatic repair of source code vulnerabilities using data-driven methods’, in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. Pittsburgh Pennsylvania: ACM, pp. 275–277. Available at: <https://doi.org/10.1145/3510454.3517063>.

- Huang, K. *et al.* (2022) ‘Repairing Security Vulnerabilities Using Pre-trained Programming Language Models’, in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. Baltimore, MD, USA: IEEE, pp. 111–116. Available at: <https://doi.org/10.1109/DSN-W54100.2022.00027>.
- Ibrahim, A., Yousef, A.H. and Medhat, W. (2022) ‘DevSecOps: A Security Model for Infrastructure as Code Over the Cloud’, in *2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*. Cairo, Egypt: IEEE, pp. 284–288. Available at: <https://doi.org/10.1109/MIUCC55081.2022.9781709>.
- Lee, J.S. (2018) ‘The DevSecOps and Agency Theory’, in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. Memphis, TN: IEEE, pp. 243–244. Available at: <https://doi.org/10.1109/ISSREW.2018.00013>.
- Mell, P.M. and Grance, T. (2011) *The NIST definition of cloud computing*. 1 edn. NIST SP 800-145. Gaithersburg, MD: National Institute of Standards and Technology, p. NIST SP 800-145. Available at: <https://doi.org/10.6028/NIST.SP.800-145>.
- Mohan, V. and Othmane, L.B. (2016) ‘SecDevOps: Is It a Marketing Buzzword? - Mapping Research on Security in DevOps’, in *2016 11th International Conference on Availability, Reliability and Security (ARES)*. Salzburg, Austria: IEEE, pp. 542–547. Available at: <https://doi.org/10.1109/ARES.2016.92>.
- Pinconschi, E., Abreu, R. and Adao, P. (2021) ‘A Comparative Study of Automatic Program Repair Techniques for Security Vulnerabilities’, in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. Wuhan, China: IEEE, pp. 196–207. Available at: <https://doi.org/10.1109/ISSRE52982.2021.00031>.
- Rajapakse, R.N. *et al.* (2022) ‘Challenges and solutions when adopting DevSecOps: A systematic review’, *Information and Software Technology*, 141, p. 106700. Available at: <https://doi.org/10.1016/j.infsof.2021.106700>.
- Rangnau, T. *et al.* (2020) ‘Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines’, in *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*. Eindhoven, Netherlands: IEEE, pp. 145–154. Available at: <https://doi.org/10.1109/EDOC49727.2020.00026>.
- Sen, A. (2021) ‘DevOps, DevSecOps, AIOps- Paradigms to IT Operations’, in P.K. Singh *et al.* (eds) *Evolving Technologies for Computing, Communication and Smart World*. Singapore: Springer Singapore (Lecture Notes in Electrical Engineering), pp. 211–221. Available at: [https://doi.org/10.1007/978-981-15-7804-5\\_16](https://doi.org/10.1007/978-981-15-7804-5_16).
- Simonjan, J., Taurer, S. and Dieber, B. (2020) ‘A Generalized Threat Model for Visual Sensor Networks’, *Sensors*, 20(13), p. 3629. Available at: <https://doi.org/10.3390/s20133629>.
- Sojan, A., Rajan, R. and Kuvaja, P. (2021) ‘Monitoring solution for cloud-native DevSecOps’, in *2021 IEEE 6th International Conference on Smart Cloud (SmartCloud)*. Newark, NJ, USA: IEEE, pp. 125–131. Available at: <https://doi.org/10.1109/SmartCloud52277.2021.00029>.
- Sun, X. *et al.* (2021) ‘Design and Implementation of Security Test Pipeline based on DevSecOps’, in *2021 IEEE 4th Advanced Information Management, Communicates*,

*Electronic and Automation Control Conference (IMCEC)*. Chongqing, China: IEEE, pp. 532–535. Available at: <https://doi.org/10.1109/IMCEC51613.2021.9482270>.

Ur Rahman, A.A. and Williams, L. (2016) ‘Software security in DevOps: synthesizing practitioners’ perceptions and practices’, in *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*. Austin Texas: ACM, pp. 70–76. Available at: <https://doi.org/10.1145/2896941.2896946>.

Zech, P. (2011) ‘Risk-Based Security Testing in Cloud Computing Environments’, in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. Berlin, Germany: IEEE, pp. 411–414. Available at: <https://doi.org/10.1109/ICST.2011.23>.