Security Tools in DevSecOps

- A Systematic Literature Review

Authors: Joel Martelleur & Amina Hamza

Semester: VT 2022

Supervisor: Dr Rüdiger Lincke

Subject: Computer Science

Abstract

DevSecOps emerged to mitigate the challenges of integrating security into DevOps. DevOps have grown tremendously, leading to difficulties in integrating security tools in its development process while maintaining speed and agility. This study aims to investigate the security tools in DevSecOps and how they have been reported in previous literature. The main objective of this study is to provide a knowledge base concerning security tools in DevSecOps that can be used to mitigate challenges regarding the selection and use of security tools in the context of DevSecOps. A systematic literature review was adopted for the research. The study collected a total of 228 studies published between 2015 and 2022; fourteen of these papers were selected to be used for data extraction after conducting a thorough review protocol.

This study has identified thirteen security tool categories used or recommended to be used in DevSecOps. These tools have been structured into seven phases of the development process and five security practices. Additionally, this study has identified twelve drawbacks and sixteen recommendations concerning the use of these security tools in DevSecOps.

The security tools categories, recommendations, and drawbacks identified in this study could potentially be used to facilitate the challenges of selecting and using security tools in DevSecOps and similar methodologies that rely on automation and delivering software frequently.

Keywords: DevSecOps, DevOps, Security Tools, SDLC phases, Shift Security to the Left, Continuous Security, Automation, Systematic Literature Review

1 Introduction

The use of software development methodology can affect the overall work of a software project. Examples of software development methodologies that have gained attention include DevOps, Waterfall, and Agile Development [1]. DevOps has been defined by R. W. Macarthy and J. M. Bass [2] as a methodology that relies on collaboration between cross-functional teams and automation to accelerate and improve the development, testing, quality, and release processes in a more reliable and faster approach. In the works of [1], the authors observed that security was often left until the end of DevOps projects leading to insecure software. The observation led [1] to investigate DevSecOps, which they describe as bringing security to all phases in the Software Development Life Cycle (SDLC).

R. Mao [3] describes DevSecOps as an extension of DevOps to integrate and improve security in DevOps. Even though DevSecOps is a new concept, larger organisations have already embraced it, e.g., GitLab [4] predicts that the DevSecOps practice will grow in 2022 and that the practice will lead to faster and more secure deployments.

A Systematic Literature Review (SLR) is used for this study to find as much relevant research on DevSecOps as possible. The approach is to help make a conclusion based on reliable sources. The method will include a rigorous review of peer-reviewed papers. Aside from fulfilling the requirements for a bachelor's degree in computer science, this study aims to identify and analyse the various security tools used in DevSecOps through an SLR. Ultimately, this study seeks to add to the knowledge base of the emerging DevSecOps methodology.

1.1 Background of Study

The project's research area and application area concerns DevSecOps and security tools used in DevSecOps. The following section provides background concerning DevSecOps, related work, research questions, knowledge contribution, and scope for the research.

1.1.1 DevOps

It is hard to discuss DevSecOps without mentioning DevOps. DevOps can be described in many ways. T. A. Limoncelli et al. [5] described DevOps with *The Three Ways of*

DevOps; 1) *The Workflow,* 2) *Improve Feedback,* and 3) *Continual Experimentation and Learning*, the following text is a summary of their description:

The first DevOps approach, *The Workflow,* is described by [5] as a development process with phases that provides value to the organisation when it moves from business and development phases to operational phases. Additionally, [5] continued to describe that the workflow should be repeatable, fast, and reliable and that the workflow should not contain local optimisations, redundancy, or pass defects to subsequent phases. To improve speed and quality, they recommend automation and that testing should be done in every phase of the process.

The second DevOps approach, *Improve Feedback*, is described by [5] as s the importance of shortening and amplifying the feedback and making the feedback visible between the phases in *The Workflow*. Furthermore, [5] continued to describe that each phase in the Workflow should serve the next phase as a customer and that communication, feedback, and understanding between the phases are vital even if both phases are internal in an organisation.

The third DevOps approach, *Continual Experimentation and Learning* is described by [5] as the importance of encouraging learning and experimenting. [5] again argued that practitioners must learn from experiments and failures and that practices and repetitions are essential to becoming skilful.

Addition to *The Three Ways of DevOps* [5] described DevOps as a methodology that is built on top of Agile Development and Continuous Delivery. They described DevOps as an extension of Agile Development to system administration and Continuous Delivery with practices and principles that relies on automation for continuously releasing software. They also added that Agile Development and Continuous Delivery are important for understanding the DevOps methodology. In this study [5], we have adopted the description of DevOps presented by [5], and its relationship to Agile Development and Continuous Delivery. Agile Development and Continuous Delivery are further discussed in Sections 3.2.1 and 3.2.3.

Another important concept of DevOps is Software Delivery Performance, further discussed in Section 3.2.4.

1.1.2 DevSecOps

In the result of R. Mao et al. [3], the authors identified that practitioners of DevOps recognised the importance of security but the security team often leaves the security

task at the end of SDLC. Additionally, [3] described that leaving the security until the end of the SDLC leads to difficulties in assuring security to frequent deployments and changes in the application.

H. Myrbakken and R. Colomo-Palacios [6] found that DevSecOps is an extension of DevOps to integrate security into the DevOps development process through collaboration between the development, security, and operational teams. The definition of DevSecOps presented by [6] is also accepted by other researchers including [3] and M. Sánchez-Gordón and R. Colomo-Palacios [7]. Also, [7] identified the names, SecDevOps and DevOpsSec to be used in the industry for DevSecOps.

The aforementioned description that DevSecOps is an extension of DevOps and the terminology DevSecOps will be used throughout the research.

In the context of DevSecOps [3] also identified the concepts of *Continuous Security* and *Shift Security to the Left* as the two capabilities that can improve DevSecOps; these two concepts are further discussed in Section 3.2.5

1.2 Related work

Before starting this study, the authors conducted exploratory research about the current state of DevSecOps. Most of the exploratory research was on the current academic research concerning DevSecOps. The process gave the authors a good view of the state of research concerning DevSecOps, which will be used as a knowledge base for this study. The related work is further discussed below:

R. N. Rajapakse et al. [8] published an SLR that identified and mapped the relationships between twenty-one challenges, nine future research directions, and thirty-one proposed solutions regarding security tools, practices, practitioners, and infrastructure when implementing DevSecOps. One challenge they identified concerned the selection of security tools in DevSecOps, and the solution they proposed to that challenge was for practitioners to embrace security tool standards for DevSecOps. They also identified challenges in commonly used security tools, and for that, they proposed that practitioners should embrace best practices regarding security tools. They also identified the future research directions that security tools need to be improved to fit the speed and agility that DevSecOps demands and that security tools should be more suited for the development team, not the security team. Regarding the type of security tools, it was observed that they mainly discussed different tools concerning *Application Security*, see Section 3.1.5.

Additionally, Rajapakse et al. [9] conducted an SLR on how practitioners can integrate and use different security tools in DevSecOps. Their research focused on the advantages and disadvantages of selecting and integrating different security tools into the deployment pipeline. In their research, they identified the problem that developers often do not have the knowledge and the skills that security tools demand and that the development team often needs help from the security team to mitigate vulnerabilities in security tool test results. They [9] observed the lack of research regarding security tool selection and that security tools use old underlying technology making security integration more difficult. Also, in [9] it was observed by this study that the discussion of security tools mainly concerned *Application Security*.

V. Mohan and L. B. Othmane [10] explored DevSecOps to find the central characteristics of the methodology. They found, among other things, tools, collaboration, configuration, compliance, automation, and best practices; as some of the main characteristics. They concluded that even though DevSecOps is an emerging methodology with a few publications, the concept embodies fundamental challenges that security and software practitioners need to tackle. The part in the paper that interested us most was the automation and tools for DevSecOps. These two sections detailed how automation and tools have contributed to testing, monitoring, and code review practices.

R. Mao et al. [3] identified *Continuous Security* and *Shift Security to the Left* as the two capabilities that can improve DevSecOps. They also identified that technical automation and integration abilities in security practices enable and improve *Continuous Security* and *Shift Security to the Left*. Some of the security practices they identified were; threat modelling, code reviews, code analysis, application security testing tools, monitoring, secret management, configuration management, version control, and container security. One interesting part of the research was that they clearly identified and described several practices that could improve security in DevSecOps projects. Also, [3] described collaboration, communication, and shared responsibility as important factors for improving security in DevSecOps.

N. Forsgren et al. [11] identified the concept of *Shift Security to the Left* as a capability that can improve security and software delivery performance in DevOps. They describe capability as integrating *Information Security* into the development process instead of putting it at the end of the SDLC. For further reading on *Information Security*, see Section 3.1.1. Also, like R. Mao et al. [3], the research provided by [11]

argued that security in DevSecOps should not be limited to *Application Security* but rather to the broader field of *Information Security*.

1.3 Problem Formulation

Security should be a given part of DevOps. However, due to the previously described common separation of security to the end of the SDLC and the difficulties of integrating security into DevOps, the methodology DevSecOps have grown stronger.

This study aims to build on the knowledge based on previous research regarding DevSecOps and security tools in DevSecOps. To the best of the author's knowledge and related work, selecting and using security tools for DevSecOps projects comes with substantial challenges. This study agrees with R. Mao et al. [3] that *Shift Security to the Left* and *Continuous Security* are the two capabilities that can improve DevSecOps and that technical automation and integration abilities in security practices enable and improve those two capabilities.

R. N. Rajapakse et al. [8, 9] mostly focused on *Application Security* practices and tools and not to the same extent on other security tools and practices. This study agrees with the works of R. Mao et al. [3] and N. Forsgren et al. [11] that security in DevSecOps should include multiple security practices and that *Information Security* should be integrated into the development process. For further readings about security practices and *Information Security* see Section 3.1.

In the related work, Rajapakse et al. [8, 9] also identified the problem that security tools should be evolved to target the development team more rather than the security team. According to [8, 9], security tools need to be improved to fit the speed, agility, and integration challenges that DevSecOps demands.

The challenges concerning practising *Shift Security to the Left* and *Continuous Security* and that security tools should target the development team more this study believes add to the challenges of selecting and using the right security tools for DevSecOps.

This study will focus on identifying and analysing security tools used in the different phases of the SDLC in DevSecOps. Also, security tools' drawbacks, recommendations, and what technical team should be responsible for the security tools in DevSecOps projects will be investigated. The main objective of this study is that the answers to these questions can add to the knowledge base regarding the

challenges of selecting and using security tools in DevSecOps. This study assumes that DevSecOps practitioners hold *Shift Security to the Left* and *Continuous Security* as essential concepts when selecting and using security tools and that security tools structured in common SDLC phases can facilitate the selection of security tools, see Section 3.2.2. With these challenges in mind, we asked the following research questions (see Table 1.1):

Table 1.1: Research questions and their motivation.

Research question	**Motivation**
RQ1: What categories and examples of security tools in DevSecOps have been reported in previous literature? In what SDLC phases are the security tools used?	To investigate if the result from this question could potentially be used to facilitate the challenge concerning the selection of security tools. In addition, to conduct a quantitative analysis to examine whether application security is the most discussed security practice in DevSecOps research.
RQ2: What are the drawbacks and recommendations of the identified security tools in the context of DevSecOps?	This is a follow-up question to RQ1. This is to give a broader picture of the identified tools.
RQ3: What technical team should be responsible for using and configuring the identified security tools in DevSecOps?	In the exploratory research, it has been observed that related work did not directly express the technical team responsible for specific security tools. This study assumes that investigating this gap is meaningful for practitioners in DevSecOps when selecting security tools for a project with practitioners with a specific skill set.

1.4 Knowledge Contribution & Motivation

Insecure software and software systems can, among other things, cost an organisation money. For example, a report from IBM [12] showed that 83% of 550 organisations over 17 countries and 17 fields had more than one data breach between March 2021 and March 2022. Additionally, [12] elaborated that the average data breach cost was USD 4.35 million during the stated period. Besides the cost, [12] also identified attack vectors for these data breaches, e.g., vulnerabilities in third-party software. The result presented by [12] is one example that indicates the need to improve security when developing and deploying software.

Related work by Rajapakse et al. [8, 9] identified the challenges concerning selecting and use of security tools in DevSecOps. This study believes that answers to the stated research questions can give valuable information concerning security tool selection and use in DevSecOps and similar methodologies.

An objective of this study is that the result can be used by researchers, practitioners, and developers of security tools in the context of DevSecOps and potentially similar methodologies to continue improving the security of software used in the real world.

This study aims **NOT** to identify technical details on how specific security tools can be improved but rather a general high-level knowledge base concerning the different security tools used in DevSecOps.

1.5 Scope and Limitations of the Problem Area

In this section, the limitation of what type of security tools and what type of software engineering methodology this study is investigating are discussed and presented.

1.5.1 Security Tools

In the context of this study, security tools are defined as independent tools designed to facilitate or improve security practices related to the target group, see Section 1.6. The motivations for the definition are to clarify what type of security tools this study is investigating.

Exclusion list of tools:

- Programming Languages and IaC languages.
- Tools/services found in specific cloud platforms, e.g., GCP, AWS, Microsoft Azure.
- The tool's main scope is not security-related, e.g., monitoring tools and logging tools.
- The security tool is not related to web development.

1.5.2 Software Engineering Methodology

In this study, we investigate DevSecOps and security tools in the context of DevSecOps. As mentioned earlier in Sections 1.1.1 and 1.1.2 DevSecOps can be seen as an extension of DevOps [3, 6], and DevOps can be seen as a natural outgrowth of Continuous Delivery and Agile Development [5]. Even if these methodologies are considered

similar to DevSecOps and may use security tools in a similar approach as in DevSecOps, they are **NOT** included in the scope of this study. The motivation is to make this study more distinct and restricted to the DevSecOps methodology.

1.6 Target group

This study targets web development teams working with DevSecOps practices and the general academic community in computer science. Though the paper is focused on DevSecOps, projects following guidelines and practices related to Continuous Delivery, DevOps, and Agile Development may also use this study due to the similarities of these methodologies previously discussed in Sections 1.1.1, 1.1.2, and 1.5.2.

1.7 Structure of thesis

This study first introduces DevSecOps, the various terminologies relevant to the research, and establishes the rationale for conducting the research. Section 2 defines the SLR methodology, including the search process, data synthesis, reliability and validity, and ethical considerations. Section 3 seeks to provide a wider insight into the concepts and practices of DevSecOps in the form of theoretical background. In Section 4, the results from the SLR for the three research questions are presented. Section 5 discusses the study result. Finally, Section 6 includes the study's conclusion, threats to validity, and suggestions for possible future work.

2 Research Methodology

A Systematic Literature Review (SLR) is the research methodology employed for this study. An SLR was considered to be an efficient methodology for collecting and assessing evidence relevant to the main objective and research questions stated in 1.3. In conducting the SLR, we adhered to the guidelines outlined by B. Kitchenham [13]. As described by [13], the goal of an SLR is to systematically identify, evaluate, and synthesise all relevant research against one or more predefined research questions, phenomena, or research fields.

Before settling on SLR, this study considered and evaluated Case Studies, Surveys, and Controlled Experiments and their suitability to the research.

Initially, we considered using Surveys to evaluate further the results obtained from the SLR. However, Surveys were not realised due to the potential risk that practitioners may be hesitant to disclose security strategies for their projects and the time constraints of this study. A Survey was planned to be conducted with practitioners within the DevSecOps setting, which would have required an additional plan for the research.

Furthermore, Case Studies and Controlled Experiments were excluded as both authors agreed that these methods would be inefficient when assessing evidence under the broader picture of DevSecOps. Controlled Experiments might be an appropriate method for evaluating a specific type of security tool, while Case Studies might be more suitable for examining a software domain with specific security concerns, e.g., financial software.

2.1 Systematic Literature Review, Stages & Activities

B. Kitchenham [13] provided a general guideline for designing and conducting an SLR specific to computer science, which is adopted in this study. Figure 2.1 shows the different activities in the guideline grouped into three stages; *Planning the review*, *Main review Process*, and *Review reporting*. Besides, the different activities outlined, the guideline also encourages the importance of conducting the activities iteratively.

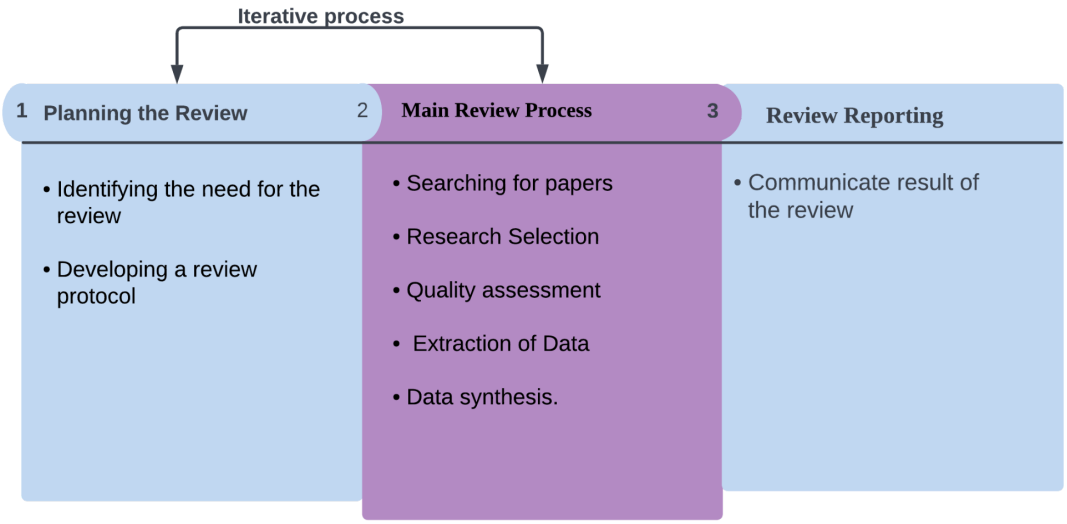

Figure 2.1: Shows the overview of the SLR process and related activities based on B. Kitchenham's guidelines [13].

The first stage, *Planning the Review*, concerns two activities; identifying the motive for the SLR and developing a review protocol to be used when conducting the SLR [13]. The motivation for this study is based on findings from related work described in Section 1.2 and the challenges and main objective formulated in Section 1.3. As previously stated, the main objective of this study is that the answers to these questions can add to the knowledge base regarding the challenges of selecting and using security tools in DevSecOps.

The final version of the developed review protocol and the implementation of the second stage, the *Main Review Process*, are described in Section 2.2.

The third and final stage, *Review Reporting*, involves presenting the result of the review [13]. This stage is evident in Sections 4 and 5, where the results are reported and discussed.

2.2 The Review Protocol

The following section describes the steps for the *Main Review Process* of this SLR.

2.2.1 Development of Research Questions

B. Kitchenham [13] argued that the research question is the most critical part of the review protocol. The guideline described that the research question should either be; 1) useful or meaningful for practitioners and researchers, 2) lead to changes in software engineering practices, or 3) identify inconsistencies in previous knowledge. The

research questions in this study are stated in Section 1.3. The main purpose of these questions is to analyse and synthesise information regarding security tools in the context of DevSecOps. This study considers the stated research questions meaningful to practitioners and researchers in the field of DevSecOps.

2.2.2 Searching for Papers

The databases ACM, IEEE, ScienceDirect, and Springer, as shown in Figure 2.2, were used to search for relevant research papers. The publications in the mentioned databases (ACM, IEEE, ScienceDirect, and Springer) have been peer-reviewed, with several publications within Software Engineering and DevSecOps. After the exploratory research, we observed that these four databases had relevant materials for the stated research questions and concluded using multiple databases minimises the risk of missing some research publications.

Moreover, the search string was replicated in Google Scholar and resulted in similar research materials already collected. Given the outcome of the replicated search, it was decided to limit the search process to the four aforementioned databases as it was assumed that the result would be comparable.

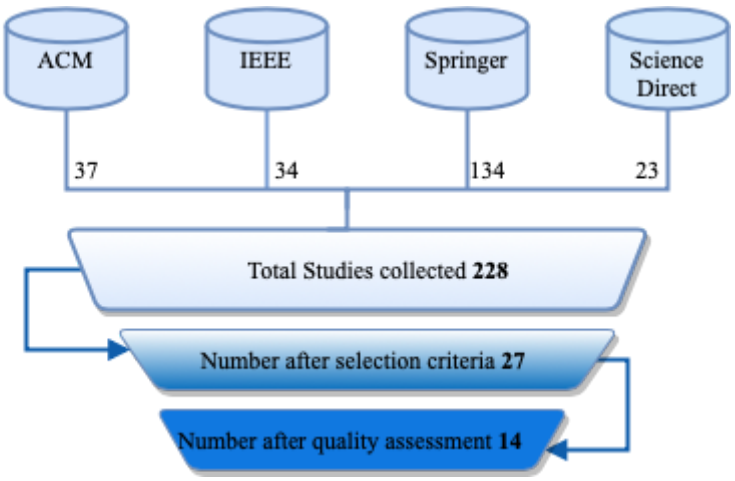

Figure 2.2 An overview of the selection process and the number of studies collected and used.

B. Kitchenham [13] recommends constructing the search string with keywords and synonyms from the research questions. The search string constructed in this research contains the same keywords in all four databases; Science Direct, ACM, IEEE, and

Springer, see Table 2.1. The same keywords and synonyms in the search string ensure that the search results did not vary in the other databases. Most importantly, RQ2 and RQ3 depend on RQ1, and the search string captures all three research questions. In the search, the publication year 2015 to 2022 was used to limit the search to current research. In practice, there were no relevant DevSecOps publications before 2015 that addressed the research question.

Table 2.1: The table displays the search strings and databases used.

Database	Search String
ScienceDirect	(DevSecOps OR SecDevOps OR DevOpsSec) AND tool AND (phase OR stage OR step), Year 2015-2022
ACM	[All: DevSecOps SecDevOps DevOpsSec] AND [All: tool] AND [All: stage phase step] AND [Publication Date: (01/01/2015 TO 12/31/2022)]
IEEE	(("All Metadata": DevSecOps) OR ("All Metadata": SecDevOps) OR ("All Metadata": DevOpsSec)) AND ("Full Text Only": tool) AND (("Full Text Only": phase) OR ("Full Text Only": stage) OR ("Full Text Only": step))
Springer	'DevSecOps or SecDevOps or DevOpsSec AND tool AND (phase OR stage)'

2.2.3 Research Selection

B. Kitchenham [13] recommends using inclusion and exclusion selection criteria to identify and select related research. The selection criteria for this SLR contain two checklists, one for inclusion and another for exclusion. The checklists were created during the development of the review protocol to minimise biases and to help find research aligned with the main objective and research questions according to [13] guidelines.

Inclusion criteria checklist:

- The research has a title, abstract, introduction, conclusion, or keyword relevant to this study's research questions or main objective.
- Full text is relevant to this study's research questions or main objective.
- The research paper must be published in a reputable journal.
- Research from 2015 to 2022.
- The subject area is in the fields of Software Engineering and Computer Science.

Exclusion criteria checklist:

- The research paper is not peer-reviewed.

- The research paper is not in English.
- The full text of the research paper is not available for free for LNU students.
- The research paper's focus is soft qualities in DevSecOps, e.g., collaboration or culture.
- The research paper's main topic does not concern DevSecOps.
- Cloud platforms are the main topic.
- IoT development is the main topic.
- In the case of duplicated studies, only one paper will be considered.
- The research paper does not have any data to be extracted relevant to the research questions.
- The research paper does not pass the quality criteria checklist; see Appendix A.1.

2.2.4 Quality Assessment

According to B. Kitchenham [13], the quality of a conducted research depends on how well a research paper minimises bias and maximises internal and external validity. Additionally, [13] suggests that if the research is focused on evaluating evidence concerning a specific topic, evidence collected from other SLRs should be considered to have the highest quality. To evaluate the quality of the selected papers a quality criteria checklist was created with a focus on methodology, bias, and validity; see Appendix A.1. In addition to assessing the quality of each selected research paper, the motivation for the quality assessment was to determine further whether the research papers contained data that could be extracted relevant to the research questions.

All research papers selected from the previous process were thoroughly evaluated and discussed to assess their quality. If a research paper did not pass all items in the checklist during the process or if a research paper could not be considered to have any relevant evidence to the research questions then the paper was removed from the included papers.

After conducting the quality assessment, fourteen papers were considered to have relevant evidence for the research questions. Since only fourteen research papers were selected, it was decided to review all papers with the same quality regardless of the research methodology. The final result of the selected papers can be found in Appendix A.3.

2.2.5 Extraction of Data

After conducting the quality assessment, B. Kitchenham [13] suggested creating data extraction forms to be used to extract data in an unbiased approach; see Appendix A.2. After constructing the data extraction forms, the researchers proceeded by considering one paper and conducting an independent data extraction, keeping in mind the research question and the quality criteria checklist defined in the previous section. The researchers then compared their findings with each other. When a mutual understanding of the data extraction process was established, the researchers then proceeded to extract data from the rest of the selected papers. To reduce biases and ensure that all relevant data was captured, Author 1 proceeded to go over all the papers Author 2 extracted data from and vice versa. Finally, the extracted data from each select paper was organised into tables in accordance with the previously mentioned data extraction forms.

2.2.6 Data Synthesis

Before synthesising the extracted data B. Kitchenham [13] recommended analysing the extracted data and separating inconsistent data into different tables. During the analysis of inconsistent data, no direct inconsistencies were found between selected papers, e.g., one paper that argues that you should use a specific type of security tool and another paper that argues that you should **NOT** use that specific security tool. Consequently, this study employed the strategy of composite extracted data from all selected papers to synthesise evidence for each research question.

During the process of data synthesis, the researchers in this study analysed and discussed the data relevant to the stated research questions before compositing the evidence. Finally, the evidence has been presented in a descriptive qualitative approach supported by quantitative analysis and summary, see Section 4.

2.3 Conference Rating

It was first planned to include in the inclusion criteria, 2.2.3, the external conference rating system *GII-GRIN-SCIE* [15]. The intention was to employ an external assessment of the quality of the selected papers' conferences before extracting data.

After thoroughly going through the *GII-GRIN-SCIE* system, only seven of the fourteen Selected Papers, see Appendix A.3, used in this study were found to have been presented at some computer science conferences. Six of these papers were presented at a rated conference, whilst one was a work in progress. As a result of the limited number of selected papers that were present in rated conferences, it was decided to exclude the

conference rating from the inclusion criteria. However, the conference rating provided by [15] has been used as an assurance that the quality of the conferences that were found in the rating system was assessed to have good quality. Table 2.2 presents the ratings on the papers found in the *GII-GRIN-SCIE* conference rating system, which was last updated on October 24, 2021.

Table 2.2: Shows selected papers classes, conference ratings, and descriptions based on the conference rating system *GII-GRIN-SCIE* [15].

Selected Papers (P1-P14)[1]	**ConferenceRating**	**Description**
[P6]	A++, A+	Top notch conference
[P5]	A, A-	Very high-quality conference
[P8, P10, P11, P14]	B, B-	Good quality conference
[P12]	Work in progress	Work in progress

[1] Selected papers that were identified after conducting the review protocol described in 2.2, see Appendix A.3.

2.4 Limitation of the Methodology

The following subsections describe the limitation of the research methodology.

2.4.1 Research Material

The focus of this study has been on the topic of DevSecOps and security tools in the context of DevSecOps. However, during the research process, this study encountered different constraints. Notable of these constraints was the inability to access some research material because of missing subscriptions. Therefore, we did not include research papers that required payment or beyond what our student accounts could access.

Another limitation of this study is due to time constraints the exclusion of grey literature, which refers to a type of publication not formally peer-reviewed and published. Although this study has the opinion that there are interesting and valuable materials in grey literature.

2.4.2 Mapping between SDLC Phases and Continuous Delivery Phases

In this study the SDLC phases described in Section 3.2.2 and *Continuous Delivery* practices described in Section 3.2.3 are systematically mapped to each other, see Table 2.6. The mapping is used to be able to interpret and synthesise results from research that uses *Continuous Delivery* to describe the development process. This study considers the mapping between the two different terminologies as a necessity to be able to synthesise and compare data from different selected papers.

Table 2.6: SDLC phases mapped to Continuous Delivery practices[1].

SDLC Phase	Continuous Delivery Practice
Plan	Plan
Code	Continuous Integration
Build	Continuous Integration
Test	Continuous Delivery
Release	Continuous Delivery
Deploy/Operate	Continuous Deployment
Monitor	Monitor

[1] The SDLC phases are described in Section 3.2.2, and the *Continuous Delivery* practices are described in Section 3.2.3.

2.5 Reliability and Validity of Methodology

B. Kitchenman et al. [13] describe that there is no fully agreed definition of research quality, but one way to define research quality is to minimise bias and maximise external and internal validity. Specifically, [13] defines bias as the tendency to produce results that systematically deviate from the actual result, internal validity as the extent to which research is designed and conducted to prevent bias, and external validity as how generalisable the results are. Also, reliability and construct validity are considered essential for this study. Reliability is commonly known to concern the reproducibility of research, and the concept of construct validity can be interpreted as how well constructs and terminology in a research paper can be trusted and understood [14].

The most critical factor in fulfilling the aforementioned validity and reliability concepts is the mindset of the researchers designing and conducting this study. Additionally, the following most vital tool is the predefined review protocol presented in Section 2.2. The rationale behind the review protocol is to conduct the research with

reliability and validity. Validity and reliability are mitigated by carefully following each stage in the review protocol during this study.

2.6 Ethical Considerations

Potentially, this study could affect decisions regarding security in research or when developing and deploying software. These decisions may lead to software security defects, which can be harmful to end-users, customers, and developers of the software. Hence there is a moral obligation to conduct and present this study in a way that does not mislead others in their decision-making regarding tool selection.

There is a risk that this study could mislead and incorrectly classify a tool or provide inaccurate information concerning the tool. Incorrect classification or information could potentially lead to incorrect decision-making concerning the adoption or comprehension of a tool. However, the SLR guideline provided by [13] which was used for this study makes the likelihood of such risks to be at the minimum.

Since the data used for this study was from previous studies, which have been peer-reviewed and extracted carefully, no further ethical issues appear to be pertinent to the thesis. As a result, there is little chance of confidential information being shared. Furthermore, no study involving test subjects or individuals from outside the study was carried out. Therefore, it is unnecessary to examine whether or not participants had consented to participate.

3 Theoretical Background

This section gives a theoretical background to some concepts the authors believe to be important to the overall understanding of DevSecOps. The motivation is to outline and define concepts concerning DevSecOps, Security, and security tools that are of interest to this study.

3.1 IT Security and Security Practices

To improve security it is often recommended to use multiple security strategies and protections. The use of multiple security strategies and protections is commonly referred to as *Defence in Depth* W. Stallings [16]. The following sections discuss Information Security and five security practices concerning Information Security; 1) Application Security, 2) Infrastructure Security, 3) Secret Management, 4) Threat Modelling, and 5) Security Information and Event Management.

3.1.1 Information Security

This study adopts the idea proposed by N. Forsgren et al. [11] that *Information Security* should be integrated into all phases of the SDLC. Additionally, this study also adopts the idea that *Information Security* can be divided into security practices or subfields. One example of such division is; 1) Application Security, 2) Cloud Security, 3) Cryptography, 4) Infrastructure Security, 5) Incident Response, and 6) Vulnerability Management [17].

To integrate Information Security into DevSecOps projects, this study believes a good understanding of *Information Security* objectives is essential. W. Stallings [18] categorised and described the high-level objectives of Information Security; see the following list.

Description of *Information Security* objectives by W. Stallings [18]:

1. *Confidentiality* is the objective that only authorised computer entities (services, systems, etc.) and humans can attain and read the information.
2. *Integrity* is the objective that only authorised computer entities and humans can change the information and that software systems operate in an intended way.
3. *Availability* is the objective that information is available for authorised computer entities and humans when needed.

4. *Authenticity* is the objective of verifying and trusting computer entities or humans. Also, information and the source of information must be verified and trusted.
5. *Accountability* aims to track the actions of computer entities and humans interacting with the system. Tracing actions often include some logging system. This objective also includes that computer entities and humans can not deny action done on specific information.

3.1.2 Threat Modelling

To fulfil the objectives of Information Security and to understand the threats, it is usually wise to do threat modelling to identify a proactive security strategy for building and deploying the software [3].

W. Stallings [19] describes *Attack Surfaces* and *Attack Trees* as two systematic approaches to understanding the current security threats. The *Attack Surface* is described by [19] as exploitable vulnerabilities that can be categorised into Network, Software, and Human attack surfaces. Furthermore, the *Attack Tree* is described by [19] as a tree structure that visualises different paths that an adversary can potentially use to exploit a vulnerability.

3.1.3 Security Information and Event Management

W. Stallings [20] describes Security Information and Event Management as a practice of collecting real-time security information, incidents, and events from multiple different types of security tools. Additionally, [20] identified SIEM as a type of security service often provided by cloud platforms.

3.1.4 Infrastructure Security

The concept of Infrastructure Security concerns protecting different types of networks and the computer resources running in those networks [17].

VMware [21] describes the related terminology Networks Security as one important concern for businesses and organisations to deliver secure applications and services to end-users and customers. They describe that defence in depth or multiple types of security tools should be used to protect infrastructure and applications deployed in a network. Some of the tools they propose are firewalls, IDS/IPS, Sandboxing, and Load

Balancers. This study agrees with the description provided by [21] concerning Network Security, but the related terminology Infrastructure Security will be used.

3.1.5 Application Security

The concept of Application Security concerns designing, developing, and protecting applications after deployment [22]. The target group of this study is web development, and the security tools identified are all related to web development. One often used starting point to understand web applications' current security needs and risks is following the guidelines from OWASP Top 10 [23]. OWASP Top 10 identify the ten most significant security risks for developing web applications and how these risks can be mitigated. For example, one of those security risks is *"Vulnerable and Outdated Components"*, which [23] recommend can be mitigated with, e.g., Software Composition Analysis tools, see Section 3.3.

3.1.6 Secret Management

Managing secrets is often essential when developing and deploying applications. For example, secrets may be shared between systems during runtime communication or when provisioning and updating infrastructure in different cloud platforms. Kief Morris [24] describes four secure approaches for managing secrets in Infrastructure as Code scripts; the following text briefly describes these four approaches.

Secret Injection can be used to manage secrets in Infrastructure as Code scripts. The idea is to parameterise the secrets in scripts and then use runtime injection by, e.g., functionality in secret management tools to provide the value of those secrets [24].

Secret Encryption is another possible approach that can be used if you want to store secrets in public places, e.g., git repositories. The idea is to encrypt the secret and then inject a secret key at runtime to decrypt the secret information [24].

Authorisation without Secrets is an approach where you do not use secrets. Instead of using secrets, a system is configured with authorised access to other services when the system is provisioned [24].

Dynamic Provisioned Secrets is an approach that can be used when dynamically provisioning multiple software systems. In this approach, a secret is automatically created when a system is provisioned and then injected into other systems that need to communicate with the first provisioned system [24].

3.2 DevSecOps History, Concepts, and Practices

The following subsection describes the history, core concepts, and practices of DevSecOps. Although these practices and concepts do not always have standard definitions and can be discussed in multiple books, the following sections briefly describe how this study interprets these concepts.

3.2.1 Software Engineering History

Software engineering methodologies can be seen as frameworks containing recommended practices and principles for designing, building, and delivering software systematically. Some of the more prominent of these methodologies are Waterfall Development, Agile Development, and DevOps; the following text describes those methodologies and how they have led to the DevSecOps methodology.

The Waterfall methodology is a conventional software development methodology that takes a sequential process. The waterfall methodology requires the completion of one phase before the next stage is started [25]. Each sequential phase should be separated and clearly specified. The phases start with requirement elicitation and proceed with coding, analysis, design, testing, and deployment [25, 26]**.** A problem with the Waterfall methodology is that it does not handle the constant changes that software projects are exposed to, e.g., changes in requirements. The challenge of managing change led to a new methodology known as Agile Development.

Agile Developments' most prominent characteristic is its response to changes and uncertainties. Instead of designing, building, and testing software sequentially, Agile Development often utilises an iterative and incremental process [25]. The methodology breaks software development into smaller iterations, where each iteration includes the phases described in the Waterfall Methodology. Each new iteration adds new value to the software and allows the engineers to learn and adapt to understand better what to do in the upcoming iterations. Test automation and test-driven development are often used in Agile Development to support the ability to change and the quality of the software [25]. An important part of Agile Development is the *Manifesto for Agile Software Development*, containing four core values/statements and twelve principles that are used as guidelines for developing software in an agile way. Compared to the Waterfall Methodology, Agile Development is much more suited to tackle changes in software

development. Scrum, Extreme Programming, and OpenUP are examples of individual software methodologies that adhere to the principles and practices of Agile Development. M.J. Thomas et al. [25] describe Agile Development as an iterative method for continuous integration and deployment that has been influenced by the Lean concept. The Lean concept is very close to the Agile concept of iterative development, improved work quality, and faster delivery among others.

DevOps can be seen as an extension of Agile Development to system administration [5]. DevOps is also highly influenced by Continuous Delivery [5, 11]; see Section 3.2.3. N. Forsgren et al. [11] identified and discussed twenty-four capabilities related to DevOps that can improve software delivery performance and software organisational performance, see Section 3.2.4. DevOps promotes collaboration and communication between all teams involved in a software project to ensure a repeatable and reliable development process. To continuously improve the software, the various steps of the DevOps lifecycle are focused on continuous monitoring, continuous delivery, and responding to end-user feedback and other required changes. DevOps, like Agile, emphasises test automation, which enables the team to focus on development rather than manual testing.

As described in Section 1.1.2 practitioners recognise the importance of security but they ignore it due to the limitation of integrating traditional security practices and tools into the DevOps workflow. DevSecOps can be seen as an extension of DevOps with the goal of integrating security processes and controls in DevOps [3, 6].

3.2.2 Phases in DevSecOps SDLC

There is no universal definition for the number of SDLC phases in the DevSecOps development process. However, to build this research on consistent terminology, this research will adopt seven phases: 1) *Plan*, 2) *Code*, 3) *Build*, 4) *Test*, 5) *Release*, 6) *Deploy/Operate*, and 7) *Monitor* [26, 27]. The various phases adopted for this research with some examples of the security practices performed in those stages are summarised below;

Plan Phase: Teams in this stage identify the business requirement regarding the project and software they want to develop. Security practices in this phase can be threat modelling and outlining where and how security testing will be done [26].

Code Phase: Code writing takes place at this stage. The code is written by developers and is committed to a source code repository. Some security practices in this

phase can include code reviews, static code analysis, and pre-commit hooks that improve security and do not impact developers' productivity. The tools in this phase can help developers discover security concerns before committing to the code repository [26].

Build Phase: The build phase starts as soon as developers commit their source code to the repository. This phase ensures the whole codebase is compatible and buildable [27]. Examples of security practices in this phase include dependencies scanning and static application software testing [26].

Test Phase: The test phase is started when the build phase is successful. In this phase, Dynamic Application Security Testing (DAST) can detect live application flows like user authentication, authorisation, SQL injection, and API-related endpoints [26, 27].

Release phase: This phase commences when the build has been tested and is ready to deploy into a production-like environment [27]. Before deployment to the production environment, application code and other dependencies are thoroughly tested.

Deploy/Operate Phase: This research put deployment and operation together. Both of these phases make operations in the live production environment. The deployment phase starts by deploying the tested release from the previous phase to the production environment where real end-users can use the new release [26]. After the deployment phase, automated configuration using IaC tools concerning the underlying infrastructure can be done during the operational phase [27].

Monitor Phase: In this stage, data is collected from the user's experience or logs, and application performance, among others, to help monitor the system [27]. This phase can allow the teams to make informed and data-driven decisions to improve security.

3.2.3 Continuous Delivery

T. A. Limoncelli et al. [5] described that DevOps is built on top of Continuous Delivery and that Continuous Delivery contains practices and principles that rely on automation for continuous releasing software. J. Humble and D. Farley [28] described the values, concepts, practices, and principles of Continuous Delivery.

In their work, they identified eight principles for delivering software: 1) Repeatable and reliable development, test, and release processes; 2) Automating all processes that can be automated; 3) Version control management; 4) If a task is difficult, do it more often and progress; 5) Quality should be built in using automation, testing, and a shared willingness to fix issues; 6) Only released features, fixes, improvements, etc., can be

considered finished; 7) Share responsibility and collaboration among everyone responsible; 8) Use previous delivery for feedback to learn and continuously improve.

Two prominent concepts and practices [28] identified for *Continuous Delivery* were *Continuous Integration* and the construction and use of a *Deployment Pipeline*. The following text briefly describes both of these concepts.

Continuous Integration is described by [28] as the practice of continuously building and testing the application for every change made to it, with the goal that the application should always be in a stable version. They also described practices to accomplish the goal of *CI.* Some of those practices they described are: a version control system, configuration management, daily commits to the main branch, automated and comprehensive test suites, short automated test and build processes, integration server, configuration management, test-driven development, daily commits, do not make changes to a broken build, and the ability to revert back to a stable version. For the practice of *CI,* they stressed the need to have fast tests that take no longer than 5-10 minutes. They propose separating the tests into different stages or creating a *Deployment Pipeline* if longer tests are needed.

Deployment Pipeline is described by [28] as a construct containing automated processes of different stages that take the software from the version control system to the end users. Some of the different stages they discussed and proposed were; the commit stage, the acceptance testing stage, and the release stage. Their description describes the commit stage as building the application and running unit tests and code analysis. They continued the description by recommending running system functional and quality testing for the acceptance tests stage, e.g., performance and security tests. Finally, they [28] proposed deploying the application to a staging or a production environment. Here they describe the staging environment as an identical or almost identical version of the production environment with the difference that real-world end-users can use the application in a production environment.

Additionally to the different stages, [25] described six core practices for the *Deployment Pipeline:* 1) Every change to the version control system should trigger the pipeline; 2) Build the application only once for each change; 3) Use the same deployment script to deploy to every environment with configurable environment parameters; 4) Running critical tests first in every stage to embrace the strategy of failing fast in the delivery process; 5) Deploy to a staging environment that is similar to

the production environment; 6) Stop the pipeline if deployment to any environment fails.

Besides *Continuous Integration* and the *Deployment Pipeline* also *Continuous Delivery* and *Continuous Deployment* are often described as two similar but different practices. *Continuous Delivery* can be described with the goal that the software should always be in a releasable state whereas *Continuous Deployment* can be described with the goal that every released version of the software should also be deployed to end-users [29].

3.2.4 Software Delivery Performance

N. Forsgren et al. [11] stated that more successful software organisations perform better on four metrics related to *Software Delivery Performance* compared to less successful software organisations.

The four metrics related to *Software Delivery Performance* speed and quality are [11]:

1. *Lead time:* Time between a new request (a new feature, security patch, server update, etc.) being discovered and the time when the request is deployed to the production environment.
2. *Deployment frequency:* How often a team deploys their application to the production environment.
3. *Change fail percentage:* How often a change in the production environment leads to a degraded deployment, e.g., a service outage or service that requires rollback or patching.
4. *Meantime to restore (MTTR):* The time it takes to restore from an outage or another system impairment.

Additionally [11] identified and discussed twenty-four capabilities related to DevOps that, according to their research, enable improvement in *Software Delivery Performance*. Included in those capabilities is *Shift Security to the Left,* see Section 3.2.5. In their research, they argued that organisations should focus on improving these twenty-four capabilities instead of trying to implement a specific and predefined maturity of technologies and processes.

3.2.5 Shift Security to the Left and Continuous Security

Shift Security to the Left or *Shift Left* and *Continuous Security* are often two concepts discussed with DevSecOps. Following is how related work describes these two concepts and how this study interprets and views these concepts.

N. Forsgren et al. [11] identified *Shift Security to the Left* as a capability that concerns integrating information security practices and security experts into the entire SDLC. They recommended that security experts can make it easier for the development team and operational team, e.g., by preparing approved secure libraries and packages for the development team and operational team. They also found that *Shift Security to the Left* will improve security and *Software Delivery Performance* as described in Section 3.2.4.

R Mao et al. [3] defined both concepts as the two capabilities for DevSecOps that practitioners should be able to improve. They defined the concept of *Shift Security to the Left* as introducing security practices in all SDLC phases and the concept of *Continuous Security* as a combination of continual learning, improving, and delivering secure software. They also found that technical automation and integration abilities in security practices like security testing, threat modelling, static code analysis, secrets management, container security scanning, unit testing, configuration management, and version control are technical enablers for *Continuous Security* and *Shift Security to the Left*.

R. N. Rajapakse et al. [8] described both concepts to be important practices in DevSecOps. They described *Shift Security to the Left* as the practice of integrating security teams, security tools, and security practices into the early phases of the development process. They motivated *Shift Security to the Left* by reducing security-related costs by identifying and removing security problems early in the development process. They further described *Continuous Security* as a continuous security practice that should have high priority across all phases of the development process and after deployment. Also for *Continuous Security*, they addressed the need to have the security team involved in all SDLC phases.

Even though the interpretation of these two concepts sometimes overlaps, we argue that *Shift Security to the Left* concerns the integration of information security, security team, and different types of security practices and tools into all SDLC phases. The motivations are to facilitate and improve security, security design and implementations, security knowledge and responsibility, break down silos, and that security issues are

often less costly to fix early than late in the development process. In contrast, *Continuous Security* concerns continuously assessing, testing, monitoring, and improving software security across all phases of the SDLC. The motivation is to ensure security in the software during continuous changes and frequent deployments in the software's lifetime. This study has the view that *Shift Security to the Left* is a prerequisite for *Continuous Security* and that a project or an organisation that focuses on improving and implementing these concepts can improve the security of the software, *The Three Ways of DevOps,* and the *Software Delivery Performance* as described in Sections 1.1.1 and 3.2.4. This study's view and interpretation of *Continuous Security* and *Shift Security to the Left* are based on related work findings and descriptions of these concepts presented in this section.

3.2.6 Security Automation and Integration

This study agrees with R. Mao et al. [3] that technical automation and integration abilities in security practices are both enablers for *Continuous Security* and *Shift Security to the Left*, previously discussed in Section 3.2.5, and it is assumed that is true for security tools also. More specifically [3] pointed out the need for security controls and tasks to be automated and integrated into the SDLC and the infrastructure in a dynamic and scalable approach.

3.3 Security Tools

This study has identified thirteen categories of security tools used in the context of DevSecOps. Each of these categories may be divided further into subcategories, and specific security tools may be categorised differently depending on whom you ask. The following sections briefly describe the identified security tools in this research, see Table 4.1.

Threat Modelling tools can help identify, communicate, and understand security threats and are used for modelling and understanding threats, see Section 3.1.4. Threat modelling can include mappings of an attack surface at different levels, e.g., network, infrastructure, application, and human attack surface, to better understand how the application should be protected [19].

Secret Management tools are designed to help manage secrets and different security practices for handling secrets; see Section 3.2.5 for examples of such practices. Besides handling secrets in a secure approach, secret detection tools are also advised to use for

testing that secrets are not published to public repositories. Secret detection can scan for secrets in public repositories to detect unwanted secrets [30].

Static Application Security Testing tools (SAST) are application security testing tools that scan the source code or compiled code without running the application. This category of tool scans for source code patterns that can potentially lead to vulnerabilities [31].

Software Composition Analysis tools (SCA) check known vulnerabilities in the application's dependencies, e.g., third-party open-source libraries without running the application [31]. Compared to SAST they are not detecting patterns in custom source code but are made for detecting vulnerabilities that already are publicly known to exist in libraries [31].

Dynamic Application Security Testing (DAST) is a category of application security testing tool that scans a running application to find vulnerabilities, compared to SCA and SAST which run static tests on source code and Infrastructure as Code scripts [31]. The running application is tested by injecting and executing input that can help find vulnerabilities, e.g., SQL injection [31].

Hybrid Application Security Testing (HAST) and *Interactive Application Security Testing* (IAST) are two modern types of application security testing tools [9]. HAST tools are security tools that unite techniques and power from two or more other security testing techniques. One example is IAST which can be considered a combination of DAST and SAST functionalities accomplished by integrating software agents and sensors into the environment of the running applications [9]. As a result of the more integrated technique, IAST can check if identified source code vulnerabilities are also exploitable when the software is executed [31].

Web Application Firewall (WAF), which works similarly to a reverse proxy, is a type of application security tool installed outside the application and can protect an organisation's entire suite of web applications [32]. However, WAF can only observe and make decisions based on the data that goes in and out of the application, not between internal components [9].

Runtime Application Self-Protections (RASP) is an application security tool that protects and monitors an application's behaviour and requests [30]. By tracking the running application's behaviour, the input, and the output, RASP can detect and protect against threats to the application. RASP is embedded within the application [9].

IaC Linting and Security Testing is a type of infrastructure security testing tool category that statically analyses and identifies linting and security issues in your Infrastructure as Code scripts [33]. Similar to SAST tools these tools scan and test the code, not the running application.

Container Security Scanning is a type of infrastructure security testing tool category that can be seen as a type of SCA that focuses on identifying container vulnerabilities. The scanner analyses the container image and checks if it contains known vulnerabilities [34].

Intrusion Detection and Intrusion Prevention Systems (IDS/IPS) are infrastructure security tools that monitor and prevent computer and network environments from intruders. IDS is a tool that monitors a network and computer environment for behaviour that may indicate a security risk or an attack and then reports those observed security incidents. IPS does what IDS does, and it also actively tries to prevent attacks on the environment [35].

Security Information and Event Management (SIEM) are tools that encapsulate the security practice with the same name described in Section 3.1.6 and are used to aggregate security reports and alerts from multiple used security tools [9].

4 Result

The results presented in this section are extracted and synthesised from selected papers, P1-P14, listed in Appendix A.3. These selected papers and the result presented in Sections 4.1 - 4.3 is the outcome of the conducted review protocol processes and analyses described in Section 2.2.

4.1 Research Question 1

What categories and examples of security tools in DevSecOps have been reported in previous literature? In what SDLC phases are the security tools used?

When synthesising data regarding this research question, this study does not differentiate between a security tool being used more or less than other security tools or if a tool is recommended to be used. A tool used often is presented in the result, similar to a tool used less often or a tool that is recommended to be used. The reason for this presentation of tools is that it has not been possible in a reliable way to extract or deduce information from the collected data on how often a tool is used in DevSecOps, but only that the tool is used or recommended to be used for DevSecOps.

In summary, this study has identified thirteen security tool categories and fifty-eight security tool examples across those categories. The categories and examples have then been grouped into five security practices based on the description of security tools and practices presented in Sections 3.1.2 - 3.1.6, and 3.3. A quantitative presentation of the security tools categories and examples per security practices is presented in Figures 4.1 and 4.2. Whereas Tables 4.1 and 4.2 show in a descriptive approach both SDLC phase(s) and security practice of the identified security tool categories and examples.

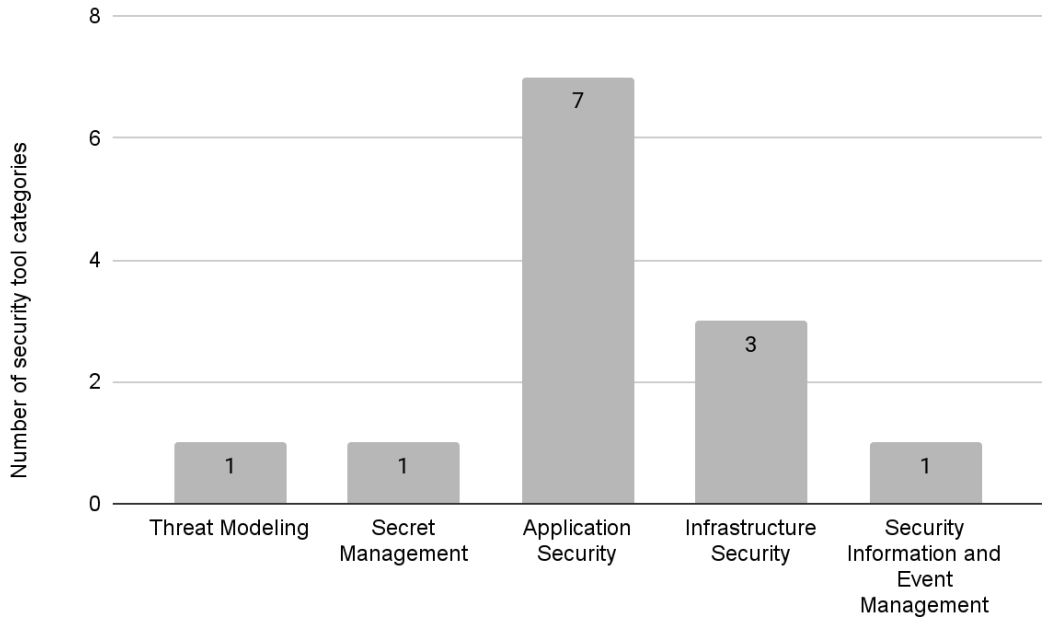

Figure 4.1: Number of security tool categories per security practice. The security tools are grouped into security practices based on the description of the security tools and practices in Sections 3.1.2 - 3.1.6 and 3.3.

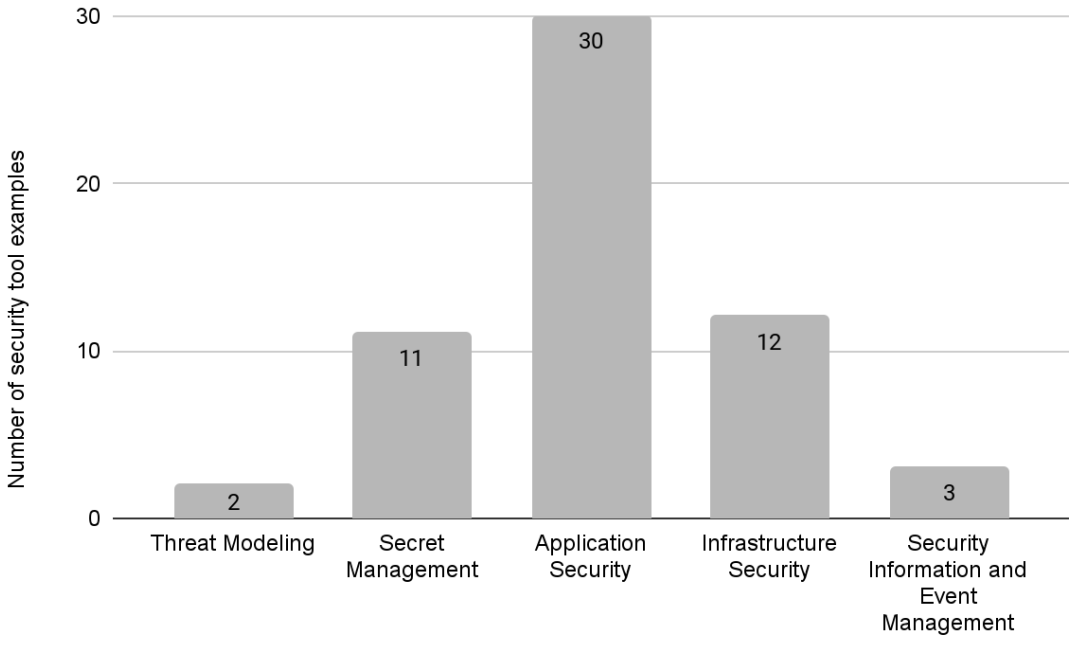

Figure 4.2: Number of security tool examples per security practice. The security tools are grouped into security practices based on the description of the security tools and practices in Sections 3.1.2 - 3.1.6 and 3.3.

1

Table 4.1: Synthesised data from selected papers [P1]–[P11] concerning categories of security tools in DevSecOps with the corresponding SDLC phases and security practices.

Security tool category[1]	**SDLC phase(s)**	**Security Practice**[2]	**Selected Papers**[3]
Threat Modelling	Plan	Threat Modelling	[P3]
Secret Management	Deploy, Operate	Secret Management	[P3, P6, P11]
SAST	Code, Build, Test	Application Security	[P1- P6, P8, P9, P11]
SCA	Build, Test	Application Security	[P3 - P6, P9]
DAST	Test, Release, Operate	Application Security	[P1, P3 - P11]
IAST	Test, Release, Operate	Application Security	[P1, P3, P5]
HAST	Code, Build, Test, Release	Application Security	[P5]
RASP	Deploy/Operate, Monitor	Application Security	[P3, P5]
WAF	Deploy/Operate, Monitor	Application Security	[P5]
IaC Linting and Security Testing	Build, Test, Deploy	Infrastructure Security	[P3, P6]
Container Security Scanning	Build	Infrastructure Security	[P1, P11]
IDS/IPS	Deploy/Operate, Monitor	Infrastructure Security	[P3, P5]
SIEM	Code, Build, Test, Release, Deploy/Operate, Monitor	Security Information and Event Management	[P3, P5]

[1] Security tools categories are described in Section 3.3.
[2] Security tool categories are grouped into security practices based on the description of the security tools and practices in Sections 3.1.2 - 3.1.6 and 3.3.
[3] Data is synthesised from selected papers; see Appendix A.3.

Table 4.2: Synthesised data from selected papers [P3], [P6]–[P10] concerning the categories of security tools with specific tools examples in DevSecOps with corresponding security practice.

Security tool category[1]	**Security tool example**	**Security Practice**	**Number of Tools**	**Selected Papers[3]**
Threat Modelling	Threat Dragon, CAIRIS	Threat Modelling	2	[P3]
Secret Management	Git Secrets, Blackbox, Hashicorp Vault, Transcrypt, Pinterest Knox, GitLeak, CyberArk Conjur, ThoughtWorks Talisman, Berglas, Docker Secrets, Trufflehog	Secret Management	11	[P3, P6]
SAST	SonarQube, Brakeman, PVS-Studio, Veracode, Coverity, Hakiri, Checkmarx	Application Security	7	[P3, P6 , P9]
SCA	Dependency Check, Snyk, Dependabot, PHP Security Checker, Clair, Blackduck, Retire.JS, SRC:CLR, Anchore, Node Security Platform, Trivy, Sonatype, Whitesource, Hakiri	Application Security	14	[P3, P6, P9]
DAST	OWASP ZAP, Arachni Scanner, Vuls, Nmap, SQLMap, Gauntit	Application Security	6	[P3, P7, P8, P10]
IAST	Hdiv	Application Security	1	[P3]
RASP	Hdiv, AppSenor	Application Security	2	[P3]
IaC Linting and Security Testing	Cloud Custodian, Dev-Sec.io, InSpec, Ansible-Lint, Puppet-Lint, Foodcritic, Serverspec, Oscap	Infrastructure Security	8	[P3, P6]
IDS/IPS	Fail2Ban, Snort, Suricata, OSSEC	Infrastructure Security	4	[P3]
SIEM	OSSIM, ThreadFix, OWASP Code Pulse	Security Information and Event Management	3	[P3]

[1] Security tools categories are described in Section 3.3.
[2] Security tools are grouped into security practices based on the description of the security tools and practices in Sections 3.1.2 - 3.1.6 and 3.3.
[3] Data is synthesised from selected papers; see Appendix A.3.

4.2 Research Question 2

What are the drawbacks and recommendations of the identified security tools in the context of DevSecOps?

The presentation of the result for this research question is categorised into drawbacks and recommendations. This study has identified and grouped twelve drawbacks and sixteen recommendations associated with previously identified security tools in RQ1. The findings are presented in Sections 4.2.1 and 4.2.2.

4.2.1 Drawbacks with Security Tools in DevSecOps

This section presents the drawbacks of various security tool categories extracted and synthesised from selected papers [P1, P2, P4, P5, P11, P12]. Table 4.3 shows the number of drawbacks categorised in different security tool categories.

Table 4.3: Drawbacks concerning security tools in DevSecOps.

Nr	Description Drawback[1]	Selected Papers[2]
	SAST	
1	Manual evaluation of false positives (a detected vulnerability that is in fact not a vulnerability) that the tool produces are not suited for frequent releases.	[P1, P5]
2	Practitioners consider the tool to be time-consuming.	[P5]
	DAST	
3	Practitioners consider the tool to be time-consuming.	[P5]
4	Difficult to use and set up when the application has frequent deployments, e.g., to get high test coverage.	[P1, P5]
5	Manual evaluation of false positives that the tool produces is not suited for frequent releases.	[P5]
	WAF	
6	Have a limited understanding of data communication inside the application. For example, communication between components that are built using microservices.	[P5]

Tabel 4.3 is continued on the next page.

Table 4.3 continued.

	General Security tools[3]	
7	Insufficient tool documentation, standards, and knowledge, e.g., configuration and the capacity of the tools used.	[P1, P2]
8	Not optimised or built for DevOps objectives, e.g., automation and frequent deployments.	[P1, P2, P4, P5, P11, P12]
9	Manual evaluation of false positives that different types of security testing tools produce are not efficient for continuous deployments and put extra demand on practitioners.	[P5]
10	Inability to change security tools.	[P1, P2]
11	Integration limitations, e.g., integration of security tools into the deployment pipeline.	[P1, P5]
12	The use of different security tools between different teams leads to collaboration issues.	[P1]

[1] The descriptions are synthesised from selected papers, see Appendix A.3.

[2] Selected papers specific to each drawback.

[3] The category of security tools has been used when a drawback could not be related to a specific security tool category.

4.2.2 Recommendations on Security Tools in DevSecOps

This section presents recommendations concerning various security tool categories extracted and synthesised from selected papers [P1, P2, P4 – P8, P10 – P14]. Table 4.4 shows the number of recommendations categorised in different security tool categories.

Table 4.4 Recommendations concerning security tools in DevSecOps.

Nr	**Description Recommendation[1]**	**Selected Papers[2]**
	SAST	
1	Developers should have the skills to recognise false positives because of the significant amount of false positives the tool produces.	[P1]
2	Should be integrated into the development environment, e.g., an IDE to help developers scan and find vulnerabilities as they code.	[P5]
3	Should be used in parallel in the deployment pipeline with other types of tests, e.g., DAST or unit tests, to improve delivery performance and to get fast feedback.	[P1, P5, P8]

Table 4.4 is continued on the next page.

Table 4.4 continued.

	DAST	
4	Should be used in parallel in the deployment pipeline with other types of tests to improve delivery performance and to get fast feedback.	[P1, P5, P8]
5	Use flexible deployment strategies, e.g., all vulnerabilities should be reported, but it may be ok to deliver the application even if some not critical vulnerabilities are reported.	[P8]
6	Combine DAST tools with combinations of other testing strategies, e.g., Behaviour Driven Development (BDD) to make the tests tool more understandable for non-security expert staff.	[P8]
	IAST	
7	IAST tools are recommended to be used in DevSecOps because they possess the strength of DAST and SAST but do not produce as many false positives and have better testing accuracy.	[P1, P5]
	RASP	
8	RASP tools protect against vulnerabilities that may be missed in security testing when the application is frequently deployed.	[P5]
9	Practitioners recommend using these tools because they have a better understanding of data communication between components or microservices compared to similar tools, e.g., firewalls., which suit DevOps deployment strategies.	[P5]
	General Security tools[3]	
10	Documentation on how to use security tools should be improved.	[P1]
11	Practitioners should use best practices when using security tools.	[P1, P7]
12	Converging towards security tool standards.	[P1]
13	Security tools should be automated when integrating them in DevSecOps to make security and security testing more scalable and predictable when continuously delivering software.	[P1, P2, P4-P7, P9 - P14]
14	Security tools should be functional on multiple platforms.	[P2]
15	Practitioners should be trained appropriately before using security tools.	[P4, P12, P13]
16	Developers require security tools that they favour.	[P5]

[1] The descriptions are synthesised from selected papers, see Appendix A.3.

[2] Selected papers specific to each recommendation.

[3] The category of security tools has been used when a recommendation could not be related to a specific security tool category.

4.3 Research Question 3

What technical team should be responsible for using and configuring the identified security tools in DevSecOps?

This research question investigates what specific team should be responsible for using and configuring the various security tools. The extraction of data regarding this question has been limited to only four selected papers [P1, P4, P5, P12] due to the other selected papers not addressing what technical team should be responsible for a specific tool. However, [P4] provided direct evidence concerning the relationship between a technical team and a security tool category. From the other three papers [P1, P5, P12], it could only be inferred which team should be responsible for the tool, e.g., papers discussing different challenges related to developers using the tools. A common identification from [P1, P4, P5, P12] is that developers often lack the expertise to configure and use the security tools efficiently, but with appropriate training, the tools could be used by the development team. The data extracted to answer this research question could not be used to separate whether a research paper discussed the tools' configurations, usage, or both. Therefore, the usage and the configuration have been merged when presenting the result. The result is shown in Table 4.5.

Table 4.5: Technical team responsible for configuring and using security tools.

Tool Category	**Development team responsible**	**Security team responsible**	**Operational team responsible**
SAST	[P4]	[P4]	-
SCA	-	[P4]	-
General[1]	[P1, P4, P5, P12]	[P4, P12]	-

[1] This is a general category of security tools in the scope of DevSecOps. It has been used when the responsibility could not be related to a specific security tool category.

5 Discussions

The study's objective is to provide further knowledge concerning security tools within DevSecOps. The authors used SLR to investigate the various research questions to achieve this objective. The entire review process has been carefully done and presented. Nonetheless, the constraints and limitations encountered have also been discussed. In this section, the results obtained and presented in Section 4 are discussed.

5.1 Research Question 1

What categories and examples of security tools in DevSecOps have been reported in previous literature? In what SDLC phases are the security tools used?

One motivation for this question was an observation from the exploratory research in this study. Specifically, it was observed that related work by R. N. Rajapakse et al. [8, 9] focused on *Application Security* tools and that security tools regarding other practices such as *Infrastructure Security* could have been overlooked in those papers. The observation led this study to investigate if the inconsistency was true in research regarding security tools in DevSecOps.

After conducting the SLR, thirteen security tool categories and fifty-eight examples of security tools were identified. Based on the description of the security tools and practices in Sections 3.1.2 - 3.1.6, and 3.3, the security tool categories and examples have been divided into the five security practices; *Threat Modelling*, *Secret Management*, *Application Security*, *Infrastructure Security*, and *Security Information and Event Management*. The results presented in Figures 4.1 and 4.2 show that security tools in DevSecOps mostly concern *Application Security*. Specifically, the result shows that against thirteen categories and fifty-eight examples of security tools 1) One security tool category and two security tool examples are related to *Threat Modelling*; 2) One security tool category and eleven security tool examples are related to *Secret Management*; 3) Seven security tool categories and thirty security tool examples are related to *Application Security*; 4) Three security tool categories and twelve security tool examples are related to *Infrastructure Security*; and 5) One security tool category and three security tool examples are related to *Security Information and Event Management*. The findings show that security tools in the context of DevSecOps mostly focus on *Application Security.* The findings are also aligned with the observations made during the exploratory phase of this study. This study has the opinion that the emphasis

on *Application Security* can lead to false security assurance in projects that adhere to DevSecOps. This conclusion is drawn from the potential risk that the other four previously mentioned security practices may not be prioritised.

The other motivation for this question was that the result could potentially be used to facilitate the challenge concerning the selection of security tools discussed in related work by R.N Rajapakse et al. [8, 9] or potentially be used as an input to a tool standard in DevSecOps. A tool standard is a solution that was proposed in related work by R.N. Rajapakse et al. [8] for mitigating the problem concerning the selection of security tools. To investigate if the result from this research question can provide information to these challenges, this study assumes; 1) Security tools structured in common SDLC phases can facilitate the selection of security tools in DevSecOps, 2) Practitioners hold *Shift Security to the Left* and *Continuous Security* as two important capabilities to consider when selecting security tools for DevSecOps projects.

The result concerning RQ1 shows that security tools are available for all SDLC phases which support *Shift Security to the Left* and *Continuous Security* as discussed in Section 3.2.5. Secondly, the result also shows that different security tool categories and examples were used to address multiple security practices, which are also aligned with related work [3] and [11]. This study believes the inclusion of multiple security practices is important and considers this as an approach that could enhance *Defence in Depth* as described in Section 3.1.

Based on support for *Shift Security to the Left*, *Continuous Security,* and *Defence in Depth*, this study has the opinion that the result indicates that the identified security tools structured in the common SDLC phases can potentially facilitate the security tool selection for practitioners and potentially be used as input to a security tool standard for DevSecOps. This conclusion is based on the two previously mentioned assumptions in this section.

A concern against the result of RQ1 is that it was not always clear in which phase the tools were used (e.g., the phase was extracted from context rather than direct statements or due to translations between different terminologies, see Section 2.4.2), which is a concern regarding the validity of the result. Another concern observed in the result of RQ1 presented in Table 4.2 is that some of the identified examples of security tools have changed, e.g., a security tool has become deprecated. An example of a deprecated tool in Table 4.2 is Hakiri. Due to the changes in the tools, the result regarding the examples of security tools risks quickly becoming obsolete.

5.2 Research Question 2

What are the drawbacks and recommendations of the identified security tools in DevSecOps?

The motivation for this question has been to bring out the various drawbacks and recommendations concerning the identified security tools. Besides the recommendation to use security tools related to the practices of *Infrastructure Security*, *Secret Management*, *Threat Modelling*, and *Security Information and Event Management*, this study could not find any more specific recommendation or drawback concerning the use of those tools in DevSecOps. Therefore, the following two sections discuss the drawbacks and recommendations presented in 4.2 grouped into *Application Security tools* and *General Security tools*. The category of *General Security tools* has been used for this research question when a drawback or recommendation could not be tied to a specific category.

When synthesising drawbacks and recommendations, this study realised that they are general and probably vary between projects. Therefore the discussions will focus on generally accepted concepts and objectives of DevSecOps and DevOps. Specifically, *The Three Ways of DevOps*, *Software Delivery Performance, Shift Security to the Left, Continuous Security*, and *Security Automation and Integration*, as described in Sections 1.1.1, 3.2.4, 3.2.5, and 3.2.6.

5.2.1 Application Security tools

One common drawback of SAST and DAST is that they produce false positives and are time-consuming [P1, P5]. The drawback can be mitigated by using IAST [P1, P5] and training developers to manually check for false positives when using SAST [P1]. That developer should be trained is a good start but may not be a final solution to mitigate false positives in DevSecOps. This study believes that the most significant problem with false positives in DevSecOps is that it has to be done manually. The manual effort to mitigate false positives goes against automation which is an important part of DevSecOps [1,3]. The recommendation to use IAST instead of SAST and DAST is interesting and could be analysed further to see if this is a generalisable solution to reduce false positives.

Another common drawback of SAST and DAST is that they are time-consuming [P5]. Parallel testing in the Deployment Pipeline is recommended when using these

types of tools [P1, P5, P8]. By implementing parallel testing both the lead time and the feedback loop will be improved.

Another recommendation for SAST is to integrate these tools into the development environment to help developers when they code [P5]. For DAST it is also recommended to combine these tests with other types of test strategies, e.g., BDD, to make the tests more understandable [P8]. This study believes that these two recommendations can improve the feedback and help developers, security experts, and security non-experts improve their knowledge and interest in security.

WAF tools are installed outside the application they protect and could be a security risk if the application is aggregated by microservices that share sensitive data [P5]. To improve security between microservices, which is common in DevOps, practitioners recommend RASP tools because they have a good understanding of data communication between microservices and because they work as an extra level of protection [P5]. This study believes that collaboration between developers, system administrators, and security teams in the planning phase is important to make the correct choice between using WAF or RASP tools.

5.2.2 General Security Tools

A total of six general drawbacks have been identified in the result of RQ2. The most common drawback was that the tools were not suited for automation and frequent deployments [P1, P2, P4, P5, P11, P12]. Also related to this drawback is the manual evaluation of false positives when frequently deploying applications [P5]. Because DevSecOps relies on both automation and frequent deployment, these two drawbacks are substantial concerns related to using these tools in DevSecOps projects.

Also, the inability to change security tools [P1, P2], insufficient documentation, standards, and knowledge [P1, P2], integrations limitations [P1, P5], and collaboration issues related to the use of different security tools [P1] are all indicator that security tools, in general, are not suited for DevSecOps. These drawbacks become more significant in DevSecOps and similar methodologies that rely on adaptability, speed, shared responsibility, and collaboration to increase quality.

Concerning general recommendations, the most identified recommendation was that security tools should be automated to support common objectives in DevSecOps, e.g., frequent deployments [P1, P2, P4-P7, P9-P14]. Regarding integration, one specific recommendation is that security tools should be functional on multiple platforms [P2].

These automation and integration recommendations or requirements concerning security tools are aligned with R.Mao et al. [3] concerning the importance of automation and integration concerning security practices to improve *Shift Security to the Left* and *Continuous Security*.

Many of the identified recommendations concern making it easier to use and select these tools. These recommendations include better documentation for security tools [P1], adoption of best practices [P1], security tools standards [P1], practitioners should be trained [P4, P12, P13], and developers requiring security tools which they favour [P5].

A conclusion drawn from the general recommendations and drawbacks identified in this research question is that they are often self-explanatory and general but still important and adequate for practitioners of DevSecOps.

5.2.3 Validity issue

When extracting the data for RQ2, two selected research papers with almost the same authors have been the dominant voice, namely [P1, P5]. These two selected papers have also been used in this study as related work, R.N. Rajapakse et al. [8, 9], and they could have led to publication bias in the result of this question.

5.3 Research Question 3

What technical team should be responsible for using and configuring the identified security tools in DevSecOps?

Related work by R. Mao et al. [3] described, among other things, the importance of shared responsibility in DevSecOps. Shared responsibilities also imply that not only security experts should be responsible for security and security tools used in DevSecOps. Also, Rajapakse et al. [8, 9] expressed that the responsibility of security tools should be shared by concluding that security tools should target developers and not security experts to fit DevSecOps better.

In the exploratory research, our research observed that related work did not directly express the technical team responsible for specific security tools. This missing information, together with the assumption that an investigation of this gap is meaningful for practitioners in DevSecOps is also the motivation for this research question.

After conducting the review, only limited information regarding this question could be extracted from the selected papers. The result shows; 1) SAST tools should be used

by the security team and the development teams [P4], 2) SCA tools should be used by the security team [P4], and 3) the development team and the security team should generally be the teams responsible for the security tools in DevSecOps [P1, P4, P5, P12]. No data shows that the operational team should be responsible for the identified security tools. This study has the opinion that the result is inconsistent with the shared responsibility discussed by R. Mao et al. [3] and that security should be integrated into all SDLC phases as described in Section 3.2.5 to improve DevSecOps.

During the data extraction for this research question, it was also identified that developers often lack the expertise to use and configure security tools efficiently and that developers should be appropriately trained before using these tools [P1, P4, P5, P12]. This study believes that the lack of knowledge can be mitigated by appropriate training and shared knowledge as recommended by [P4, P12, P13]. Also, improved documentation [P1] and best practices recommended by [P1, P7] can facilitate the use of security tools; see Section 4.2.2 for these recommendations.

Because of the limited extracted data, the result from this research question is believed to have insufficient external validity. The opinion of our study is that the research question should be further investigated using other research methodologies, e.g., surveys or interviews with practitioners engaged in DevSecOps projects.

6 Conclusions

The main objective of this study is to contribute to the existing research on the challenges associated with the selection and usage of security tools in the context of DevSecOps. The research began with exploratory research of DevSecOps in peer-reviewed research databases. Through this initial phase of research, challenges regarding the selection and usage of security tools in the context of DevSecOps were identified. Furthermore, it was also observed that related work primarily focused on application security tools and not to the same extent on other security practices, e.g., infrastructure security.

Thus, prompted by these observations, a Systematic Literature Review (SLR) was undertaken to investigate these observations further. The SLR was conducted following the guidelines outlined by [13]. The guideline provided by [13] aims to improve the quality of SLRs by reducing bias and increasing the validity and reliability of the research. An essential part of the guideline is to formulate the research questions for the SLR. The three research questions are;

- RQ1: What categories and examples of security tools in DevSecOps have been reported in previous literature? In what SDLC phases are the security tools used?
- RQ2: What are the drawbacks and recommendations of the identified security tools in the context of DevSecOps?
- RQ3: What technical team should be responsible for using and configuring the identified security tools in DevSecOps?

To answer the above questions, research literature from four databases has been collected: ScienceDirect, Springer, ACM, and IEEE. After conducting the SLR, this study extracted and synthesised data from fourteen papers (P1-P14), presented in A.3. The following sections summarise the results and the discussion regarding these questions.

6.1 RQ1

This study has identified thirteen security tool categories and fifty-eight security tool examples used or recommended to be used in DevSecOps. The result has been categorised into seven Software Development Life Cycle (SDLC) phases and five security practices. The results indicate that the current research focuses on the security practice of *Application Security*. Specifically, the result shows that seven of thirteen security tool categories and thirty of fifty-eight security tool examples are classified in

this study as *Application Security*. Security tools categories and examples for *Infrastructure Security*, *Threat Modelling*, *Secret Management*, and *Security Information and Event Management* have also been identified but not to the same extent as *Application Security*. Concluded from the result this study is of the opinion that the emphasis on *Application Security* can lead to false security assurance in projects that adhere to DevSecOps. This conclusion is drawn from the potential risk that the other four previously mentioned security practices may not be prioritised.

Additionally, the identified security tools are believed to support *Shift Security to the Left*, *Continuous Security*, and *Defence in Depth*, as detailed in Sections 3.1 and 3.2.5. Based on the support, this research has the opinion that the identified security tools can facilitate the security tool selection for practitioners and could potentially be used as input to a security tool standard for DevSecOps. This conclusion is based on two assumptions: Firstly, security tools structured in common SDLC phases can facilitate the selection of security tools in DevSecOps, and secondly, practitioners prioritise the capabilities of *Shift Security to the Left* and *Continuous Security* in their evaluation of potential security tools for DevSecOps projects.

6.2 RQ2

This study has found twelve drawbacks and sixteen recommendations concerning security tools used in DevSecOps. Two of the more significant drawbacks identified in this study are automation capabilities and false positives (a false vulnerability reported from the tool) produced by security tools.

To solve the problem with false positives, it is recommended to use Interactive Application Security tools (IAST) instead of, e.g., Static Application Security Tools (SAST) and Dynamic Application Security Tools (DAST). It is also recommended that developers need proper training before using security tools that produce a lot of false positives.

To solve the lack of automation capabilities, there is no explicit recommendation but more of a general requirement that security tools should be automated when used in DevSecOps.

6.3 RQ3

The synthesised results for this research question have been limited. The result shows that development and security teams are the ones that most often use security tools. The

result also shows that developers do not always have the right skill set for using security tools and should be trained before configuring and using these types of tools.

Because the data collected for this research question was limited, the result is believed to have insufficient external validity. As such, the result should not be considered generalisable to software projects that adhere to DevSecOps. However, the answer to this question is assumed to facilitate the selection process of security tools with practitioners with a specific skill set. This research question should be further investigated, e.g. with surveys or interviews in real-world DevSecOps projects.

6.4 Future Research

A survey was planned as an exploratory methodology to probe our findings from the literature. The survey was intended to be conducted as a questionnaire using the survey designed by Creswell [36]. Questionnaires were the preferred choice because of their capability to gather data from many subjects [37]. The survey was to serve the following purposes.

- Identify the tools used in DevSecOps in current industrial DevSecOps projects.
- Who are those working with these tools within those DevSecOps projects?

The opinion of our research is that RQ3 should be further examined, as it is assumed that the answer to the question could facilitate the selection of security tools in projects that adhere to DevSecOps. The planned survey discussed in this section concerns what technical teams are working with specific security tools in DevSecOps projects and could be used to further examined RQ3. A possible approach could be to use the identified security tools from RQ1 in this study to ask practitioners with a specific skill set if they are responsible for any of those security tools.

6.5 Threat to Validity and Reliability

Besides investigating the research questions, the objective has been to conduct the research in an unbiased, valid, and reliable way. Discussions regarding potential sources of bias and threats to the validity and reliability objectives are as follows:

Internal validity issues mainly concern the implementation and conduct of the review protocol. The first concern is missing relevant literature during the search process. The choice to only use four databases to identify research papers makes the threat notable.

During the search process, a replicated search was conducted on Google Scholar using the search string defined in the review protocol, Table 2.1. During the replicated search, similar results were identified as compared to the results from the four databases used in this study. Given the outcome of the replicated search, it was decided to limit the search process to the four used databases as it was assumed that the result would be comparable. The assumption that the result is comparable may have led to missing papers but has been necessary due to time constraints for this study.

The second concern is that the result is based on selected research containing synthesised data from grey literature. The sources of the grey literature have been verified to be from trusted sources and have been peer-reviewed. To be even more sure of the quality of the selected papers, this study has also used an external conference rating system, GII-GRIN-SCIE, to verify the quality of the selected research papers, which assesses six of the selected papers' conferences of the fourteen selected papers used in this SLR. GII-GRIN-SCIE shows that those six conferences have high or very high-quality ratings. Because the field DevSecOps is relatively new and consists of a small set of academic research but is more discussed in grey literature and used in practice. The opinion in our study is that by indirectly using grey literature more research can be used and the results will be more valid.

The third and last concern regarding internal validity is all the manual steps taken when conducting the review protocol, e.g., selecting, extracting, and synthesising data. All these steps included manually processing a large set of research and information. As a result, human mistakes may have occurred. To mitigate manual mistakes, the two researchers peer-reviewed each other's work when conducting the SLR.

Construct validity issues. During this study, we have noticed that a significant amount of the terminology concerning DevSecOps is not always clear. And extracting data from different research using unclear terminology could lead to interpreting data in a way that was not intended. To mitigate incorrect interpretation of evidence, common DevSecOps and DevOps terminology in each selected paper have been analysed to see how those papers define those concepts. The concepts analysed are DevSecOps, DevOps, Continuous Security, Sift Security to the Left, and SDLC phases.

External validity depends on construct and internal validity, so all issues we have described regarding these validity issues also affect external validity. Another concern regarding external validity is the data collection of security tool examples. The result regarding those security tools examples is just a snapshot of security tools between 2015

and 2022. The snapshot of tools risks becoming obsolete fast due to the changes, e.g., some tools have become deprecated.

Reliability issues. The biggest threat to reliability is data extraction, some data may have been overlooked or misinterpreted throughout the data extraction process. If an independent researcher tries to reproduce this study, they may not extract the same data that has been collected, and the research result will not be identical. To mitigate the issue, data have been extracted independently through multiple sessions. The researchers have then also discussed the extracted result. With different opinions concerning the extracted data, the researchers have dived deeper into the issue to solve the problem. Only data that both researchers agreed to have met the quality criteria in A.1 and are relevant to the research questions were extracted.

Bias issues. The study focused on increasing internal validity to reduce biases. Because the previously discussed internal validity issues can lead to systematic internal errors or bias, this study can not confidently claim that the research is unbiased. However, the use of two researchers is believed to mitigate such systematic internal errors. Both researchers have done the same work independently in all stages of the review protocol. A prerequisite to moving forward in the research has been that both researchers must agree on a task to proceed with the review protocol.

A second concern regarding bias is that some authors are present in multiple selected research papers used for this study. If these authors have strong positive opinions concerning security tools in DevSecOps, the results of this study might be influenced. As a result, this study risks publication bias. Extracted data have been discussed and analysed for inconsistency between the researchers to detect evidence for such publication bias. As a result of this process, this study is of the opinion that there is no notable publication bias that invalidates the result.

References

[1] Z. Ahmed and S. C. Francis, "Integrating Security with DevSecOps: Techniques and Challenges", *Proc. International Conference on Digitization (ICD),* 2019, pp. 178-182, DOI: 10.1109/ICD47981.2019.9105789.

[2] R. W. Macarthy and J. M. Bass, "An Empirical Taxonomy of DevOps in Practice" *Proc. 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA),* 2020, pp. 221-228, DOI: 10.1109/SEAA51224.2020.00046.

[3] R. Mao et al., "Preliminary Findings of DevSecOps from Grey Literature" *IEEE 20th International Conference on Software Quality, Reliability and Security (QRS),* 2020, pp. 450-457, DOI: 10.1109/QRS51102.2020.00064.

[4] GitLab Inc. "2022 DevOps predictions: GitLab experts weigh in on AI, security, remote work, and more", About.gitlab.com, 2021. [Online]. Available: https://about.gitlab.com/blog/2021/12/06/devops-predictions-gitlab-experts-weigh-in-on-ai-security-remote-work-and-more/. [Accessed: 30- Maj- 2022].

[5] T. A. Limoncelli, S. R. Chalup, and C. J. Hogan, “DevOps Culture”, in “The Practice of Cloud System Administration: DevOps and SRE Practices for Web Services”, vol 2. Addison-Wealey Professional, 2014, ch. 8, pp 171-192.

[6] H. Myrbakken, and R. Colomo-Palacios, (2017). “DevSecOps: A Multivocal Literature Review.” In Mas, A., Mesquida, A., O'Connor, R., Rout, T., Dorling, A. (eds) Software Process Improvement and Capability Determination. SPICE 2017. Communications in Computer and Information Science, vol 770. Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-67383-7_2.

[7] M. Sánchez-Gordón, and R. Colomo-Palacios. 2020. “Security as Culture: A Systematic Literature Review of DevSecOps”. *Proc. of the IEEE/ACM 42nd International Conference on Software Engineering Workshops.* Association for Computing Machinery, New York, NY, USA, 266–269. DOI: https://doi.org/10.1145/3387940.3392233**.**

[8] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen. 2022. “Challenges and solutions when adopting DevSecOps: A systematic review”**,** *Information and Software Technology,* 2022, vol 147. DOI: https://doi.org/10.1016/j.infsof.2021.106700**.**

[9] R. N. Rajapakse, M. Zahedi, and M. A. Babar. 2021. “An Empirical Analysis of Practitioners' Perspectives on Security Tool Integration into DevOps”. *Proc. 15th ACM / IEEE Inter. Sym*, ESEM. ACM. DOI: https://doi.org/10.1145/3475716.3475776.

[10] V. Mohan, and L. B. Othmane, "SecDevOps: Is It a Marketing Buzzword? - Mapping Research on Security in DevOps", in *11th Int. Conf.,* ARES, 2016, pp. 542-547, doi: 10.1109/ARES.2016.92.

[11] N. Forsgren, PhD. J. Humble, and G. Kim, “Accelerate The Science of Lean Software and DevOps: Building and Scaling High Perming Technology Organisations”. Portland, OR, USA: IT Revolution Press, 2018.

[12] “How much does a data breach cost in 2022”, IBM Corporation, Ponemon Institute, IBM Security, Armonk, NY, USA, Rep. 2022 [Online]. Available: https://www.ibm.com/security/data-breach. [Accessed: 12-August-2022].

[13] B. Kitchenham. “Procedures for Performing Systematic Reviews", Keel University, Keele, Eng, Rep. TR/SE-0401, 2004. [Online]. Available: https://www.inf.ufsc.br/~aldo.vw/kitchenham.pdf.[Accessed: 10-May-2022].

[14] X. Zhou, Y. Jin, H. Zhang, S. Li and X. Huang, "A Map of Threats to Validity of Systematic Literature Reviews in Software Engineering," 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), 2016, pp. 153-160, DOI: 10.1109/APSEC.2016.031.

[15] The GII-GRIN-SCIE (GGS) Conference Rating - About the GGS Rating 2021, *Scie.lcc.uma.es*, 2022. [Online]. Available: https://scie.lcc.uma.es:8443/gii-grin-scie-rating/conferenceRating.jsf. [Accessed: 25-July- 2022].

[16] W. Stallings, “Fundamental Security Design Principles”, in “Network Security Essentials: Application and Standards”, 6th ed, Global ed, Pearson 2017, sec 1.6, pp. 32-36.

[17] What Is Information Security?, [Online]. Available: https://www.cisco.com/c/en/us/products/security/what-is-information-security-infosec.html. [Accessed: 29-July-2022].

[18] W. Stallings, ”Computer Security Concepts”, in "Network Security Essentials: Application and Standards”, 6th ed, Global ed, Pearson 2017, sec 1.1, pp. 20-24.

[19] W. Stallings, “Attack Surfaces and Attack Trees”, in "Network Security Essentials: Application and Standards”, 6th ed, Global ed, Pearson 2017, sec 1.7, pp. 36-39.

[20] W. Stallings, “Cloud Security as a Service”, in "Network Security Essentials: Application and Standards”, 6th ed, Global ed, Pearson 2017, sec 5.7, pp. 182-185.

[21] What is network security?, [Online]. Available: https://www.vmware.com/topics/glossary/content/network-security.html. [Accessed: 10-August-2022].

[22] What is application security? [Online]. Available: https://www.vmware.com/topics/glossary/content/application-security.html. [Accessed: 10-October-2022].

[23] OWASP Top 10, 2021 [Online], Available: https://owasp.org/Top10/ . [Accessed: 10-August-2022].

[24] Kief Morris, “Infrastructure as Code - Dynamic Systems for the Cloud Age”, O'Reilly Media, Inc. 2020, pp. 102-104.

[25] M.J. Thomas, et al. "Development and Evolution of Agile.", 2020 [Online]. Available: http://www.iiisci.org/journal/PDV/sci/pdfs/SA349YP20.pdf. [Accessed: 10-june-2022].

[26] K. Zettler, “DevSecOps Tools: The DevSecOps tools that secure DevOps workflows”, [Online] Available: https://www.atlassian.com/devops/devops-tools/devsecops-tools. [Accessed: 10-May-2022].

[27] H. Dhaduk, "DevOps Lifecycle: 7 Phases Explained in Detail with Examples", *Simform - Product Engineering Company*, 2022. [Online]. Available: https://www.simform.com/blog/devops-lifecycle/. [Accessed: 31- Jul- 2022]

[28] J. Humble, D. Farley. “Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation”, O'Reilly Media, Inc. 2010.

[29] “Continuous Delivery vs Continuous Deployment”, 2010 [Online]. Available: https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/. [Accessed: 20-Oct-2022].

[30] GitLab Inc., “Secret Detection”, [Online]. Available: https://docs.gitlab.com/ee/user/application_security/secret_detection/. [Accessed: 9-Oct-2022].

[31] T. Scanlon, “10 Types of Application Security Testing Tools: When and How to Use Them”, 2018 [Online]. Available: https://insights.sei.cmu.edu/blog/10-types-of-application-security-testing-tools-when-and-how-to-use-them/. [Accessed: 2-June-2022].

[32] “Web Application Firewall” [Online]. Available: https://owasp.org/www-community/Web_Application_Firewall. [Accessed: 9-Oct-2022].

[33] “Infrastructure as Code Security Cheatsheet” [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Infrastructure_as_Code_Security_Cheat_Sheet.html. [Accessed: 9-Oct-2022].

[34] “Container Scanning”, [Online] ”https://docs.gitlab.com/ee/user/application_security/container_scanning/. [Accessed: 10-Oct-2022]

[35] “Intrusion Detection”, 2022 [Online]. Available: https://owasp.org/www-community/controls/Intrusion_Detection. [Accessed: 10-Oct-2022].

[36] J. W. Creswell, “Research Design - Qualitative, Quantitative and Mixed Method Approaches, fourth edition”, SAGE Publications, Inc. 2014.

[37] C.W. Dawson, “Projects on computing and information systems: a student's guide,” Addison Wesley Publishing Company, 2005.

Appendix

A.1 Quality criteria checklist

1. Is the main objective or research question, or both stated?
2. Is the research methodology stated?
3. Is the collected data, result, and conclusion valid?
4. Is the selected research paper unbiased?
5. Does the selected research paper have internal validity?
6. Have the authors analysed or discussed the result or both?
7. Have the authors answered the stated research questions?
8. Does the selected research paper have external validity?

A.2 Extracted Data Items from Selected Papers

Table A2.1: Data relevant to research metadata.

Data item	Description	Purpose
Author	Who is the author(s) of the research?	Reliability
Title	What is the title of the research?	Reliability
Publisher	Who is the publisher of the research?	Reliability
Year	When was the research published?	Reliability
DOI	What is the Digital Object Identifier of the study?	Reliability
Database	What research database?	Reliability
Date	When was the research selected?	Reliability

Table A2.2: Data relevant to research methodology, type, objective and terminology.

Data item	Description	Purpose
Research methodology	What is the research methodology of the study?	Internal/External Validity
Research type	Qualitative, quantitative, or mixed? Peer-reviewed or grey literature?	Internal/External Validity
Data source	Grey literature, Academic literature, Case study, Interviews, etc.	Internal/External Validity
DevOps	How do the studies define DevOps?	Construct Validity
DevSecOps	How do the studies define DevSecOps?	Construct Validity
Shift security to the left	How do the studies define Shift security to the left?	Construct Validity
Continuous Security	How do the studies define Continuous Security?	Construct Validity
SDLC terminology	What terminology is used to describe the SDLC phases?	Construct Validity
Main Objective	What is the study's primary objective	Internal/External Validity
Threats to validity	What are the threats to validity?	Validity

Table A1.3: Data relevant to research questions RQ1, RQ2, and RQ3.

Data item	Description	Purpose
Security tool category[1]	What are the categories of the tools reported from previous research?	RQ1
Security tool examples[1]	Are there any examples of tools for those categories?	RQ1
SDLC phase tool	Where in the SDLC are the tools utilised?	RQ1
Advantages and recommendations	What are the advantages or recommendations for using the tool in DevSecOps?	RQ2
Disadvantages and drawbacks	What are the disadvantages or challenges of using the tool in DevSecOps?	RQ2
Technical expertise	What technical expertise is associated with managing the tool in the context of DevSecOps?	RQ3

[1] Security tools that are not in the scope of this research definition of security tools, see Section 1.5.1, will not be extracted.

A.3 Selected Papers

The following selected papers are the result of conducting the review protocol described in Section 2.2.

[P1] R. N. Rajapakse, M. Zahedi, M. Ali Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review"**,** *Information and Software Technology*, 2022*,* vol 147.DOI: https://doi.org/10.1016/j.infsof.2021.106700

[P2] M. A. Akbar, K. Smolander, S. Mahmood, and A. Alsanad, "Toward successful DevSecOps in software development organisations: A decision-making framework", *Information and Software Technology*, 2022, vol 147. DOI: https://doi.org/10.1016/j.infsof.2022.106894

[P3] R. Kumar, and R. Goyal, "Modelling continuous security: A conceptual model for automated DevSecOps using open-source software over the cloud (ADOC)", Computers & Security, 2020, vol 97. DOI: https://doi.org/10.1016/j.cose.2020.101967.

[P4] M. Sánchez-Gordón, and R. Colomo-Palacios, "Security as Culture: A Systematic Literature Review of DevSecOps". Proc. IEEE/ACM 42nd Int. Conf. on Software Engineering Workshops. Association for Computing Machinery, New York, NY, USA, 266–269, 2020, DOI: https://doi.org/10.1145/3387940.3392233

[P5] R. N. Rajapakse, M. Zahedi, and M. A. Babar, "An Empirical Analysis of Practitioners' Perspectives on Security Tool Integration into DevOps". Proc. 15th ACM/IEEE Int. Symp. Empirical Software Engineering and Measurement (ESEM), Association for Computing Machinery, 2021, DOI: https://doi.org/10.1145/3475716.3475776

[P6] F. Angermeier, M. Voggenreiter, F.Moyón, and D. Mendez, "Enterprise-driven open source software: a case study on security automation". *Proc. 43rd Int. Conf. on Software Engineering*: Software Engineering in Practice. IEEE Press, 2021, 278–287, DOI: https://doi.org/10.1109/ICSE-SEIP52600.2021.00037

[P7] M. Efendi, T. Raharjo, and A. Suhanto, "DevSecOps Approach in Software Development Case Study: Public Company Logistic Agency", *Int. Conf. Informatics, Multimedia, Cyber and Information System* (ICIMCIS), 2021, pp. 96-101, DOI: 10.1109/ICIMCIS53775.2021.9699316.

[P8] 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", *IEEE 24th Int. Enterprise Distributed Obj. Comp. Conf. (EDOC)*, 2020, pp. 145-154, DOI: 10.1109/EDOC49727.2020.00026.

[P9] N. Tomas, J. Li, and H. Huang, "An Empirical Study on Culture, Automation, Measurement, and Sharing of DevSecOps", 2019 *Int. Conf. Cyber Secu. Protection of Digital Services (Cyber Security)*, 2019, pp. 1-8, DOI: 10.1109/CyberSecPODS.2019.8884935.

[P10] V. Mohan, and L. B. Othmane, "SecDevOps: Is It a Marketing Buzzword? - Mapping Research on Security in DevOps", *11th Int. Conf. Availability, Reliability, and Security (ARES)*, 2016, pp. 542-547, DOI: 10.1109/ARES.2016.92.

[P11] R. Mao et al., "Preliminary Findings of DevSecOps from Grey Literature", *IEEE 20th Int. Conf. on Software Quality, Reliability, and Security (QRS)*, 2020, pp. 450-457, DOI: 10.1109/QRS51102.2020.00064.

[P12] H. Myrbakken, and R. Colomo-Palacios, “DevSecOps: A Multivocal Literature Review.” *In Mas, A., Mesquida, A., O'Connor, R., Rout, T., Dorling, A. (eds) Software Process Improvement and Capability Determination. SPICE 2017. Communications in Computer and Information Science,* vol 770. Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-67383-7_2.

[P13] A. A. U. Rahman, and L. Williams. 2016, “Software security in DevOps: synthesising practitioners' perceptions and practices”. *In Proceedings of the International Workshop on Continuous Software Evolution and Delivery (CSED '16). Association for Computing Machinery,* New York, NY, USA, 70–76. DOI: https://doi.org/10.1145/2896941.2896946

[P14] Rafi, Saima, Wu Yu, and Muhammad Azeem Akbar. "Towards a hypothetical framework to secure DevOps adoption: Grounded theory approach." In *Proceedings of the Evaluation and Assessment in Software Engineering*, pp. 457-462. 2020.
DOI: 10.1145/3383219.3383285