Anna Koskinen

DEVSECOPS: BUILDING SECURITY INTO THE CORE OF DEVOPS

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2020

ABSTRACT

Koskinen, Anna
DevSecOps: Building Security Into the Core of DevOps
Jyväskylä: University of Jyväskylä, 2019, 64 pp.
Information Systems, Master's Thesis
Supervisor: Costin, Andrei

The constantly growing rate of sophisticated, high-speed cyber-attacks brings new challenges to the people working in cyber defense. How can security prevent vulnerabilities, detect attacks in real time and respond to security incidents effectively? At the same time further down the development pipeline, another kind of time pressure is felt by software developers: business needs are constantly pressing for faster software release cycles. How can security be properly addressed in the ever-increasing pace of modern software development? In the last decade, DevOps has grown steadily as a software development method and its ability to deploy products constantly has made organizations deploy applications up to hundreds of times per day. In the rapid-fire development life cycles, the question becomes, how can security be ensured at the same pace? This Thesis used a Systematic Literature Review to discover how security activities can be added into the core of DevOps development process in order to evolve the development methodology into DevSecOps, i.e., a development methodology that encompasses not only Development (Dev) and Operations (Ops) but also Security (Sec). The research looked at 18 different articles to understand how security activities can be used in DevOps processes as well as what challenges DevOps brings to security. The Building Security In Maturity Model (BSIMM) was used as a framework to chart the activities described in the academic research. The research literature was also reviewed through the four principles of DevOps: Culture, Automation, Measurement and Sharing (CAMS). As a result, it was found that the available research focuses heavily on securing the technologies frequently used in DevOps infrastructures (e.g., containers, development pipelines and cloud infrastructures). Looking at the challenges of security in DevOps, the research found the biggest challenges to be securing the deployment pipeline, balancing security with fast deliveries, as well as combating insider threat. The research also concluded that there are still many conflicting views on what DevOps is, which is shown by the DevOps principles not being reflected in the current research. The research gives an overview of the current state of research of security activities in DevOps, paves the way for DevSecOps style software development and brings forth research gaps for further researchers to explore.

Keywords: DevOps, DevSecOps, secure software engineering, BSIMM, SDLC

TIIVISTELMÄ

Koskinen, Anna
DevSecOps: Turvallista DevOpsia rakentamassa
Jyväskylä: Jyväskylän yliopisto, 2019, 64 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Costin, Andrei

Hienostuneiden ja nopeatahtisten kyberhyökkäysten jatkuvasti lisääntyvä määrä aiheuttaa haasteita tietoturvallisuuden parissa työskenteleville. Miten uudistuvassa toimintaympäristössä pystytään ehkäisemään haavoittuvuuksia, havaitsemaan hyökkäyksiä ja reagoimaan tietoturvaongelmiin tehokkaasti? Samaan aikaan toisenlainen aikapaine vaivaa ohjelmistojen kehittäjiä: liiketoimintavaatimusten vuoksi ohjelmistoja halutaan julkaista yhä nopeammalla tahdilla. Miten tietoturva varmistetaan kiivaassa kehityssyklissä? DevOps on viime vuosina saavuttanut vankan aseman ohjelmistojen kehittämismetodina ja sen mahdollistama jatkuva integrointi saa yritykset työntämään uusia järjestelmäversioita tuotantoon jopa satoja kertoja päivässä. Nopeassa kehityssyklissä tärkeäksi kysymykseksi nousee, miten voidaan varmistaa ohjelmistojen tietoturva yhtä nopealla tahdilla. Tässä työssä tarkasteltiin systemaattisen kirjallisuuskatsauksen kautta, miten tietoturvaa parantavia aktiviteetteja voidaan lisätä DevOps-kehittämisprosesseihin, jotta kehittämismenetelmässä päästäisiin todelliseen DevSecOps-malliin – malliin, johon kehittämisen (Dev) ja ylläpidon (Ops) olisi integroitu myös tietoturva (Sec). Työssä tutkittiin 18 eri akateemisen artikkelin näkemystä siitä, mitä tietoturva-aktiviteetteja DevOps-prosessissa voidaan käyttää sekä mitä haasteita DevOps asettaa tietoturvalle. Viitekehyksenä työssä käytettiin BSIMM-mallia (Building Security In Maturity Model), jonka avulla kartoitettiin turvallisuusaktiviteettien esiintymistä tutkimuksessa. Tutkimuskirjallisuutta tarkasteltiin myös DevOpsin neljän periaatteen (kulttuurin, automaation, mittaamisen ja jakamisen) kautta. Tuloksena huomattiin, että nykytutkimus keskittyy pitkälti DevOps-infrastuktuurissa käytettyjen teknologioiden (esim. konttitekniikat, kehitysputki ja pilvi-infrastruktuuri) turvaamiseen. DevOpsin turvallisuushaasteista tutkimus havaitsi suurimmiksi kehitysympäristön turvaamisen, turvallisuuden ja nopeiden toimitusten tasapainottamisen sekä niin sanotun sisäisen uhan (eli työntekijäväärinkäytösten) lisääntymisen mahdollisuuden. Lisäksi tutkimus havaitsi, että tutkijoiden kesken vallitsee edelleen erilaisia näkemyksiä siitä, mitä DevOps on, sillä DevOpsin perusperiaatteet ilmenevät heikosti nykytutkimuksesta. Tutkimus antaa yleiskuvan turvallisen DevOps-kehittämisen nykytutkimuksesta, edesauttaa DevSecOps-tyylistä kehittämistä sekä tuo esiin tutkimusaukkoja tulevien tutkijoiden tutkittaviksi.

Asiasanat: DevOps, DevSecOps, tietoturvallinen kehittäminen, BSIMM, ohjelmistokehityksen elinkaari

FIGURES

TABLES

TABLE OF CONTENTS

1 INTRODUCTION

The DevOps development model has taken the software development world by storm. With estimated adoption rates ranging from over 50 % of organizations (Stroud 2017, Mansfield-Devine 2018) to 88 % of organizations (Ur Rahman and Williams 2016), the popularity of DevOps is for certain. But what exactly is DevOps? According to one definition, DevOps is a development methodology which emphasizes communication between software developers and operations and aims at fast delivery times through automated delivery pipelines (Jabbari et al. 2016). However, the definitions vary and not everyone who says they are "doing DevOps" agrees on the term's meaning (Mansfield-Devine 2018). This can cause confusion and lead to failed adoption initiatives, if no clear definition has been made of what is being talked about when talking about DevOps.

The reasons for the growth of the DevOps paradigm are various. The desire for more rapid software development cycles to satisfy customer needs has led to automated delivery pipelines, where code written by a developer can be pushed to production instantaneously. As DevOps maximizes the use of automation tools in software development, communication becomes faster and more effective, because it happens automatically through systems (Cois et al 2014). Pushing code into production frequently has also led to a closer relationship between the development team (Dev) and the operations team (Ops), and from this close collaboration, the development methodology has derived its name. More than just a development method, DevOps is also very much a cultural issue within the organization implementing it. Automation technologies also change the work of IT staff, as manual tasks become automated and automated tasks require new expertise. The increased automation of software deliveries leads to the so-called "developer self-service", where a developer is single-handedly able to push his code into the production environment. This autonomy is considered to contribute to higher work satisfaction. Both Amazon and Netflix state that increased autonomy makes developers happier (Khan 2018, Hahn 2016).

When we talk about increasing the speed of deliveries, a relevant question to ask is how does this speed affect software quality in all its aspects. Some scholars feel that the aim for great quality is part of DevOps' essence, whereas others talk of the speed being the only name of the game. Some feel that developers gain a "sense of pride and ownership" through the new development regime (Mackey 2018), whereas others (e.g., Ahmanavand et al. 2018, Ullah et al. 2017) warn of the increased insider access bringing a whole new can of security worms to the development environment and beyond. All we can say is, when it comes to software security, DevOps definitely antes up the game.

In this Master's Thesis, I have set myself the task of understanding the current state of research on how security practices fit into the DevOps development process. The task I have set myself with this thesis is not simple. It has been said that "There is little evidence of how to implement security practices in the software industry, much less in an agile context" (Jaatun et al. 2017). If the software industry in general and agile practices in particular lack guidance on security practices, the case could be perceived to be even more severe with DevOps, as it is a significantly younger practice. Blog posts and industry guidance on secure DevOps practices exist, but as many writers have the agenda of recommending their own products or services, the academical non-biased view would be beneficial to understand the situation better. Many academic pieces echo the need for more understanding of how security fits into DevOps (e.g., Mansfield-Devine 2018, Tuma et al. 2018) and have recorded the interest of established business players such as IBM (Mohan et al. 2018) of getting a better understanding of how security can be integrated into DevOps.

To understand the current state of research, I have conducted a Systematic Literature Review that tracks the available research on the subject and analyses the relevant articles. The goal of the Thesis is to synthesize how the academic world sees security practices in relation to DevOps development. It suffices to say that DevOps' popularity is not yet reflected in the academic research in the field of software security – only a limited number of DevOps works concerning security are available. In this Systematic Literature Review, 18 articles comprise the primary studies upon which the analysis is based. As a tool for analysis I am using two frameworks: the Building Security In Maturity Model (BSIMM) (McGraw et al. 2019) and the CAMS-principles (Willis 2010). The BSIMM maturity model is a framework of prevalent security practices and activities, which have been collected from the software industry. Through surveying which security activities practitioners actually perform, BSIMM unifies the activities into a framework and grades the maturity of each activity. In the BSIMM model, security activities are actions "undertaken for the purpose of building secure software". The different actions are organized into larger groups of security practices, which in turn belong to security domains. (McGraw et al. 2019.) As such, the BSIMM model offers an organized framework of security domains, practices and activities. The framework is explained in detail in Chapter 2.4.2. The second framework used in this research are the four principles of DevOps which go under the acronym CAMS. The four principles are Culture, Automation, Measurement and Sharing. The four principles are introduced in more detail in Chapter 2.3.

In my research, I am answering three research questions:

- RQ1: What are the challenges of security in DevOps as reported by the authors of primary studies?
- RQ2: Which security activities are associated with DevOps in the literature?
- RQ3: How are the CAMS (culture, automation, measurement and sharing) principles reflected in secure DevOps research?

The first research question investigates which challenges the authors of primary studies link to security in DevOps. By looking at the challenges, I am hoping to capture DevOps' uniqueness as a development method and to understand how that uniqueness translates to the security activities DevOps needs in particular. The second research question charts which security activities have been thus far linked with DevOps. Through analyzing the results, an understanding is gained of where research efforts have been concentrated and where there might be research gaps. Through a synthesis of the recommended security activities, an outline of recommendable security activities for DevOps development is attained. The third research question goes through the primary studies and looks at how the studies speak of DevOps' CAMS principles. By analyzing different authors' understanding of culture, automation, measurement and sharing, it is possible to observe whether the authors see DevOps identically or whether different interpretations surface. Through the three research questions I get an understanding of the current state of research, how DevOps is understood by the security researchers and where future research efforts should be concentrated.

My results show that 42 % of the security activities mentioned in DevOps research have a focus of securing the technological infrastructure of DevOps – mainly container and cloud infrastructure as well as the deployment pipeline itself. My research also confirms the view of Jabbari et al. (2016), who found that in the research community, DevOps is understood in a number of ways that are possibly different or even disjoint. In other words, some writers of the reviewed articles see DevOps as a cultural movement that requires communication and collaboration, where as some perceive it to be a means to end of achieving speedy deliveries through automated deployment technologies. Suffice to say, for any organization wanting to implement DevOps, having a clear definition of what their interpretation of DevOps is, would be the first thing to establish.

Previous work on secure DevOps practices is limited. Some secondary studies from related fields have been conducted. To my knowledge, there has not been a study that looks at the state of research of secure DevOps practices from the same viewpoint as I will do in this work; that is, from trying to map the security activities that have been applied to DevOps in academic literature, the perceived challenges of security in DevOps and how the DevOps principles are reflected in the studies. Previous systematic literary reviews related to my research topic are listed in Table 1.

TABLE 1 Previous literary reviews related to the research topic

Reference	Year	Title	Focus	No. of reviewed works
Jabbari et al.	2016	What is DevOps? A Systematic Mapping Study on Definitions and Practices	The definitions of DevOps and how does DevOps differ from other development methods.	49
Mohammed et al.	2017	Exploring software security approaches in software development life cycle: A systematic mapping study	A mapping study on the use of different security approaches in software development life cycle.	118
Mohan & ben Othmane	2016	SecDevOps – is it a marketing buzzword	A mapping study of the state of research in secure DevOps: what it is and which security practices and tools are used.	8
Myrbakken & Colomo-Palacios	2017	DevSecOps: A Multivocal Literary Review	DevSecOps definitions, benefits and challenges from (mostly) non-academic sources.	52

Jabbari et al. (2016) looked at available DevOps research and how DevOps was defined in them. What they found was that the definitions of DevOps varied greatly depending on the author. Security aspects of DevOps were not addressed in their research.

Mohammed et al. (2017) did a mapping study on different security approaches in the software and how they fit into the Software Development Life Cycle (SDLC). They divided the development life cycle into four parts and found that in the reviewed studies, 19 % of security practices were conducted in the requirements phase, 29 % in the design phase and 41 % in the coding phase. Finally, 11 % of security practices spanned the whole life cycle. The most popular approaches to security during development were dynamic analysis and static analysis. Mohammed et al. (2017) did not pay attention to the SDLC changing due to new development methods and their work did not contain any mentions of DevOps.

Mohan and ben Othmane (2016) conducted a mapping study of secure DevOps research. During the time of their study, only five pieces of applicable academic research where found. They used those five studies, along with three industry pieces, to map the state of secure DevOps and to understand the phenomenon better, and – in their words – to understand whether SecDevOps or DevSecOps (the merging of Development, Security and Operations) were just "marketing buzzwords". Their study concluded that the phenomenon was not just marketing, but rather an important subject for further studies, as the security aspect of DevOps is a major concern for organizations considering the implementation of DevOps. My research deepens the knowledge of their work, as at the time of my writing this more academic research on secure DevOps is now available, which enables the creation of this more in-depth Systematic Literature Review.

Myrbakken and Colomo-Palacios did a multivocal literary review on the definitions, benefits and challenges of DevSecOps in 2017. Their work focused on analyzing mostly the industry’s voice, as 50 of the 52 reviewed texts were Internet artifacts attained through the Google search engine. The focus of their work was on gaining an understanding of what the phenomenon of DevSecOps is. Myrbakken and Colomo-Palacios used the four principles of DevOps (CAMS) to gain a deeper understanding of DevSecOps. My research continues their work by looking at later academic articles to study the subject of security practices in DevOps, though not limiting myself to works that explicitly state themselves as belonging to the field of “DevSecOps”, as they did. The lack of available literary reviews on secure practices in DevOps indicates that there is a research gap for more work on the subject.

The rest of this Thesis is organized as follows. In Chapter 2 I will offer a theoretical background to my work. I explain software development processes, the four principles of DevOps and how security practices can be included in the software development life cycle. Chapter 3 presents the research methodology: the systematic literary review as a research method and how it was implemented in this research. Chapter 4 offers results of the reviewed works and Chapter 5 offers a discussion on the subject. Chapter 6 concludes this Thesis.

2 THEORETICAL BACKGROUND

2.1 Software development and the development life cycle

Before software lands on our laptops or mobile devices, it undergoes a rigorous and complex process to arrive there. Once an idea for new software is attained, multiple phases of design and development are needed before an actual product is available for use. As modern software are complex, abstract products with multiple functions, preferably arriving to customers with quality and security in check, the road from an idea to a product can be rather long. To make the development of software rigorous and well-organized, a development method is used. The path of a software idea coming to fruition is generally called the Software Development Life Cycle (SDLC). In the traditional waterfall development model, different phases of the software development follow one another, as illustrated in Figure 1 (adapted from Cois et al. 2014).

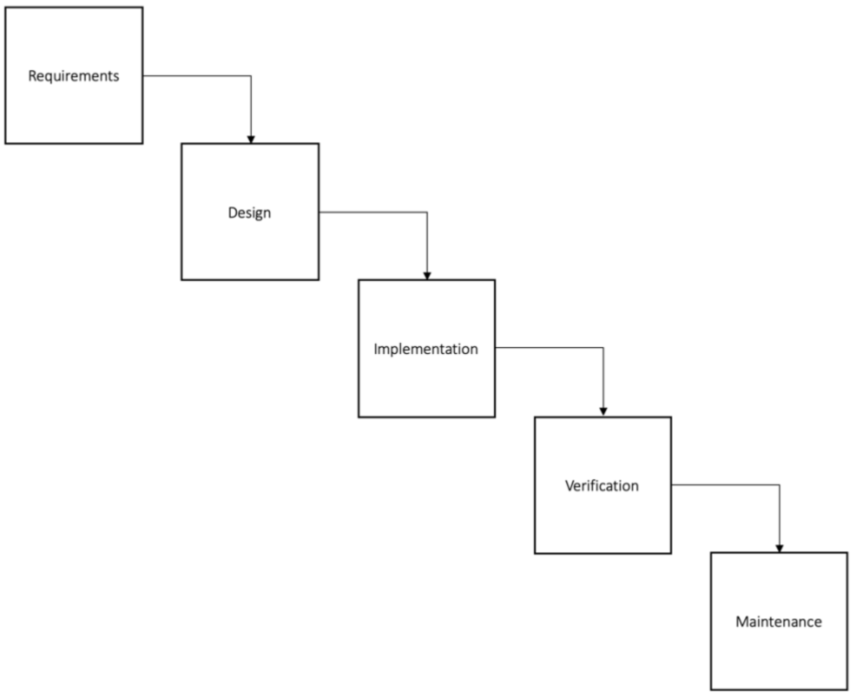

FIGURE 1 The waterfall development model (Cois et al. 2014).

In the waterfall model, the development of requirements comes first, after which comes the software design, implementation (coding), verification (testing) and so on. Each phase requires the completion of the previous phase. The waterfall model has been critiqued (e.g., Cois et al. 2014) for its challenges in adapting to changes. As software projects often take long to finish, the time span from the requirements phase (recording what the customer wants) to the deployment phase (delivering the product to customer) can be months or even years. This makes it difficult to accommodate evolving customer needs. In long projects, the customer needs might change during the product's development, which – in the worst-case scenario – ends in delivering an already out-of-date piece of software to the customer's hands. To combat this, software development has moved into a more agile development model. In agile development, the focus is on iterative and incremental development. This development style (or styles, as there are multiple variations) derives from the Agile Manifesto (2001), a manifesto created by frustrated industry insiders who yearned for development that would be, as the chosen name implies, agile and dynamic. The Agile way has gained popularity in its almost 20 years, replacing the waterfall development style almost entirely. Agile offers dynamic adaptability, as software is built in small increments that enable accommodating new business needs as they surface.

2.2 DevOps as a development method

The development methodology that is the subject of this Thesis, DevOps, builds on Agile's legacy. Jabbari et al. (2016) noticed that the academia does not agree on DevOps' and Agile's relation. Some consider DevOps to be an extension of Agile, some that Agile methods enable DevOps practices, yet others that DevOps and Agile answer to the need of different stakeholders as DevOps enhances the relationship between Dev and Ops, whilst Agile does the same to business owners and Dev (Jabbari et al. 2016). The same incongruence is apparent from looking at what researchers state DevOps' goal to be. The goal of DevOps has been said to be to get the development and operations teams to work together for the success of the software product (Cois et al 2014), to improve productivity and waste less time (Khan 2018), to satisfy dynamic business needs (Michener & Clager 2016), to deliver products to customers faster (Tamburri et al. 2019) and to build more dependable products (Mansfield-Devine 2018). As can be seen from the varied definitions of DevOps' goals, there is not a single DevOps paradigm that the research community agrees on. Yet, DevOps is not a spring chicken anymore, as it has been around for over ten years, with, for instance, Amazon having adopted it in 2009 (Khan 2018).

How DevOps differs from Agile is that it increases the collaboration between Dev and Ops. Other important facets of the method are the use of the deployment pipeline and increased automation, which provide the developers with the autonomy of publishing their code into production. Operations, on the other hand, monitor the product and use the enhanced collaboration with Dev as a way to fix any issues quickly. Work is not done in isolated silos. Therefore, the DevOps life cycle is often depicted as a continuous loop running from development to operations and back again, as depicted by Figure 2.

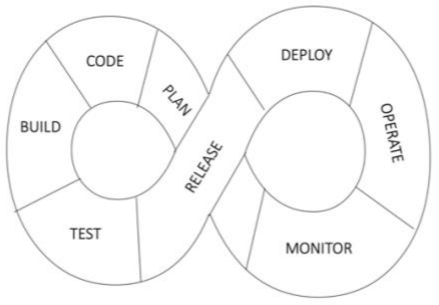

FIGURE 2 The DevOps development life cycle (Loughman 2019)

The DevOps development life cycle loop shows how the planning of software leads to coding, building, testing and releasing software, which turns to operating and monitoring the software, and finally back to planning new functionality. The loop structure indicates that the collaboration between Dev and Ops continues even after deployment, and that the information gained from operations and monitoring is used by the Dev in developing the product further.

Though DevOps has gained popularity as a development method, it can't be said that DevOps has been implemented with proper maturity everywhere. Partly this might be due to the ambivalence of what DevOps is. Some say they are "doing DevOps" when they have dabbled with continuous integration (Mansfield-Devine 2018). According to the maturity levels set for DevOps organizations by Puppet (2019), DevOps can be seen to be on a mature level when the four principles of DevOps (culture, automation, measurement and sharing) are in place and there exists a culture of collaboration and constant dialogue between Developers and Operations. In this research, DevOps is understood as a development methodology that is underpinned by the four DevOps principles, which are presented next.

2.3 The four principles of DevOps

DevOps, in its simplicity, can be said to be "about aligning the incentives of everybody involved in delivering software" (Humble & Molesky 2011, 6). However, not everyone talking about DevOps perceives it as such an ideological manner. Jabbari et al.'s (2016) mapping study examined 49 research pieces and looked at their definition of DevOps. The definitions they found varied greatly, with different authors understanding different principles to be at the core of the development method. Some see DevOps' focus to be highly automated deliveries, while others perceive the main idea to be increased communication and collaboration between stakeholders. When Jabbari et al. (2016) merged their findings into one, they derived the following definition:

> DevOps is a development methodology aimed at bridging the gap between Development (Dev) and Operations, emphasizing communication and collaboration,

> continuous integration, quality assurance and delivery with automated deployment utilizing a set of development practices. (Jabbari et al. 2016, p. 6)

What can be concluded, is that DevOps is not a siloed practice. It aims at delivering business value through speed, agility, automation and increased communication. As these are goals most organizations would like to achieve, it is no wonder that up to 88 % of organizations (Ur Rahman and Williams 2016), have jumped on the DevOps bandwagon.

The DevOps mindset lies in its four principles, culture, automation, measurement and sharing, which have been spelled out with the acronym CAMS. These four CAMS principles were first introduced by DevOps pioneer John Willis (2010) in a blog post discussing the core values of DevOps development. Since then, they have been used as a theoretical framework in a number of academic (e.g., Myrbakken & Colomo-Pacios 2017, Humble & Molesky 2011) and industry (e.g., Puppet 2019) works. In a key industry report by Puppet *(*2019), the state of DevOps is based on the state of adoption of the CAMS principles in the surveyed companies. For example, an organization shows more maturity in its automation when there is a high level of "self-service" for developers, as opposed to the automation of only a select number of services. (Puppet 2019.) As the CAMS principles have previously been used in academic works, I chose them as a yardstick in my research to measure the primary studies' understanding of DevOps as an organizational phenomenon that changes culture, facilitates sharing and expects automation and measurement. Next, the four principles are presented in more detail.

2.3.1 Culture

In DevOps, the objective is to get the development and operations to work together for a common goal, i.e. for a product that works as well as possible. The first step towards DevOps is to break down silos (e.g., Khan 2018; Cois et al 2014): in DevOps, a product is not done by a team of developers in one place and then handed down to a team of operations somewhere else. In DevOps, the responsibility is shared (Khan 2018) and thus a defect or a problem also becomes a shared issue. This differs from the traditional software development model where development has focused on developing the software and once the software is done, the development team has figuratively "thrown it over the fence" (Humble & Molesky 2011) over to operations. Defects or bugs in the product which are discovered in production have not been problems of the developers. DevOps shifts this perspective: it considers it the developers' responsibility to aid operations in troubleshooting and solving problems (Humble & Molesky 2011). It's a model of co-operation and working towards a common goal.

The importance of culture should not be underestimated: Gartner (2019) estimates that 75 % of DevOps adoptions will fail because of organizations not being able to learn and change – which are very much cultural issues. In other words, if the organization only adopts the DevOps technologies and an automated pipeline, problems will surface due to the people not being sufficiently brought up to date with the culture change the technology demands. Also, Bass (2018) points out that adopting DevOps changes the organizational culture through the change in roles: e.g., if all tests are automated, the former testers need to adopt new roles in the organization. What starts as a mere adoption of faster deployment technology, can in fact become an organization-wide change in culture.

2.3.2 Automation

The main difference that sets DevOps apart from other practices is its focus on software release automation. This practice is done through release pipelines. For example, Amazon has done 50 million deploys within a single year (Khan 2018). As such, one of DevOps' characteristics is the deployment pipeline. In the pipeline, the build, testing and deployment have been automated (Humble & Molesky 2011). Humble and Molesky (2011, 7) define the deployment pipeline as "a single path to production for *all* changes to a given system, whether to code, infrastructure and environments, database schemas and reference data, or configuration." In its essence, the deployment pipeline enables software releases at the push of a button. It's the key to multiple daily releases. The pipeline also affects the organizational culture as it provides the developers with added autonomy and responsibility: the developers are able to push the code they have made into production at will. The pipeline can also facilitate instant feedback: if tests have been automated, the developers are able to see if their code passes the tests or not. This developer self-service can change the way work is done and also the satisfaction it provides. For example, Netflix (Hahn 2016) says its main goal in enabling continuous development is to provide developers with total freedom of creation, where they will not be hindered by the restraints of systems. In their case, the development infrastructures are created in a way that will allow the developers to maximize their productivity – and in the wake of it bring a beautiful customer experience (Hahn 2016). Figure 3 shows how developer self-service can, sometimes with just a single push of a button, pass many development phases in an automated deployment pipeline. When we consider that before the deployment pipeline, the developer was responsible only for the coding phase, we understand how much DevOps changes the actual work processes through automation.

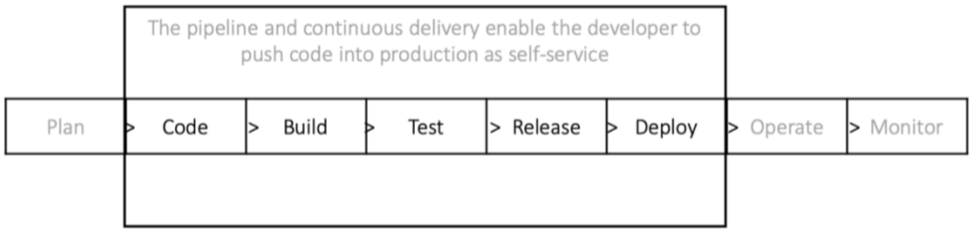
The pipeline and continuous delivery enable the developer to push code into production as self-service
Plan
Code
Build
Test
Release
Deploy
Operate
Monitor

FIGURE 3 The five phases of the developer self-service in the SDLC

2.3.3 Measurement

Monitoring and measuring are fundamentals in DevOps and they can partly be by-products of automation. In DevOps, opportunities for measurement are plentiful. Measuring provides opportunities for quick response, as well as for learning and growth. However, in a world where data is abundant, it is necessary to measure the *right* things. Puppet (2019) recommends that organizations choose key metrics which are aligned with business objectives. Also, as research shows that high-level management has very different views of how far along the DevOps adoption is and how well security has been incorporated into it, it is beneficial to provide metrics on the true state of how far along an organization is in the DevOps adoption process (Puppet 2019). Measuring should cover the development phases as well as use metrics from operational monitoring. Another important issue is to make sure that the organization not only measures but also uses the end-results wisely. For instance, monitoring results from operations should form a feedback loop to developers which enables the developers to improve the product.

2.3.4 Sharing

Whilst sharing can seem like the most lightweight principle of DevOps, it is the leg that really keeps the practice stable. Both DevOps adoption and DevOps performance depend on sharing both successes and failures (Puppet 2019). In an organization where failures are not shared, each team is left to stumble upon the same obstacles. Where successes are not shared, successful practices become secrets of the teams that have invented them. In order for the whole organization to benefit and grow, sharing must be a strong principle that is enforced throughout the whole organization. Furthermore, Puppet (2019) proposes that with the high availability of automation tools, it is relatively easy for any company to start on the DevOps journey but for genuine DevOps adoption (and the business benefits that follow), an organization must also implement a culture that facilitates sharing. Developers, operations and security should work together to make the product as good as possible. With a sharing mindset, problems are not dealt with in isolated silos – instead, there is a collaborative effort to solve the problems. The product's behaviour in production brings feedback to the developers and vice versa. For example, if a security issue is detected in development, the developers can collaborate with the information security team members to decide how the problem is best solved, e.g., can it be fixed with a firewall solution or does it need to be settled by a change in design (Mansfield-Devine 2018). With DevOps, "the aim isn't just continuous integration and deployment but also continuous communication, collaboration and notification" (Mansfield-Devine 2018), which all serve the common goal of producing and operating high-quality software. When it comes to security, sharing can be seen as a form of communication: when security teams openly discuss their findings and consult the developers in implementing more successful security controls, more progress is made. When developers and operations share with the security team their security concerns, disasters can be prevented.

2.4 Security practices in software development

Securing systems is nothing new. The question of system security was first being raised in the 1960's and according to cyber security veteran Donn B. Parker "the mid-seventies was when it all hit the fan" (Charles Babbage Institute 2003). The techniques used for securing systems go under various names but can be collectively called e.g. *software security practices* (Williams et al. 2018), *software security activities* (McGraw et al. 2019) or *information systems security design methods* (Baskerville 1993). The importance of system security is constantly growing as more of our lives and devices are connected to the web. Digital information is valuable and therefore susceptible to cybercrime. The global nature of the Internet makes it a playground for criminals looking for greener grass in the digital realm. Just like a pick-pocket will steal a wallet if it's not properly protected, so will the cybercriminals misuse an application if it is not secured. For this reason, security of software is a must.

There are several ways to approach software security. According to Mohammed et al. (2017), there are currently three main approaches:

1) security is addressed as something to be added on after the deployment of the software, by fixing found flaws,

2) security is seen as a feature/function of the operating environment (e.g., security is added to the network's outer perimeters via firewalls and intrusion detection systems) or

3) secure software engineering, where security is baked into the software life cycle (Mohammed et al. 2017).

Williams et al. (2018) have a different approach. They divide the security practices into three categories according to their proactiveness towards vulnerabilities: the practices can be preventive, detective or responsive. *Preventive* practices are used before deployment in order to prevent vulnerabilities. *Detective* practices aim at discovering vulnerabilities that have become exploited and *responsive* practices are used to mitigate exposed vulnerabilities. (Williams et al. 2018.) By picking and choosing the right practices from each category that suit the organization's needs and development practices, the organization can make its development and operations processes more secure. Figure 4 illustrates Williams et al.'s view on the three types of security practices.

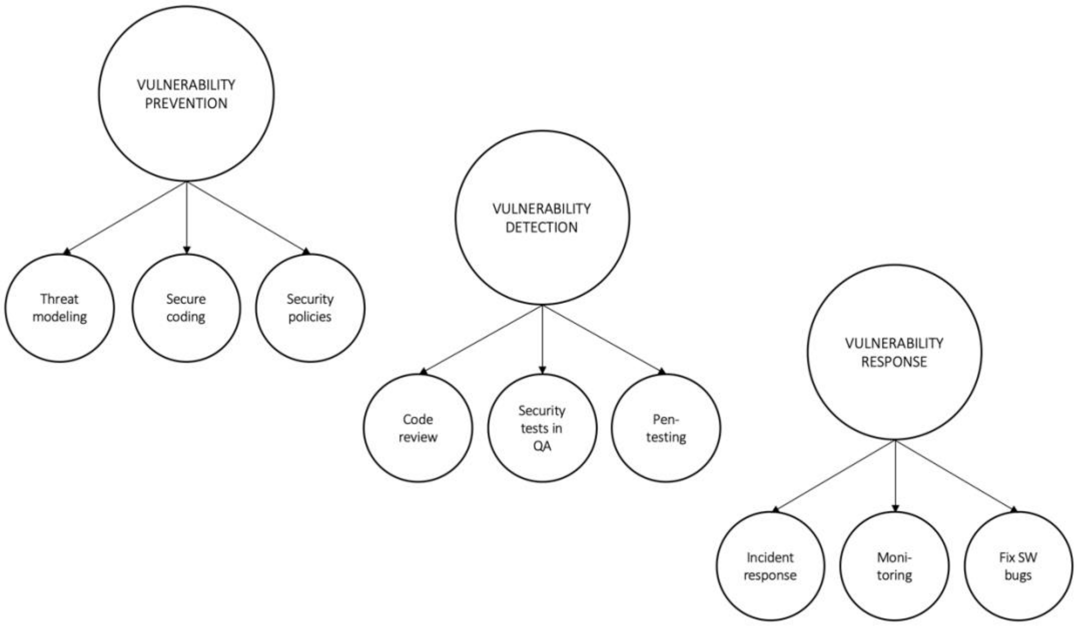

FIGURE 4 The three categories of security practices

The different security practices can be mapped into the software development life cycle, to make sure security is addressed in every phase of development and thereafter. Figure 5 shows an example by McGraw (2005) on how different security practices can be mapped onto the development life cycle.

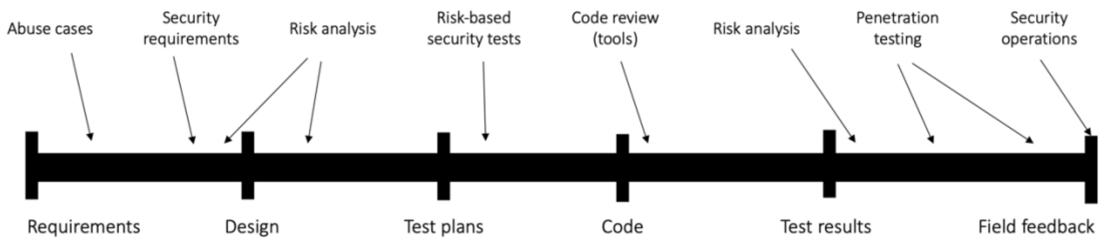

FIGURE 5 Ways to address security practices during development

In McGraw's (2005) approach in Figure 5, different software security practices are addressed during different phases of development. In the requirements phase, security is addressed through building abuse or misuse cases and creating security requirements. In the design phase, risks are analyzed first from the business perspective through considering the possible impact of security issues. After arriving at an initial design draft, risks are assessed from an architectural risk analysis perspective, which looks at the proposed design and its possible security issues. In creating the test plans, security tests should be included in the test plan. These tests should ensure the functioning of the security features as well as create some adversial tests to ensure the security of the software. After code has been created, it should be tested with static analysis to make sure the most basic bugs are eliminated. Once the build is (almost) finished, penetration testing should be used to ensure the security. Finally, once the product has been deployed, operational security should be configured and monitored. (McGraw 2005.)

Not all agree with the usefulness of the typical security design methods, though. In them, Dhillon and Backhouse (2001) find an inherent flaw: they all see the evaluated system or organization as a machine-like apparatus, that can be inspected using a checklist or other kind of evaluation methodology. In this aspect, the methods fail to see the organizations as social entities where the individuals in the organization play a significant role in how the organization – and security as its subset – is constructed. For example, they build on Mintzberg's (1983 cited Dhillon & Backhouse 2001, 139) work and name *power* as a social concept that highly influences how influence, authority and control are expressed in an organization. (Dhillon & Backhouse 2001.) Therefore, organizational structure and culture can be seen as important components of also the organization's security landscape – security is not just about systems, intruders and exploits. This view aligns with the DevOps principles that recognize the cultural component as an important facet for making a development method implementation go well. Adding security checkpoints to the SDLC as advised by a playbook is useful, but even more so when the security activities are chosen with an understanding of the organization's unique circumstances.

2.4.1 A call for DevSecOps

DevOps has been criticized for its lack of consideration for security. This doubt in DevOps' lack of attention to security comes from its urge towards rapid release of software which can raise suspicion of a lack of security prioritization in the development process. Still, doing things fast does not necessarily mean doing them badly. Worth mentioning is the fact measuring the level of a product's security is notoriously difficult. The conundrum of security is that it is best seen when there is a lack of it: that is to say, a non-secure system can raise headlines through cyber-attacks and data leakage, whereas a well-secured system stays unnoticed. Furthermore, the level of collaboration between the organization's security specialist and other IT personnel can be hard to evaluate. Intrerestingly, from the managers' perspective, the collaboration between security and developers can seem stronger than it really is. According to research (Puppet 2019), 64 % of upper-level management believe that the security personnel are involved in software development, whereas only 39 % of the software developers agree with this view. As the views on the collaboration levels and what is actually done differ, it is hard to know the exact state of security of an organization's development process.

How security can be integrated in to the DevOps process is a key research question in this paper. As a founding principle of DevOps is the culture of collaboration, also the security teams need to be incorporated into the culture of working together towards common goals. Curphey (2019) offers a good analogy to clarify the role of the security people and tools in the development process:

> Introducing security earlier in the development process creates a sort of blueprint for teams to build within the specifications of major IT regulation and helps reduce risk by fixing vulnerabilities before they can be exploited. Think of it like building an apartment complex according to city codes with inspectors integrated in the construction crew. (Curphey 2019)

Incorporating security into DevOps development in such an organized way has been coined DevSecOps (whilst some preferring SecDevOps, or secure DevOps). Figure 6 shows the Google Trends (2019) search results for DevSecOps and DevOps respectively.

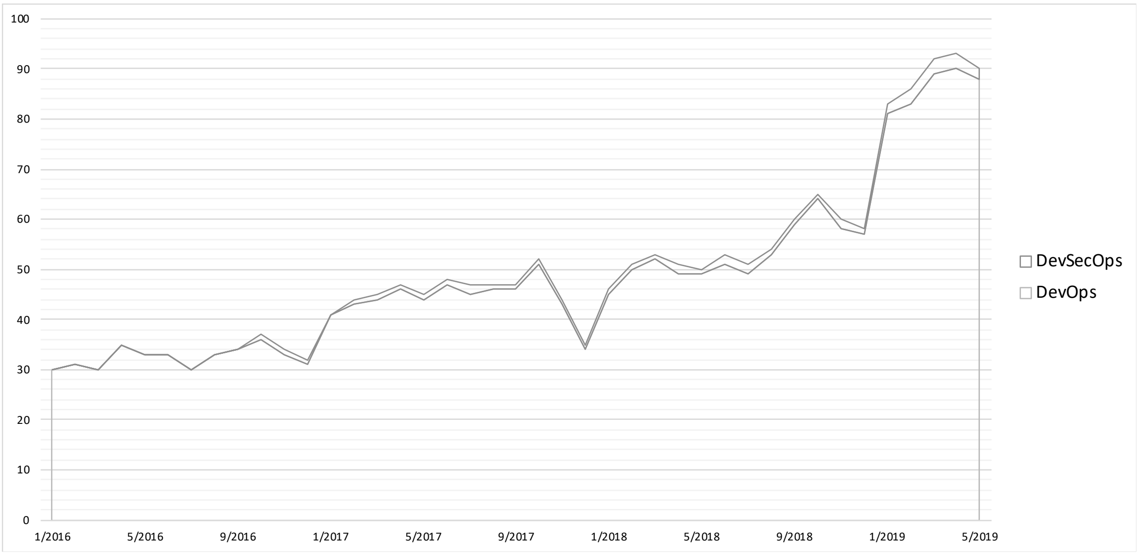

FIGURE 6 Search term trends for DevSecOps and DevOps on Google

Figure 6 shows the trend of the DevSecOps and DevOps search terms between 1.1.2016 and 31.5.2019. From Figure 6 we can observe how the popularity of both terms has grown. The use of the search term DevOps has grown over 50 % during the time period, whilst the popularity of DevSecOps remains a slight shadow by its side. The popularity of DevSecOps grows only slightly during the time period, but a growing interest might be on the way, as for instance Deloitte (2019) places DevSecOps as one of the top technology trends for 2019.

Doing DevSecOps does not simply mean adding security tools to the pipeline – instead, it requires also the Sec to be aligned and integrated with the before-mentioned four principles of DevOps. In Deloitte’s (2019) view, DevSecOps is a “transformational shift that incorporates secure culture, practices, and tools into each phase of the DevOps process.” Where exactly should the security practices shift in the faster paced delivery life cycle, is a top question posed by the industry. With DevOps’ fast delivery cycles and the desire for bringing customer needs to software functionalities quickly, a “shift-left paradigm” has surfaced (Lietz 2016). Shifting security to the left means to shift security practices to the early stages of the software development. For instance, penetration testing is an example of a security activity done typically very late in the development life cycle and threat modeling is an example of an activity done early in the development life cycle. To shift the security left, the focus should be on the practices that are done early in the life cycle. The shift to the left can be done using automated tools, such as static analysis. Automating security activities can function well in the DevOps pipeline, as integrating the security tools into the pipeline allows for increased developer self-service also when it comes to checking the security of the product. Shifting to the left also has other benefits, including making a product more secure and high-quality. The ideology seems almost self-evident: "With developers under constant pressure to create more software in less time, the last thing you need is for your code to fail at the end of the development life cycle. […] The sooner a developer can identify, view and correct a flaw, the more efficient it becomes to fix in the software development life cycle (SDLC)" (Curphey 2019).

Despite the growing interest of automating security activities, not all security practices should be added to DevOps from the get-go. A step-by-step approach might be better, as DevOps is a way of working that an organization adopts incrementally. DevOps maturity is achieved through iterations and learning. According to seasoned industry insiders (Puppet 2019), integrating security usually happens in the most mature stages of DevOps adoption. For those looking for an easy start, static analysis is a practice that can be integrated into the integrated development environments (IDE) and it is recommended as a security automation first both by Mansfield-Devine (2018) and Curphey (2019). By integrating security into the coding phase of the SDLC, the developer both gains self-efficacy when it comes to security, and costs are reduced as vulnerabilities are found early on. Later, security policies can be automated into the pipeline, where part of the DevOps culture of shared goals becomes to develop well-functioning software that also keeps the business assets and data secure (Puppet 2019).

2.4.2 BSIMM and measuring software security

As said before, software security is most noticeable when there is a lack of it. In other words, once the security measures fail and an attack is conducted and caught, the lack of security becomes apparent. Whereas a lack of security is noticeable, successful security is harder to measure. In order to be able to make security measurable, security maturity models have surfaced. Two of the best-known ones are Building Security In Maturity Model (BSIMM) and OpenSAMM (Jaatun et al. 2015). In these two models, an organization can disclose which software security practices it carries out and by comparing their status to others, rate themselves for the maturity of their software security. The two models have a different underlying philosophy: BSIMM, which learns from industry insights, captures the software security practices of world-class companies (for example Adobe, Cisco and PayPal) and makes the best security practices the yardstick of mature software development. OpenSAMM, on the other hand, has taken the existing best practices as the start point of their metrics and uses them to measure organizational maturity. (Jaatun et al. 2015).

For my research, I have chosen to use the BSIMM model as a framework against which I will plot the software security practices that my research uncovers. I chose BSIMM as the framework of choice because it has been developed using ”the largest set of data collected about software security anywhere” (McGraw et al. 2019). BSIMM represents a collection of the best practices and incorporates different facets of information security cleverly into one single model. When using the BSIMM model to gauge security maturity, an organization can see which security activities it currently does and determine the concluding maturity level. To enhance security and maturity, the organization can seek to add further security activities into its processes.

The BSIMM has 12 security practices in total, which are divided into four domains: Governance, Intelligence, SSDL touchpoints (SSDL stands for Secure Software Development Life cycle) and Deployment.

> **Governance** activities are those activities that manage and organize security practices – and also develop security further in the form of security education. These activities are generally conducted by the management level and they form the backbone of an organization’s security.
>
> **Intelligence** collects and uses organizational information that relates to information security(e.g., identification of assets and applicable security standards).
>
> **SSDL touchpoints** are activities that deal with the software development process and its components. They are the activities done before the product is deployed.
>
> **Deployment** activities deal with network security, configuration and maintenance.

Each above-mentioned domain has three security practices allocated into it. Table 2 illustrates the security practices, of which there are 12 in total.

TABLE 2 The four domains and twelve practices of BSIMM

Governance	**Intelligence**	**SSDL touchpoints**	**Deployment**
Strategy & Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance & Policy	Security Features & Design	Code Review	Software Environment
Training	Standards & Requirements	Security Testing	Configuration Management & Vulnerability Management

The 12 practices of Table 2 divide further into 116 BSIMM activities. Each activity is hosted within a maturity level and can be used to assess the maturity of the organization. (McGraw et al. 2019.) In my research I look at which software security practices current DevOps research has captured and where there are still research gaps. Therefore, I am using only the security practices as a framework in my research and not the maturity grading scales.

3 RESEARCH METHODOLOGY

3.1 Systematic literary review and the research process

The goal of this Master's Thesis is to gain an understanding of security in DevOps through a review of academic writing on the subject. The method used for this research will be a *systematic literature review* (SLR). A systematic literary review is beneficial for getting new research findings and learning research skills (Kitchenham & Brereton 2013, p. 2061). As such, it is a very suitable method for conducting a Master's Thesis, where one of the goals is to learn the art of research. A systematic literary review aims to understand a topic based on previous research (i.e., primary studies) that have been done on the selected research subject.

> The goal of a systematic review is to search for and identify all relevant material related to a given topic (with the nature of this material being determined by the underlying question and the nature of the stakeholders who have an interest in it). (Kitchenham et al. 2016, p. 10)

In this research, the underlying questions are the research questions. The stakeholders who I want to be able to benefit from this research, are the practitioners of secure software engineering and the research community, who might benefit from the research findings. Systematic literary review's role as a research method is indeed two-fold: 1) it provides new knowledge by analyzing prior research and 2) it notices and presents research gaps and thus identifies needs for new primary studies. The results of systematic reviews can both influence practitioners and also provide input to policies and standards (Kitchenham et al. 2016, 13).

A systematic review "aims to provide an objective and unbiased approach to finding relevant primary studies, and for extracting, aggregating and synthesizing data from these" (Kitchenham et al. 2016, p. xxxiii). This is done by selecting a research topic and research questions that have not yet been studied. The researcher uses prior studies, i.e. *primary studies,* to get answers to the research questions. The new knowledge produced by the systematic literary review is thus a synthesis and analysis of prior studies on the selected subject. A systematic literary review needs to be both systematic and rigorous. The researcher using the method first needs to identify a research gap and then, *without a bias*, find primary studies that are relevant to the research question(s). From the primary studies, the researcher makes an objective analysis on the research topic. The research process is illustrated in Figure 7 (adapted from Kitchenham et al. 2016).

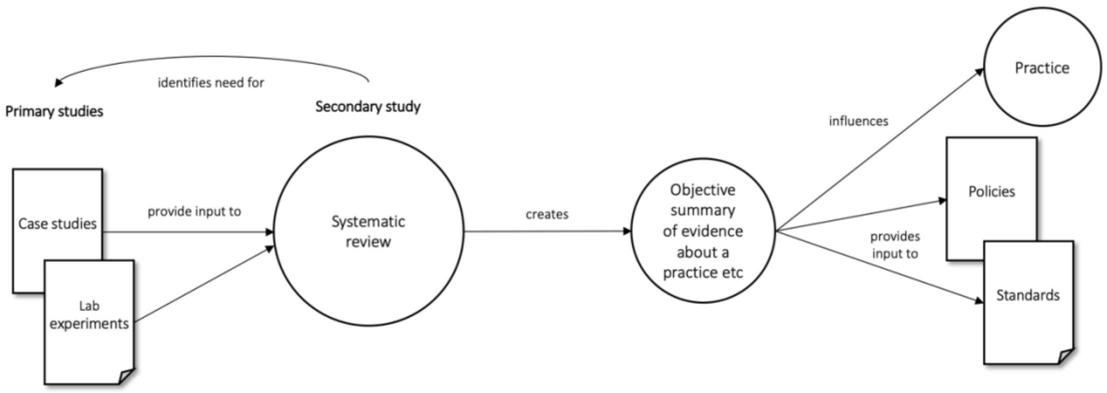

FIGURE 7 The inputs and outputs of systematic review.

As an input, the researcher uses primary studies, which can be e.g., case studies and lab experiments done by other researchers. The researcher uses search criteria to make a systematic decision system, which determines whether a study is included or excluded from the systematic literature review. From the included studies, the researcher produces a review which aims at providing an objective summary of the research topic – which, in this case, is the current state of research of security activities in the context of the DevOps development method. The systematic review can provide guidance by merging the results of previous studies into an output that can be used as a basis for policies or standards, or to identify needs for more primary studies.

3.2 Research questions

In this Thesis I am conducting a systematic literary review to get an understanding of how security practices can be integrated into DevOps according to available academic research. To understand the phenomenon, I formulated three research questions.

- RQ1: What are the challenges of security in DevOps as reported by the authors of primary studies?
- RQ2: Which security activities are associated with DevOps in the literature?
- RQ3: How are the CAMS (culture, automation, measurement and sharing) principles reflected in secure DevOps research?

The results of the research should offer a concise look at current state of security practices in the context of DevOps development. The first research question investigates which challenges the authors of primary studies link to security in DevOps. By looking at the challenges, I am hoping to capture DevOps' uniqueness as a development method and to understand how that uniqueness translates to the security activities DevOps needs in particular. The second research question charts which security activities have been linked to DevOps in research thus far. Through analyzing the results, we gain an understanding of where research efforts have been concentrated and where there might be research gaps. Through a synthesis of the recommended security activities, a rough draft of recommendable security activities for DevOps development is attained. The third research question goes through the primary studies and looks at how the studies speak of the four principles of DevOps. By analyzing different authors' understanding of culture, automation, measurement and sharing, it is possible to observe whether the authors see DevOps identically or whether different interpretations surface, as was the result in Jabbari et a.'s (2016) research.

3.3 Search method, strategy and criteria

In a systematic review, the researcher creates a search strategy that will guide the researcher towards finding the relevant primary studies on which she will base her analysis. In developing a search strategy, the objective is to come up with a relevant set of search terms that can be used to conduct a search string. The search terms are used to find research relevant to the research questions. When developing the search strategy, the researcher also makes strategical decisions on what type of input (e.g., journal articles, magazine articles) will be used for the research and from which databases the primary studies will be acquired. The researcher also determines the inclusion and exclusion criteria that will decide whether a certain study will be included in the final selection of the systematic review. The goal of the criteria is to make the research process systematic, non-biased and manageable.

In my research, the search terms were chosen as advised by Kitchenham et al. (2016). I wanted to find a set of search terms that would give me all the relevant results but not exhaust me with heaps of irrelevant research not related to the research topic. To come up with such a set, I did trial searches from digital databases with terms related to my research topic. Searching only with the term ″DevOps″ without adding other terms to it, would have resulted in an amount of search results unmanageable within the scope of a Thesis done by a solo researcher. Thus, through experimentation, I chose to add the term ″secur*″ with DevOps, to get results where DevOps is used in conjunction with terms such as secure or security. Also, to capture papers where secure DevOps is talked of using other terms, I added the terms "devsecops", "secdevops" and "devopssec" to my set of search terms. The search terms are listed in Table 3.

TABLE 3 Search terms that were used to search for relevant papers

Topics	Search terms derived from topics
Main research topic	devops & secur*

Variations	devsecops, secdevops, devopssec

The search terms were used to search for articles in four digital libraries, which were:

- Science Direct
- ACM Digital Library
- IEEE Xplore
- Springer Link.

These four databases were selected as they provide a good coverage of articles and have been used in other literary reviews on the field (e.g., Felderer & Fourneret 2015 and Souza et al. 2019). To have a system of including or excluding the search results from the final selection of the systematic literary review, an inclusion and exclusion criteria was developed.

The inclusion criteria of the articles selected for the study are:

- The article discusses security in relation to DevOps practices or technologies.
- The article has been published in a journal or it is a conference paper.
- The article is written in English.
- The article is accessible with the University of Jyväskylä rights.

The first of the inclusion criterion makes sure that the article happens in a DevOps context and is thus relevant to the research questions. The second inclusion criterion determines that the scope of this systematic literary review stays in academic research. In this research, I am interested purely in the current state of *academic* research on DevOps and therefore a criterion defining the academic nature of the research is a must. The third criterion marks off articles that are written in another language than English. Finally, the last criterion is a practical matter: the article has to be accessible with the University of Jyväskylä rights. To make the inclusion and exclusion even more precise, I also determined additional exclusion criteria.

The exclusion criteria are:

- The article is not relevant to the research questions.
- The text is an opinion piece.
- It is a duplicate article.

The exclusion criteria mark off articles that are not relevant to the research questions, i.e., if an article would seem suitable for the research based on the inclusion criteria but upon closer inspection offers no answers to the research questions, it is excluded from the study. The same applies, if the article is purely an opinion piece (for instance a column discussing security). Also, duplicate articles are excluded from the study.

3.4 Study selection process

The search was conducted in April 2019. The search string was constructed according to each of the digital libraries' search fields. Where it was possible, the search was narrowed down to yield only conference and research papers and for the search terms to be present in the title and/or the abstract of the article. The searches were conducted without limiting the year of publication, which resulted in works from the current year were also being included in the search results.

The automated searches using the search string yielded a total of 292 search results. The highest number of results came from Springer Link (236 results) where it was not possible to narrow the result list down based on whether the search terms were found in the abstract. Thus, the search yielded works where security and DevOps were mentioned even once in the whole text, and not surprisingly, many of these works were not relevant to my research.

From the initial search result of 292 articles from the four databases, I narrowed down the search results in two rounds. I did the initial round of discarding based on reading the article's title and abstract. Through this round I was able to discard 254 articles, leaving me with 38 articles left for closer inspection. In the second round, I read all 38 articles and used the inclusion and exclusion criteria to determine whether the article should be included in the final selection. Many literary reviews are done by research teams, where the inclusion/exclusion criteria are exerted as a team effort. When a team member is in doubt of whether to include or exclude an article, he/she consults another member of the research team and the matter is discussed until an agreement is reached. The hardest thing for me, as a solo researcher, was making the decision whether to include some article that was on the borderline of inclusion and exclusion. In such cases, I consulted my research questions and made the inclusion/exclusion based on the question: "Does this article feature any security activities of BSIMM and are they practiced in a DevOps context?" If the answer was yes to both, I included the article the study. Table 4 clarifies the study selection process per database and shows that in the final selection, two to five articles per database were chosen for the

TABLE 4 Study selection progress.

Database	**Initial results**	**After reading title + abstract**	**Final selection**
IEEE Explore	34 results	9 results	5 results
ScienceDirect	7 results	4 results	2 results
ACM Digital Library	15 results	8 results	5 results
Springer Link	236 results	17 results	4 results
Total	**292 results**	**38 results**	**16 results**

After narrowing down the search results to the final selection of 16 results, I conducted a backward and forward snowballing as per the guidelines of Wohlin

(2014). The snowballing practices are used to arrive at better coverage of the articles to review. The 16 chosen articles were used as a start set for the snowballing. In backward snowballing, the researcher looks at the references used by the start set articles with the aim of finding additional articles that would be relevant for the research. Through backward snowballing, I was able to find two more articles for the final selection. In forward snowballing, the researcher looks at articles that have cited the articles of the start set. For forward snowballing, I used Google Scholar to identify works, which have used the articles from my final selection as a reference. Forward snowballing yielded no positive results: most articles were already included in my final selection. Also, as the selected articles are on the newer side (with over 40 % published the year before the publication of this Thesis), the shortage of new articles from forward snowballing was understandable. By conducting both backward and forward snowballing, I was able to be confident that my final article selection of 18 articles offers a good coverage of the current research. Figure 8 (modelled after Felderer & Fourneret 2015) illustrates the paper selection process.

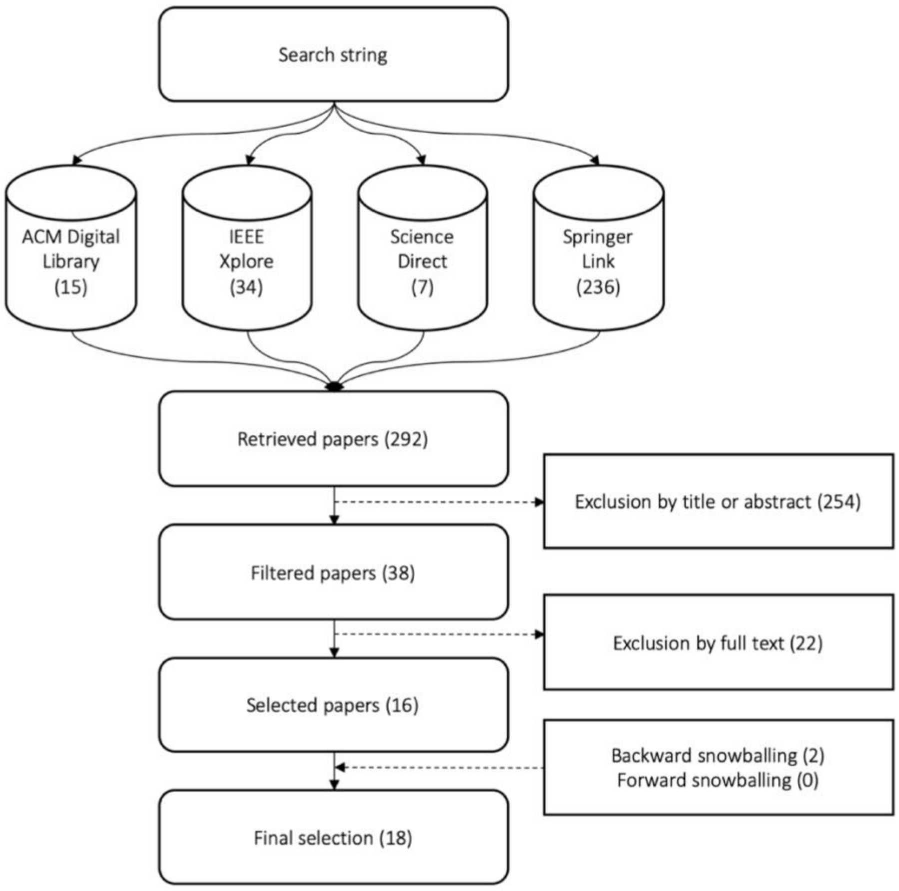

FIGURE 8 Selection process of papers

In the paper selection process the search string guides the search results of the four databases and through the filtering process, 16 papers a selected. Finally, two more papers are added through the practice of snowballing, arriving at a final selection of 18 papers.

3.5 Data extraction and analysis

To answer the research questions, I extracted data from the selected studies that fit my search criteria. To make the data extraction process rigorous and systematic, I formulated a data extraction spreadsheet. In the spreadsheet, I logged data items relevant to the research questions, as well as demographic data. The data extraction formulation follows Shahin et al.'s (2017) example and is showed in Table 5.

TABLE 5 Data extraction formulation

Data item	Description	Data needed for
Author(s)	The name(s) of the author(s)	Demographic data
Year	When was the work published	Demographic data
Title	The name of the article	Demographic data
Venue	The name of the publication venue	Demographic data
Summary	A summary of the article	Research questions 1-3
Challenges	Did the article mention challenges in security of DevOps?	Research question 1
Mentioned security activities	Which security activities were mentioned in the article and what was said about them?	Research question 2
Culture	Was culture discussed in the article?	Research question 3
Automation	Was automation discussed in the article?	Research question 3
Measurement	Was measurement discussed in the article?	Research question 3
Sharing	Was sharing discussed in the article?	Research question 3

For logging the BSIMM security activities, I created a separate spreadsheet where I entered all the BSIMM security domains with their correlating security practices and activities. For each reviewed article, I logged which security activities were mentioned. After logging the data for all articles, I had a spreadsheet which showed me the coverage of the BSIMM domains and enabled me to analyze the data further. The analysis of the data was done using qualitative methods.

The challenges to security perceived by the authors were extracted from the data using typification. I first recorded the different challenges, of which there was a total of 27, into the data extraction spreadsheet as they were explained by the authors. Through a process of analyzing the different challenges, I typified them into nine different types. The final results of the typification as well as an analysis of the results is offered in chapter 4.2. To find answers to the last two research questions, content analysis was used. First, to chart the different security activities mentioned in the reviewed works, content analysis was used to interpret the correspondence between the security activities mentioned in the articles and the BSIMM framework. Secondly, the articles were analyzed for mentions of the four principles of DevOps. The answers to the second and third research questions are presented in chapters 4.3 and 4.4 respectively.

4 RESULTS

4.1 Demographic data

The 18 articles included in this systematic literary review have a wide scope in the topics they handle. With topics ranging from insider threat to added IoT security, the researchers in the field have a wide variety of interests. As for the publishing years, the discourse on the topic of security DevOps practice seems to have risen over the past few years, as shown in Figure 9. The first academic work included in the review has been published in 2015 and the latest were published in the same year as this Thesis. So far, year 2018 has been the most active year on the field. The number of works in total that discuss security in DevOps is still relatively small and presents more research opportunities for future researchers.

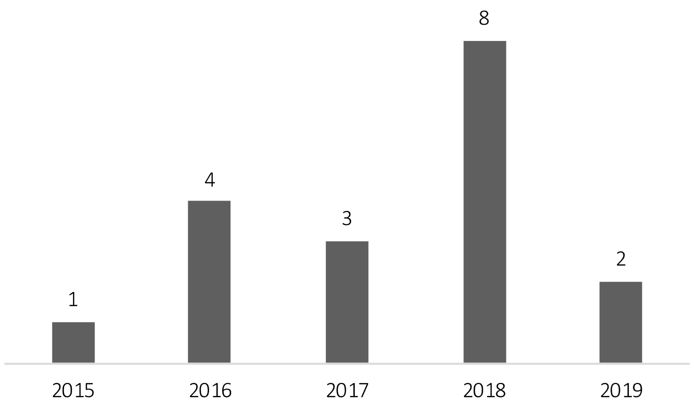

FIGURE 9 The publishing years of the reviewed articles.

There were no especially active authors in the field of the study, as only one author, Martin Gilje Jaatun, was the author of two articles [7 and 8]. Other authors were responsible for a single paper in the final selection only. Table 6 presents an overview of the final article selection. In my analysis of the results, I will refer to the articles by their ID number, which is shown in the first column. The full references of the articles are available in the Reference section of this Thesis.

TABLE 6 An overview of the final article selection

ID	Authors	Year	Description
[1]	Ahmanavand et al.	2018	Ahmanavand et al. take note of the security implications of using microservice-based architectures.
[2]	Bass et al.	2015	Bass et al. describe the process of hardening the deployment pipeline.
[3]	Beigi-Mohammadi et al.	2018	Beigi-Mohammedi et al. propose a self-protecting framework for DevOps environments. Their solution uses static analysis during development and dynamic analysis once the system is deployed.
[4]	Diekmann et al.	2019	Diekmann et al. explore enhanced network access control management in the container age.
[5]	Dullmann et al.	2018	Dullman et al. bring forth the importance of securing the development pipeline itself.
[6]	Ferry et al.	2019	Ferry et al. explore how the DevOps development method could be leveraged to serve the needs of development of trustworthy (e.g., secure, resilient and robust) smart IoT systems.
[7]	Jaatun	2018	Jaatun proposes that in the DevOps- era, incident management should be more closely connected to developers.
[8]	Jaatun et al.	2017	Jaatun et al. suggest a risk-based process for enhancing security in cloud-based solutions. They suggest a continuous process, where throughout the development life cycle risks are identified and assessed, then treated and controlled.
[9]	Mackey	2018	Mackey explores the use of open source components in software and how to ensure their security.
[10]	Mansfield-Devine	2018	Mansfield-Devine explores the current state of security practice usage in DevOps organizations and explores what are the best ways to implement security in DevOps.
[11]	Michener & Clager	2016	Michener and Clager have considered how organizations with "little tolerance for failure" can hold on to their compliance while adopting DevOps practices.
[12]	Ur Rahman & Williams	2016	Ur Rahman and Williams researched practitioners’ views and experiences on DevOps security.
[13]	Raj et al.	2016	Raj et al. suggest multiple different ways to harden Docker, which is often used in DevOps environments.
[14]	Rios et al.	2017	Rios et al. offer a security solution to multi-cloud environments, with an emphasis on continuous monitoring.
[15]	Schoenen et al.	2018	Schoenen et al. observe the complexity of cloud infrastructures that are comprised of multiple stakeholders and systems.
[16]	Thanh et al.	2016	Thanh et al. explore how microservices, which are frequently used in DevOps environments, can be made more secure by design.
[17]	Torkura et al.	2018	Torkura et al. conclude that technologies used frequently in DevOps can contain vulnerabilities and that these can be hard to identify.
[18]	Ullah et al.	2017	Ullah et al. propose a method for hardening a continuous deployment pipeline.

The inclusion criteria presented in chapter 3.3 determined that only articles that were published in journals or as conference papers were included in the study. Of the 18 articles in the final selection, only three (papers [4], [9] and [10]) were journal articles and the rest were conference papers. The papers were from distinct conferences. Only papers [7] and [8] were from the same conference venue, *International Conference on Availability, Reliability and Security (ARES)*, though they were from separate years. Of the journal articles, two (articles [9] and [10]) were from the same journal, *Network security*. Table 7 shows the publishing venues of all of the articles of this systematic literary review.

TABLE 7 The publishing venues of the articles

ID	Published in	Year	Journal or conference
[1]	Software Technologies: Applications and Foundations (STAF)	2018	Conference
[2]	IEEE/ACM International Workshop on Release Engineering	2015	Conference
[3]	Annual International Conference on Computer Science and Software Engineering (CASCON)	2018	Conference
[4]	IEEE Transactions on Network and Service Management	2019	Journal
[5]	International Workshop on Rapid Continuous Software Engineering (RCoSE)	2018	Conference
[6]	Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (DEVOPS)	2019	Conference
[7]	International Conference on Availability, Reliability and Security (ARES)	2018	Conference
[8]	International Conference on Availability, Reliability and Security (ARES)	2017	Conference
[9]	Network Security	2018	Journal
[10]	Network Security	2018	Journal
[11]	Annual Computer Software and Applications Conference (COMPSAC)	2016	Conference
[12]	IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED)	2016	Conference
[13]	International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)	2016	Conference
[14]	IEEE Conference on Communications and Network Security (CNS)	2017	Conference
[15]	International Conference on Service-Oriented Computing (ICSOC)	2018	Conference
[16]	International Telecommunications Network Strategy and Planning Symposium (Networks)	2016	Conference
[17]	International Conference on Security and Privacy in Communication Networks (SecureComm)	2018	Conference
[18]	International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)	2017	Conference

4.2 Challenges to security

To answer the first research question of *What are the challenges to security in DevOps as reported by the authors of primary studies?* I reviewed the articles to uncover security challenges. My research found nine different challenges from the reviewed articles that showcase the typical security challenges of DevOps development as perceived by the authors of the reviewed works. The challenges are listed in Table 8.

TABLE 8 Security challenges in DevOps

Security challenges in DevOps	**Mentioned in [article ID]**
Ensuring pipeline security	[1], [2], [4], [5], [13], [18]
Balancing security and fast deliveries	[3], [16], [17], [10]
Increased insider access	[1], [4], [12], [18]
Balancing automated security activities with manual activities	[8], [9], [10]
Getting the security requirements right	[4], [5], [11]
Getting developers' security knowledge to the required level	[6], [10]
Finding the right security activities and tools that fit DevOps development style and technologies	[7], [12]
Faster deliveries require constant monitoring and faster bug-fix processes	[14], [15]
Including the security team in the development life cycle	[12]

The most frequently mentioned security challenge in the reviewed works was ensuring the deployment pipeline’s security. This challenge was mentioned in six of the 18 reviewed articles which makes it the most prominent of all challenges. The literature consistently states that in DevOps environments, the existence and importance of the deployment pipeline and the microservice as well as cloud infrastructure brings a new level of security challenges to the users. Article [15] describes this new situation best:

> Cloud services and cloud infrastructures become increasingly complex and dynamic: many different physical and virtual machines, applications and their components interact and all of these entities may be differently reconfigured, deployed, and migrated during run time. (Schoenen et al. 2018, p. 296)

The challenge thus becomes, how do you keep a desired level of security in an environment that is constantly changing? The reviewed works make it clear that any organization wanting to adopt the DevOps development style and technologies (e.g., microservices, containers, cloud-based solutions) should take the time fully consider how to ensure the security of each and every component. For instance, the authors of [17] identified several possible security issues arising from the use of application containers: container misconfigurations, image vulnerabilities (due to no automatic updates), image configuration vulnerabilities (e.g., an image might contain default credentials and thus need security hardening before use), downloading images from untrusted sources and not validating them properly. Article [18] stressed the importance of assessing the security risks of each component (e.g. repository, CI server, main server) in the pipeline. To enhance the security of the technologies, article [17] suggested creating security policies for each used technology and a schedule for their periodic security assessments. The technology-based security policies could be used to create a baseline, which would then be ensured in pre-deployment security tests, according to [17].

The second most frequently mentioned challenge has to do with balancing DevOps' fast deliveries with security. In DevOps development, there is on the one hand a desire towards making deliveries as fast as possible, but on the other hand the will to deliver high quality and customer value. Balancing these two aspects creates a challenge for security. To make fast deliveries and security a trouble-free pair, security activities should be fitted into the development process in a way that does not slow the deployment down unnecessarily, as stated by [10]. If the goal of DevOps development is to deliver software changes fast through automation, the only way security can be incorporated into the rapid-rate development is through automating many of the security activities as well. According to [12], automating security can have both negative and positive effects on security. This conundrum is explained by the fact that when security is automated, it relies on tools. If those tools are chosen poorly or do not actually match the technologies used, the desired security effects are not achieved, as recorded by [10] and [12]. A tool also seldomly works completely on its own: in case security issues are found, the output of security tests most often needs manual attention. Therefore, an important matter registered by [10] is to make sure that the process in the organization is clear on what to do with the findings of the automated tools: defining who is responsible for fixing the issues found by static analysis and within what time frame, when should the security team be called in, when should the build fail etc. A key issue in this process becomes the collaboration between the Dev, the Ops and the security team, where all members are working towards the same goal with the desire to enforce the organization's vision – whether that is security-oriented, delivery-oriented or quality-oriented. According to [10], in most organizations practicing DevOps, there are still difficulties in this process, as security is often only moderately involved with development and operations. To be able to fix surfaced security issues as fast as possible, good collaboration and communication with the security team is needed. When the security team is included in the development process, security issues identified during development can be solved together.

The third most common security challenge was the increased insider access that comes from allowing the “Dev” access to the “Ops” environment and vice versa. Though this increased access makes the development flow more easily and is in the heart of DevOps, articles [1] and [18] highlight the security implications this raises. The DevOps principle of sharing and the culture of collaboration are great and noble philosophies to admire, but in practice, they raise a conundrum for the security experts. If the Dev can also do Ops and the Ops can also do Dev, the bottom line is that there is much wider access to systems by insiders than before. As a consequence, new security controls that take this insider threat into consideration are needed. Article [1] states that it is important to know who has access where and to have proper logging measures to make sure also insider non-repudiation is achieved. Logs must also be tamper-proof. In a continuous integration/deployment environment, logs must be able to provide information on who is responsible for the deployed changes. As a practical bad example, article [11] pointed out a severe vulnerability in a popular firewall system, of which the developer company of the firewall could not explain “how the code got there or how long as it has been there”. An organization should make sure that they do not end up in such a situation but always have the means to log the activities of the developers and be able to track any changes that have been made to the system. It is also good to keep in mind that compliance requirements can pose demands on separating who can do what in which environment, as stated by [11]. In other words, compliance-heavy organizations might have to limit the access of Dev and Ops to each other’s environments to make sure compliance requirements are met.

Another security challenge noticed by three articles ([8], [9], [10]) had to do with balancing automated security activities with manual activities. That is to say, any organization implementing automated security activities needs to understand that not everything can be automated. For example, automated tools can easily be used to analyze code for vulnerabilities but making sure the design is secure usually needs manual review by security experts. Many automated tools supervise the adherence to the policies created by the organization – and in order to do this, a policy has to be created manually. For this exact reason, another set of articles ([4], [5], [11]) highlighted the need to get the security requirements right. Without having the correct security requirements, it’s very hard to keep a check on having the security at the correct level. Especially, if using monitoring solutions, one has to know which metrics should be monitored and what are the acceptable levels for those metrics. As said before, there also needs to be a clear process in place as to what to do when the monitoring shows undesired results.

The rest of the security challenges recorded in the reviewed works got only a few mentions. Two works ([6] and [10]) noted that the fast pace of DevOps might pose additional requirements for developers’ security skills. For instance, if it is part of a developer’s job to know which parts of code should be tested for security more rigorously or to fix the findings of security tests, additional security training is needed. Another set of works ([7] and [12]) talked of the lack of resources available for understanding which security activities align with the DevOps development style and technologies. Authors of [14] and [15] saw it as a challenge that the rapid delivery cycles require new types of security activities and increase the need for their speed as well. For instance, bug-fix processes during runtime need to be sped up in order to keep the deployed product secure. Finally, article [12] concluded that if the security team is not closely involved with the development and operations, there is the risk of possibly releasing software that contains vulnerabilities.

4.3 Security practices/activities

For the second research question, I analyzed the articles for mentions of security activities that relate to the activities in the BSIMM framework. Some of the security activities were easy to map, where as other required a bit more thought: does a certain activity described in an article match the activity described in the BSIMM framework. When in doubt, I read the BSIMM descriptions of the security activities carefully and compared them to the security activities described in the text. In paper [7] the BSIMM activities were suggested directly by the author who used the BSIMM framework as a reference point for security activities that could benefit incident management. In other papers, I analyzed the solutions of the authors and mapped which BSIMM activities they would be equivalent to.

The BSIMM framework comprises of four security domains, which are divided into twelve practices, which in turn comprise of a total of 116 security activities. The results from the research showed security activities from each security domain and practice. The reviewed works mentioned 47 different security activities. When analyzing the results, I counted mentions of a security activity by different articles as separate mentions. Thus, counting all of the mentions together, there was a total of 139 distinct mentions of security activities. I plotted the security activities mentioned in the reviewed works against the BSIMM framework and the results can be viewed in Table 9.

TABLE 9 Security practice and activity mentions in the primary studies

Governance (23)	**Intelligence (21)**	**SSDL touchpoints (36)**	**Deployment (59)**
Strategy & Metrics (8) Identify gate locations, gather necessary artifacts. [6] [10] [11] [17] Enforce gates with measurements and track exceptions [11] Create or grow a satellite. [7] [8] Require a security sign-off [11]	Attack Models (10) Create a data classification scheme. [1] [15] Identify potential attackers. [1] [10] Gather and use attack intelligence. [9] [10] Build attack patterns and abuse cases. [1] [15] Create technology specific attack patterns. [2] Build an internal forum to discuss attacks. [1]	Architecture Analysis (13) Perform security feature review. [1] [2] [5] [10] [11] [12] [14] [15] [17] Define and use AA process. [11] [12] [15] Make the SSG available as an AA resource or mentor. [11]	Penetration Testing (4) Feed results to defect management and mitigation system. [3] Use penetration testing tools internally. [3] [10] [11]
Compliance & Policy (13) Unify regulatory pressures [11] [12] Create policy. [3] [4] [12] [16] [17] [18] Identify PII data inventory. [7] Implement and track controls for compliance. [11] [12] Include software security SLA in all vendor contracts. [14] Impose policy on vendors. [14]	Security Features & Design (2) Build and publish security features. [16] Build secure-by-design middleware frameworks and common libraries. [16]	Code Review (12) Use automated tools along with manual review. [2] [10] [11] [12] Make code review mandatory for all projects. [10] [11] Use centralized reporting to close the knowledge loop and drive training. [10] Use automated tools with tailored rules. [3] [9] [10] [17] Use a top N bugs list. [2]	Software Environment (43) Use application input monitoring. [7] [12] [14] [16] Ensure host and network security. [1] [2] [3] [4] [5] [13] [14] [16] [17] Use application behavior monitoring and diagnostics. [3] [5] [6] [7] [8] [12] [14] [15] [16] [17] Use application containers. [1] [4] [13] [16] [17] Use orchestration for containers and virtualized environments. [4] [6] [13] [14] [16] [17] [18] Enhance application inventory with operations BOM. [2] [9] Ensure cloud security basics. [8] [14] [15] [16] [17] [18]
Training (2) Provide awareness training. [12] Establish SSG office hours. [7]	Standards & Requirements (9) Translate compliance constraints to requirements. [4] [12] [15] Identify open source. [9] [10] [12] Control open source risk. [7] [9] [12]	Security Testing (11) Drive tests with security requirements. [2] [3] [5] [6] [12] [17] Share security results with QA. [10] Include security tests in QA automation. [6] [10] Drive tests with risk analysis results. [15] Begin to build and apply adversial security tests (abuse cases). [6]	Configuration Management & Vulnerability Management (12) Create or interface with incident response. [7] Identify software defects found in operations monitoring and feed them back to development. [7] [8] [10] [14] Have emergency codebase response. [7] Track software bugs found in operations through the fix process. [8] [10] Develop an operations inventory of applications. [7] [9] Simulate software crises. [5] [7]

Table 9 is divided according to the logic of the BSIMM framework. The columns show the four different domains: Governance, Intelligence, SSDL touchpoints and Deployment. Each domain consists of three practice categories, so there are twelve practice categories in total. The practices are comprised of activities, which are smaller security-related actions or processes. For example, in the top left corner, *Governance* is a domain, *Strategy and metrics* is a practice and *Identify gate locations, gather necessary artifacts* is an activity. Table 8 shows which activities were mentioned in the reviewed works. The numbers in brackets ( ) show the total of mentions per practice or domain and the numbers in square brackets [ ] refer to the article ID. For example, the security activity *Identify gate locations, gather necessary artifacts* got four mentions, as there were four distinct articles ([6], [10], [11], [17]) that mentioned it. From Table 9 we can see that the domain with most mentions is Deployment (59 mentions out of the 139 mentions in total). The second most popular domain, SSDL touchpoints, is far behind with only 36 mentions of activities. As for the lower-level practice categories, the most popular is Software Environment with 43 mentions of the security activities in that category. The most often mentioned activity was *Use application behavior monitoring and diagnostics*, which was mentioned in ten of the reviewed articles. This observation goes well together with DevOps' principle of Measuring. In other words, in the heavily automated pipelines, the usefulness of monitoring and using data from deployed applications is heavily appreciated. Figure 10 illustrates how the mentioned security activities are divided into different categories of the BSIMM framework.

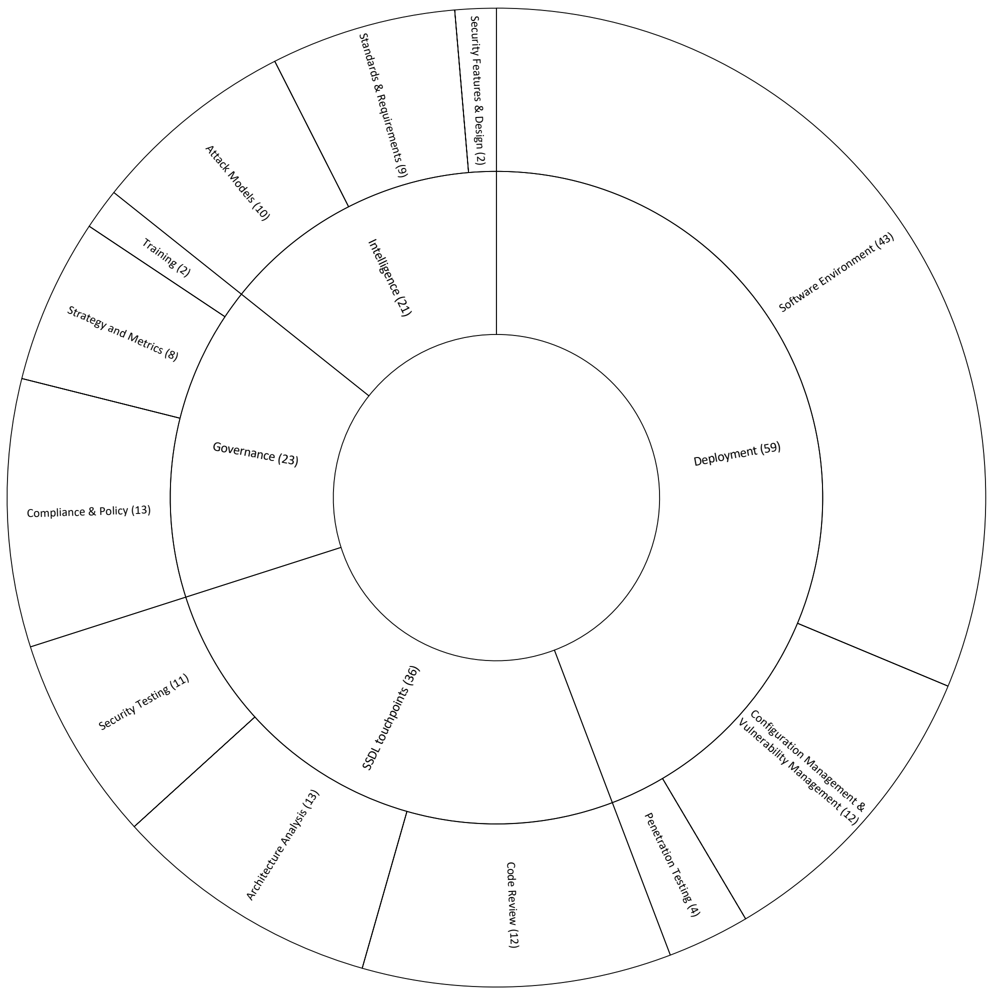

FIGURE 10 Research's coverage of the BSIMM practices.

When the security activities from the reviewed works are plotted against the BSIMM framework we notice that the current research focuses heavily on deployment activities and in particular, securing the software environment and configuration management. All and all, 42 % of the security practices that are mentioned in current research on DevOps security, deal with deployment activities. As is evident from Figure 10, the intelligence and governance domains have gotten the least attention in current research on secure DevOps. In particular, the security practice categories of training, security features & design as well as penetration testing got only a few mentions each. As DevOps is a development method (Jabbari et al. 2016), one could presume that research on secure DevOps would focus on the domain that deals with software development, SSDL touchpoints. However, my research points out that this is not the case. Based on the current research, researchers focus on the securing the DevOps environment as well as its key technologies. Security activities that influence the security governance and secure development are left far behind in popularity.

To delve a little bit deeper into the security activities that were recommended in the reviewed works, I will next talk about them in more detail. I will explain the authors' view on the thirteen most recommended security activities here, as each of those activities received four or more mentions in the reviewed works. Thus, they give an overview of the activities that over 20 % of the reviewed works recommend. The security activities that got less than four mentions are not presented here in detail, and for the curious reader I recommend the BSIMM report (McGraw et al. 2019) for a more comprehensive look at them. The thirteen most recommended activities are listed in Figure 11.

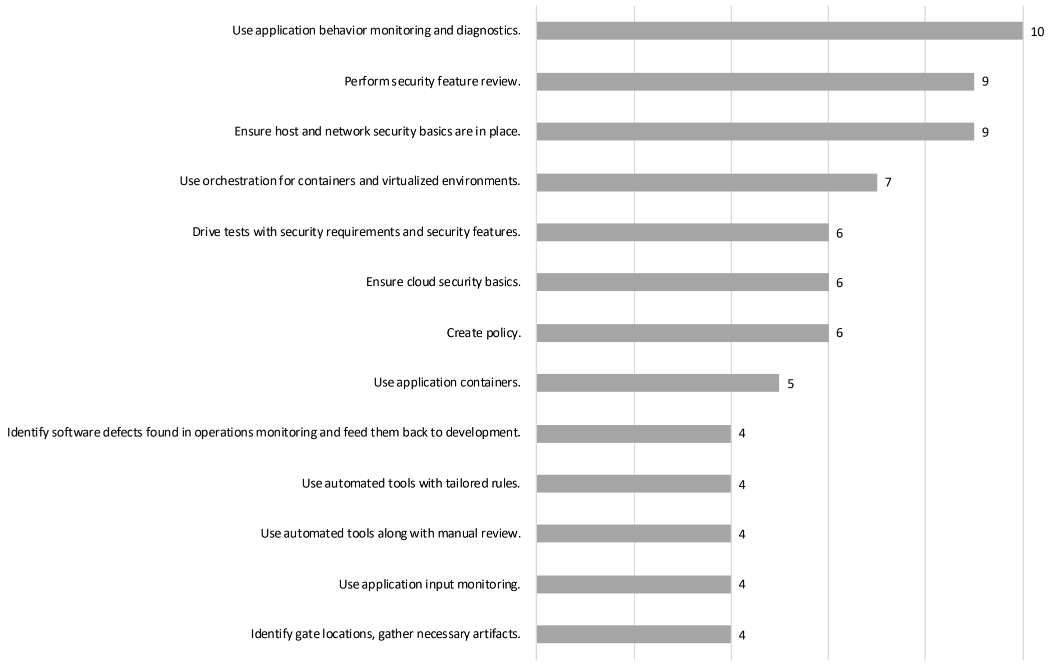

FIGURE 11 The thirteen most mentioned BSIMM security activities

Out of the top-13 activities listed in Figure 11, six activities were from the BSIMM practice group *Software Environment*:

- Use application behavior monitoring and diagnostics,

- Ensure host and network security basics are in place
- Use application containers
- Ensure cloud security basics
- Use orchestration for containers and virtualized environments and
- Use application input monitoring.

The practice group *Code review* contains two highly recommended practices, *Use automated tools with tailored rules* and *Use automated tools along with manual review*, which both received four mentions. The five remaining security activities that got four or more mentions were from miscellaneous security practice groups (Strategy & Metrics, Compliance & Policy, Architecture Analysis, Security Testing, and Configuration Management & Vulnerability Management). This shows that while the majority of current research is focused on security activities that deal with the software environment, also other security practices are getting attention from the authors, though to a lesser degree. Next, I will present the 13 most mentioned security activities and the authors' discourse on them.

4.3.1 Use application behavior monitoring and diagnostics

Over half of the reviewed works ([3], [5], [6], [7] ,[8], [12], [14], [15], [16], [17]) recommended monitoring the application during runtime, which is in alignment with the BSIMM activity *SE3.3 Use application behavior monitoring and diagnostics*. In [3, 6 and 15], monitoring during runtime provides information upon which the system can self-diagnose and self-correct. In [3], the system is even made aware of the trade-offs between quality and security and can determine which actions it should take. In [5], the application being monitored is the pipeline itself. By analyzing the behavior and performance of the pipeline, possible security issues can be catched early. In [7], the author perceives the ability to fix issues fast as DevOps' strength and therefore he views well-performed incident management a key component of DevOps security. To arrive at great incident management, one needs proper monitoring and diagnostics that catch the issues as soon as they arise. In [8], the monitoring is turned towards cloud service providers and making sure that the service is up to par with the requirements set for them. Paper [12] analyzed DevOps practitioners' views on beneficial security practices and noted automated monitoring as one of the most highly recommended ones. In [17], a monitoring system is used to monitor the health and security of microservices.

4.3.2 Perform security feature review

The BSIMM activity *AA1.1 Perform security feature review*, which belongs to the Architecture Analysis practice group and was mentioned in eight works ([1], [2], [5], [10], [11], [12], [14], [15], [17]). This security activity, which is also known as threat modeling, is one of the basic "traditional" security activities. In DevOps, threat modeling's role has been debated, as incorporating an activity that is based on the design of the system is difficult in processes where the design unfolds incrementally. BSIMM (McGraw et al. 2019) defines this activity of reviewing security features as looking at the most security-heavy features of the application (encryption, authentication and authorization etc.), and after identifying potential threat situations, analyzing whether the security is sufficient. In the works reviewed for this systematic literature, the authors had varying views on how a security feature review can be accomplished in DevOps. The most heavy-handed approach was in [11], where solely the threat model determines whether a piece of code will be accepted by developers via self-service, or whether it requires approval from the security team as well. In [1] the focus was on understanding what security features are needed to arrive at sufficient protection from insider threat in DevOps environments. This was done through analyzing potential attackers, the system's assets, possible threats and then finally by conducting a security analysis that was based on all of the aforementioned factors. In [2], the authors wanted to secure a deployment pipeline and in order to do so, first used threat modeling to understand the threats to the pipeline. Securing the pipeline was also the focus on [5], where the authors used the STRIDE threat modeling method to find out the possible threats to their system. A similar approach, though without a specific threat modeling method, was used in [17], to first document the biggest threats to application containers and then developing ways to prevent those. In [15] security features were reviewed from another angle: from figuring out which combinations of cloud service features would lead to security or privacy violations. Then monitoring was used to ensure that this combination is avoided (and if it is not, the system self-adapts accordingly). Article [10] reminds us that in the agile world of DevOps, a heavily documented threat model is not what is needed, but instead communicating the threat model's practical implications to the developers – what are the things that they need to understand and to consider as they work. Another thought from the same article is that in DevOps security work can happen at two levels: one dynamically in the pipeline and another out of band in securing the whole process through threat models, risk management etc. Where these two levels meet though, is when a change in development introduces a new asset, which should then be reflected back to the out of band security work in the form of threat modeling. Article [12] which drew its wisdom from DevOps practitioners, was also specific on the need for both automated and non-automated activities in secure software development – with threat modeling being a key component of the latter group. Threat modeling was conducted as part of the development process in six of the interviewed nine top-level DevOps organizations. In [14] security features were modelled for risk assessment purposes.

4.3.3 Ensure host and network security basics are in place

As stated before, many of the reviewed works focused on securing the DevOps environments, which is in alignment with BSIMM activity *SE1.2 Ensure host and network security basics are in place*. BSIMM (McGraw et al. 2019) includes in this activity e.g. patching, firewall configurations and cloud service security. In the reviewed articles, the host and network security spanned quite a wide area. In [1] the development environment is secured against insider threats using several security practices (e.g., access control, hardening, logging). In [2] the focus was also on the deployment pipeline and in identifying all its untrustworthy components and undertaking security activities to make sure the desired level of security is achieved. In [3] runtime security was achieved through monitoring security metrics, identifying vulnerabilities and using adaptive self-defense mechanisms to protect the system (e.g., automatic patching and firewall actions). Article [4] used a firewall setting within a container infrastructure to make sure the security requirements set by the organization are followed also on "low-level network access control lists". As access lists are prone to misconfigurations by the administrators, the authors' solution offered a visual representation of the applied firewall rules to enhance correctness of the configuration. In [5], the availability of the pipeline is seen to be of utmost value to the business and to ensure it, four security practices were suggested: 1) *canary testing*, where releases are tested on small fragments of the environment and their behavior is monitored, and in case of errors, a rollback is easy to do, 2) *A/B testing*, that, in this case, could be used to monitor the performance of different pipeline configurations, 3) *fault injection*, that would test the resilience of the system by choosing to put it under strain and 4) *monitoring*, to constantly identify possible security and/or performance issues. Article [13] focuses on hardening the development environment and in [16] the attention is turned towards ensuring cloud security and for instance a Firewall as a Service extension is used. In [14], the multi-cloud environment is secured through a network monitoring agent. Article [17] aims at ensuring that security is achieved in the pipeline, all the way to the deployment and during runtime as well.

4.3.4 Use orchestration

The BSIMM activity *SE3.5 Use orchestration for containers and virtualized environments* is explained by BSIMM (McGraw et al. 2019) to mean that automation is used "to scale container and virtual machine deployments in a disciplined way. Orchestration processes take advantage of built-in and add-on security controls to ensure each deployed container and virtual machine meets predetermined security requirements." In this systematic literary review, orchestration was mentioned in seven works ([4], [6], [13], [14], [16], [17], [18]). Article [4] wanted to draw attention to the need to secure containers and to ensure network access control within the development environments. In [18], the focus was also on the deployment pipeline and in how that environment can be orchestrated into a more secure one through use of virtual machines in the CI server. In [6], the goal of the authors was to provide a domain-specific modelling language that enables orchestration in smart IoT development in a "secure and context-aware" manner. Article [13] promotes security hardening of Docker, which can be used in secure cloud orchestration. In [14], the authors discussed the security implications of a multi-cloud situation, i.e., of using more than one cloud service provider (CSP), which requires security orchestration to enable changing the CSP dynamically. Microservice management and orchestration was a key topic in [16]. In that article, orchestration was policy-driven, which means that security requirements were translated into policies, which, in effect, are translated into rules that the framework obeys. Article [17] leverages security policies to dynamically allocate resources as well as assess vulnerabilities that might potentially be harmful to the application.

4.3.5 Drive tests with security requirements and features

In the BSIMM activity *ST1.3 Drive tests with security requirements and security features*, security testing is driven by the security requirements and features with the goal of verifying that the requirements are met and the features function as expected. A similar approach was mentioned in six of the reviewed works ([2], [3], [5], [6], [12], [17]). In [2], securing the pipeline starts with identifying its security requirements and then fixing found problems until the requirements are satisfied. In [5], the pipeline's resilience is tested via fault injection – deliberately testing how much failure it withstands. In [3], the self-adaptive monitoring system worked based on the security requirements – it constantly monitors that the requirements are met and uses self-adaption in case of violations. The framework described in [6] contains a system that simulates and tests the security requirements of IoT systems before they are released. In [12], eight out of nine DevOps practitioners state that manual security testing is part of their organization's development routine to "ensure that the software's functionality is properly implemented." In [17], automated tests were based on the security requirements.

4.3.6 Ensure cloud security basics

The BSIMM activity *SE3.7 Ensure cloud security basics* deals with making sure that also cloud deployments are done in a secure manner. This was a security activity mentioned in six of the reviewed articles ([8], [14], [15], [16], [17], [18]). Articles [8 and 14] were focused on enhancing cloud security, the latter concentrating especially in multi-cloud environments. The authors of [15] wanted to ensure especially privacy requirements in cloud environments and in [16] both the privacy and the security of cloud services was in focus. In [17] the cloud's security surveillance was done in a dynamic fashion to keep an eye out on potential vulnerabilities of applications and containers. In [18], the focus was on enhancing the security of the pipeline in a setting where the repository was in GitHub and the main server was hosted by Amazon Web Services (AWS). With both services, the authors considered uncontrolled access as a key security risk that needs to be mitigated to ensure security of the pipeline. Rounding up, all of the aforementioned six articles help DevOps security to consider different facets of the environment and to understand how security should be addressed in the cloud.

4.3.7 Create policy

With the security activity *CP1.3 Create policy,* BSIMM (McGraw et al. 2019) recommends that the organization creates software policies centrally, so that each project in the organization does not have to go through internal, external and compliance requirements on their own. Six of the reviewed articles ([3], [4], [12], [16], [17], [18]) referred to policy as a noteworthy security activity. In [3], the security policy is the underlying component that drives security requirements and enables them to be monitored automatically. In [4], the focus was on configuring network access policies correctly, which was done by first creating security goals, then deriving a policy out of those goals, and finally, enforcing the policy through access control configurations. In [12], policies were enforced both in an automated and non-automated fashion: in automation, an automated software defined firewall makes sure access is done according to policies. In the non-automated security activity, security policies are *performed*, i.e., a review process ensures that the policies are adhered to. In [12], all of the surveyed DevOps organizations conceded to performing policies in a non-automated fashion, whereas only six of the nine organizations claimed to use an automated software defined firewall for policy surveillance. In [16], such a firewall acts precisely in a policy-driven fashion and in [17], a *security gateway* is used to enforce policy requirements in microservice architectures.

4.3.8 Use application containers

BSIMM activity *SE3.4 Use application containers* was mentioned in five articles ([1], [4], [13], [16], [17]). In most of these, containers were not mentioned to be used explicitly for security purposes, but rather presented from the perspective of containers being a typical technology used in DevOps practices. The authors of these articles wanted to present their views on how to enhance the security of the DevOps-typical technologies. In [1], container security is leveraged through enhanced access control to suppress the threat of insider violations. In [4], the idea is similar but the access control policies are presented visually to limit the possibility of unintentional misconfigurations. In [13], hardening techniques are used to enhance the security of container technologies and in [16] container security is considered throughout the application life cycle in a policy-driven fashion. In [17], the focus is on identifying vulnerabilities in containers dynamically.

4.3.9 Send SW defects found in monitoring back to development

DevOps culture should be all about sharing issues and learning from them collectively. A good example of such an activity is the BSIMM activity *CMVM1.2 Identify software defects found in operations monitoring and feed them back to development*. This type of security activity was mentioned in four articles ([7], [8], [10], [14]). In [7] the focus of the article was incident management and naturally, in order to learn from mistakes and to grow, feedback from incidents should be fed back to development to deepen the understanding of how such events could be prevented in the future. In [8], the idea was that the rapidness of the DevOps development cycle can be leveraged for dealing with security issues found in operations: security bugs are fed back to development and fixed quickly. Article [10] offers another piece of advice: the security team's findings (e.g., via threat intelligence), should be communicated to the development, who can then use that information for building more robust products. In [14], the idea was that operations monitoring can find flaws in the (multi-)cloud infrastructure, and in some cases these findings can lead to a re-design of the application or changing the components to ensure security is achieved again.

4.3.10 Use automated tools along with manual review

In the BSIMM activity *CR1.4 Use automated tools along with manual review,* the goal of the organization is to leverage the code review process through automating parts of it. Such a view towards code review was found in four of the reviewed articles ([2], [10], [11], [12]). Static analysis was mentioned in [2] as one of the ways of making an application trustworthy. Article [10] emphasized that more important than running a static analysis tool, is what is done with its output. In the worst-case scenario, a static analysis tool produces a CSV file that no one looks at. In other words, the analysis is done in vain, just for the sake of being able to say that static analysis is part of the development process. According to [10], it is important to use the right tool for the right development technologies and to have a clear process as to what to do with the results – to have a policy that states with which results is the build allowed to continue and when should the build fail. In article [11], which focused on doing DevOps in compliance-heavy organizations, the threat model of the software being built decides which parts of the code can be done by developers only and when should the design and code be reviewed by security experts. If the changes that are being made affect the threat model, a manual review by a security expert is always in order. The article also recommended that code should be peer reviewed by someone with an understanding of secure coding practices. In [12], the opinion of the DevOps practitioners is, that in DevOps, security is best built through a potent combination of automated and non-automated security practices. Automated code review was used in seven of the nine surveyed DevOps organizations, which makes it the third most popular automated security practice after automated monitoring and testing according to [12].

4.3.11 Use automated tools with tailored rules

The BSIMM activity *CR2.6 Use automated tools with tailored rules* got four mentions ([3], [9], [10], [17]). Using automated tools with tailored rules, in BSIMM's (McGraw et al. 2019) view, means customizing static analysis in a way makes it more efficient. One of the goals for doing so is reducing the number of false positives. This is an important matter, as according to [10], precisely the idea of security slowing development down (e.g., through a large number of false positives) reduces developers' enthusiasm towards security initiatives. According to BSIMM (McGraw et al. 2019), the tailored rules can be made specific to the organization's needs and this is the case in [3], where the static analysis during the development phase checks that the code is compliant with the organization's policy. Article [9] highlights the fact that static analysis is "not designed to identify open source software vulnerabilities", and in order to catch those, other security tools (such as software composition analysis tools) are needed. In [10 and in 17], the authors discussed the importance of using the right analysis methods for the used development technologies in order to gain a trustworthy outcome.

4.3.12 Use application input monitoring

The BSIMM activity *SE1.1 Use application input monitoring* is used to signify that an organization monitors the input to a system it has deployed, i.e., via a firewall or another monitoring solution. This was mentioned in four of the reviewed articles([7], [12], [14], [16]). In [7], the focus was on high-speed incident management process and in order to achieve that, incidents must be identified rapidly. Therefore, monitoring the input to an application is critical. In [12], DevOps practitioners swore by the name of automated monitoring and automated firewalls, as they increased security in DevOps. The multi-cloud security framework suggested by [14] contains a network monitoring agent, that captures packets and detects security issues and a similar approach is in use in [16] as well.

4.3.13 Identify gate locations, gather necessary artifacts

The BSIMM activity *SM1.4 Identify gate locations, gather necessary artifacts* belongs to the framework's Governance domain, and the Strategy and metrics practices specifically. This security activity deals with having checkpoints within a software development life cycle. These checkpoints can go under names like gates or milestones, and their purpose is to provide the development process with defined places where security is analyzed in a pre-defined manner. In these checkpoints, an artifact that meets the pre-defined security criteria is allowed to pass, and an artifact with less than satisfactory security is considered to need further work. This type of a security practice was mentioned in four of the reviewed articles ([6], [10], [11], [17]). In [6] a checkpoint was located before deployment: artifacts are tested based on security requirements and tested for resilience against basic attacks. Article [10] talks of the need for clear rules in DevOps development when it comes to security: if security is assessed e.g. through static analysis, the security criteria under which the build is allowed to progress needs to be established and communicated. For creating security checkpoints, the article suggested Sec, Dev and Ops to discuss together the best places where security could be assessed without slowing down the development and operations unnecessarily. In [11], the focus was on compliance driven by the threat model – if changes affecting the threat model were made, it automatically created a checkpoint beyond which development couldn't pass the code without getting the security checked by the security team. In [17], there were pre-deployment security checks that could result in a no-go result.

4.4 The four principles of DevOps

My third research question dealt with how the four principles of DevOps, *culture*, *automation*, *measurement* and *sharing* (CAMS) are represented in the reviewed works. I wanted to know first of all, if the DevOps principles were reflected in the articles and second of all, whether the researchers saw these principles as something worth mentioning in relation to security. The further I got in my research, the more I realized that the way the researchers acknowledged these principles or not seemed to reflect how they understood DevOps. As said before, some see DevOps as a heavily automated process, while others as a management style encouraging autonomy. These divergent views come to light through analysis of the interpretation of the DevOps principles. Table 10 illustrates the aspects of the four principles that emerged from the reviewed works.

TABLE 10 The four principles of DevOps in the reviewed works

CULTURE	Shared/wider responsibilities [1], [7], [8], [9]	Automation of security activities [all articles]	AUTOMATION
	Culture of collaboration [3], [12], [14]	Pipeline automation creates challenges for security [1], [2], [5], [13], [18]	
	Security and quality-oriented culture [6], [7], [8]	Pipeline automation creates opportunities for security [7], [14], [15]	
	DevOps culture needs the support of management to flourish [7], [10], [11]	Manual activities are needed alongside automated activities [10], [11], [17]	
	Culture of innovative practices [5]	Self-correcting security systems [6], [15]	
	Culture of "pride and ownership" [9]		

MEASUREMENT	Monitoring during runtime [3], [6], [7], [8], [12], [13], [15], [16] Feedback loop from runtime monitoring back to developers [14], [16] Monitoring should trigger self-correction in systems [6], [15] Logging [1] Monitoring during development [5] Tracking and measuring defects [10]	Sec, Dev and Ops should work together [3], [8], [10], [11], [12], [14], [15] Shared responsibilities create security challenges [1], [12], [18] Communication is needed for succeeding [7], [8], [10] The importance of having shared goals [9]	SHARING

Next, I will explain the aspects presented in Table 10.

Culture

When it comes to culture, the reviewed works had very different views on it. Some ([3], [12], [14]) took as a given that DevOps is all about Dev and Ops (and oftentimes also Sec) working together. Four works ([1], [7], [8], [9]) talked of the many implications of shared and wider responsibilities: how they change the employees' responsibilities and require an increase in communication. Some ([6], [7], [8]), interpreted the DevOps culture to be naturally oriented towards high quality and security, with [9] believing that the culture of DevOps gives workers freedom and a "sense of pride and ownership" that translates to motivation and high achievement. Three works ([7], [10], [11]) acknowledged that culture is a management issue and the emphasis towards security should be created at the management level from where it can be submerged into company culture. One article ([5]) interpreted that innovative practices were natural in DevOps culture and this cultural facet could be used to create more innovative security solutions as well.

Automation

All the reviewed works mentioned the automation of security practices in one way or another. None of them claimed that security should be fully automated, though. For example, in securing the development environment, the security expert has a key role. The automated practices varied a great deal, with many focusing on securing the pipeline itself or on how fast deliveries can be made more secure. Five works ([1], [2], [5], [13], [18]) talked of the new security challenges that come from the deployment pipeline: the increased insider threat, the increased attack surface as well as the complexity of the infrastructure that complicates security. Also, the issue of the pipeline becoming a hugely important business asset was raised by [5]: as the business is dependent on the pipeline, its security controls should match the risk. Some authors ([7], [14], [15]) took on a more positive outlook and shone the light on how the pipeline can increase security: the fast deployment times mean that bug fixes can be almost instantaneous, incident management is easier and feedback loops are able to provide a heightened understanding of security to developers. Two authors ([6],

[15]) took automation even a nudge further by providing insights of systems that could not only detect security issues, but also self-correct upon encountering them. Some works ([10], [11], [17]) stressed that even though many security activities can and should be automated, one needs to develop an understanding of where *not* to automate. Also, the importance of having a policy regarding security related findings is in order: if an automated code review reveals an error, there should be a protocol in place that decides whether to continue with the build or not. The data also revealed that a security policy is often needed *before* security activities are automated: the automation simply enforces the policy.

Measurement

DevOps is said to thrive on metrics and monitoring. It was no surprise, that many ([3], [6], [7], [8], [12], [13], [15], [16]), of the reviewed works highlighted monitoring during runtime and its importance towards security. A feedback loop from monitoring to developers was suggested in [14] and [16], to provide them with better understanding of security and in [7] the developers were made responsible for rapidly fixing any security issues that arose from monitoring. Logging was stressed in article [1] which emphasized that non-repudiation needs to be achieved in the development environment as well as in production to combat insider threat. Also, article [5] recommended monitoring during development, but due to a different reason: they accentuated that as the pipeline is a very important business asset, its performance and security need constant monitoring. Finally, article [10] noted that it is not only important to detect problems through monitoring, but also to track the trajectory of those defects, to make sure that detected problems are dealt with.

Sharing

Many works ([3], [8], [10], [11], [12], [14], [15]) acknowledged that the security team should be included in the DevOps work to some degree. In [8] the idea is that the security collaborates and communicates with Dev and Ops frequently, and that the organizational culture is made security-oriented. In [10], security is achieved by the security team having a service mindset which they use to aid the developers in creating secure software, taking care not to slow them down unnecessarily. In other words, security should be a shared goal, where developers acknowledge the importance of security and the security team acknowledges the importance of fast deliveries – and together they make a process which serves both needs. In [11] the security team is involved in the start of the project to create a threat model and then later step in only if changes affecting the threat model are made. Article [12] saw the inclusion of the security team in development from a practical standpoint: by including the security team in the development, more vulnerabilities can be avoided. In the spirit of sharing, communication was mentioned in three works ([7], [8], [9]) as a necessity for DevOps success. Also the importance of having shared goals was mentioned by

[9] – unnecessary tug of war is avoided when there is a shared understanding of what the organization is working towards.

5 DISCUSSION

In this chapter I will provide answers to the research questions based on the findings from the previous chapter. I will also provide further insights revealed by the analysis of the reviewed works.

5.1 RQ1: Challenges of security in DevOps

The answer to the first research question was gained through carefully reading the articles of the systematic literature review and analyzing them for any mentions on challenges with security in DevOps. These challenges were recorded in a data extraction spreadsheet and analyzed through using typification. A total of nine major challenges were found. The challenges with the most mentions were: ensuring pipeline security (mentioned in six articles), balancing security with fast deliveries (mentioned in four articles), and increased insider threat (mentioned in four articles). Other challenges had less than four mentions each. The challenges highlight DevOps' position as a relatively new development method that is still finding its foothold. The development environment is not yet stable and securing it has captured the interest of many researchers. DevOps' desire for fast deliveries and the possibility of multiple daily deployments can raise concerns by those with an interest in security. Also, the sharing responsibilities between Dev and Ops leads to a nicely flowing model of collaboration, but from a security point of view, having insiders with access to more systems can mean the possibility of insider threat.

5.2 RQ2: Security activities that are associated with DevOps

To answer the second research question of security activities that are associated with DevOps in the research literature, the selected studies were carefully read and analyzed using content analysis. References to security activities in the reviewed works were marked down in the data extraction formulation. Based on these actions, a total of 139 mentions of security activities were found from the 18 articles. These activities included 47 different activities. Figure 12 illustrates the BSIMM domains and practices that the current research literature mentions.

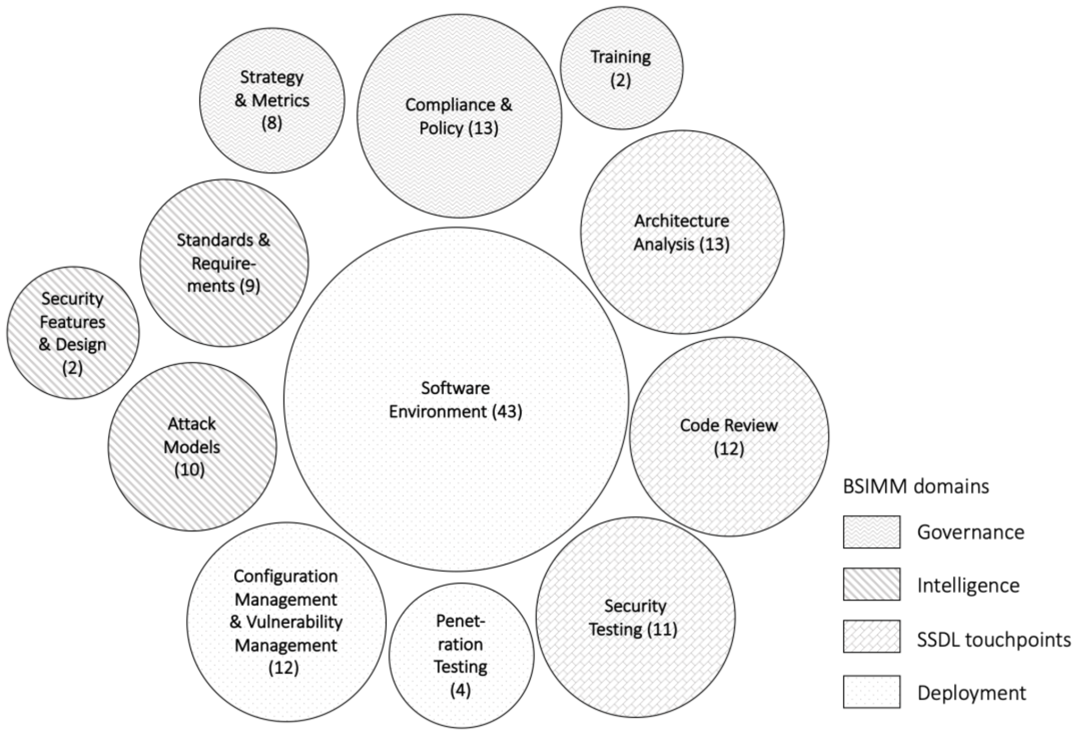

FIGURE 12 Security activity mentions per BSIMM security domain

Figure 12 shows the twelve security practice groups of the BSIMM framework. The four different domains have each gotten a distinct pattern to separate them from each other. From Figure 12 we can observe that the domain of Deployment is the domain the current literature focuses on. The security activities from the Deployment domain make up for 42 % of all security practice mentions. That means that there is significantly less research on the other three domains of BSIMM: governance, intelligence and SSDL touchpoints. Analyzing the different security activities in DevOps research and mapping them to the BSIMM framework has proven to be a concrete way to showcase in which security domains research has been done and where there is room for more research. In the BSIMM framework, security is built in throughout the whole organization, ranging from the strategical security decisions to the operational processes. Security maturity is achieved when security activities from all twelve practice categories of the BSIMM framework are carried out by the organization (McGraw et al. 2019). The current research does not reflect this diversity of security activities, as the literature heavily focuses on securing the DevOps environment and technologies, forgoing the governance and intelligence domains and portraying a lack of practical guidance of how security activities should be added to the development process.

In the reviewed articles, there was a shortage of literature discussing the placement of security activities in distinct phases of the software development life cycle (with article [10] being the only exception). DevOps literature from the industry talks of shifting security left, i.e., of incorporating security practices as early on in the product life cycle as possible. This can be done for example through code review, which was recommended in several of the reviewed articles. The findings of this thesis also spotted a trend towards shifting security right: e.g., to make security a dynamic and flexible process that happens during run time. For example, article [15] talked of software engineers needing to design systems with decision logic that afflicts the product's security behavior during runtime. E.g., the system will be able to make decisions based on its environment and the challenges it perceives to its security. Article [7] on the other hand, shifted security even further to the right, and emphasized the benefits and opportunities provided by efficient incident management practices. The reasoning behind this was that the dynamicity of DevOps allows for faster incident response and fixing problems as soon as they appear. Furthermore, according to [7], all vulnerabilities can never be prevented, so efficient detection response mechanisms are a good insurance policy. In other words, the speed of the pipeline and continuous deliveries work to the organization's advantage also in times of security incidents: issues found by Ops are fed to the Dev, which can fix them immediately and deploy them instantly as well. Needless to say, such a well-oiled incident response process does demand a lot from the organization's capabilities, as well as the communication and collaboration culture. That being said, an organization that has nailed the DevOps culture of shared responsibilities, could use it to leverage their incident response game as well. The authors of [3] agree with this: "Investing resources on addressing some security vulnerabilities at development time may prove futile, either because at runtime a new vulnerability may appear under an unforeseen usage scenario or because we failed to properly rank vulnerabilities and we addressed the wrong one." Considering moving security to the left or right, it is still wise to remember the words of wisdom from BSIMM creators (McGraw et al. 2019): "DevOps is a cultural change that can't be solved with any amount of incremental procedural alteration." Therefore, in addition to the technical security activities, a closer look at the culture and the other three DevOps principles is in order.

5.3 RQ3: CAMS principles reflected in DevOps research

To answer the final research question, the articles were read and content analysis was used to analyze the references to any of the four principles of DevOps, *culture*, *automation*, *measurement* and *sharing* were recorded in the data extraction spreadsheet. The reviewed literature contained mentions of all of the four principles of DevOps. However, the wide variety of references made to these principles seem to indicate that not all researchers understand DevOps in the same way. This finding aligns with a previous systematic literature review by Jabbari et al. (2016), who found that DevOps is understood by different authors to signify different things. In Jabbari et al.'s (2016) study, 31 % of the articles interpreted the core of DevOps to be collaboration and communication, and 24 % considered its focus to be on delivering software using automated delivery pipelines. The same great divide was also seen in my research. As stated in an interview in [10]:

> "…many companies have no clear definition of what it means to do DevOps. A lot of organizations that are doing a bit of continuous integration or static analysis think they're doing DevOps. They don't realize that the whole thing – whether you're talking about continuous integration or delivery or deployment – is completely different from what DevOps actually focuses on. The culture, the role, the emphasis on responsiveness are all lacking. The operations team doesn't even talk to the development team. But people just love the word 'DevOps'. (Mansfield-Devine 2018, p. 15)

A total of seven articles ([1], [3], [7], [8], [10], [12], [14]) mentioned all of the four principles. Their viewpoint on these principles varied, though. For instance, [1] perceived the culture of DevOps to mean shared responsibilities, automation to create security challenges, referred to measurement in increased need for logging and sharing as another facilitator of security challenges through increased insider threat. Then again, [12] talked of culture as one of collaboration, automation as a security enhancing technology, measurement as the importance of runtime monitoring and sharing as increased cooperation between Dev, Ops and Sec. The lack of clear definition of the practice of DevOps became evident in the research: DevOps was seen by different authors to signify a culture of quality-orientation, a culture of innovation, technology-only, developer self-service and customer value – all under the name of DevOps.

An important discovery from the research is that an organization wanting to undertake DevOps first needs to establish what it actually means by DevOps. How does the organization interpret the four DevOps principles of culture, automation, measurement and sharing? All four aspects need management support and organization in order to be executed in a good manner. Security-oriented culture cannot be born without a clear orientation towards it. The same goes for emphasis on quality. Without clear direction, the risk with DevOps is doing fast deliveries without paying attention to security. Security needs to be baked into the organizational culture in order to become a mindset for the DevOps team. As a matter of fact, a recent Gartner (2019) survey revealed, that neglecting to see DevOps as a cultural phenomenon highly affected the ability to scale DevOps and that organizational aspects are the biggest cause of DevOps adoption failure.

When it comes to automation, DevOps usually means a high adaptation rate of automation technologies. These technologies demand the organization to pay attention to any new security issues that they might bring along: securing the pipeline is important, as said before, and paying attention to the possibility of insider threat. Automation brings many possibilities to security activities, but manual activities still are needed. Automating security also does not eliminate the need for security professionals, as many automated tools are configured based on security policies (which are created by the security professionals) and the output of those tools oftentimes requires manual work and security expertise. The reviewed articles also stressed the importance of choosing the right tools, if security is left to the hands of automation. As a cautionary tale, one article [10] talked of a company that was proud of their use of a static analysis tool – yet failing to understand that the tool was not equipped for their environment and was not catching the coding errors at all. A second common mistake mentioned in the same article [10], is using a tool to catch errors, but not having a process that demands a developer to fix the problematic issues.

Measuring is another DevOps principle and in the sense of security, it is usually achieved through monitoring. In monitoring, the security policies and requirements set the tone for what is being monitored and how that information is processed further. As such, having a solid foundation for how security is managed and valued in the organization is necessary for measuring the right things. Eight of the reviewed articles mentioned monitoring the software at runtime to catch security issues, and one during development, to ensure correct performance. Logging is needed for obvious security purposes but also to make sure non-repudiation is achieved within the deployment pipeline as well. The worst-case scenario is deploying harmful code and not having any idea how it got there, by whom and when.

Sharing is a big component of DevOps, as even the name of the development method points to the merging of development with operations. What emerged from the data is yet again the need to clearly establish roles in the organization: who does what and how is information between these stakeholders shared. One article [9] highlighted the need for a common goal and I think from analyzing the research this becomes apparent. If an organization wants to create a culture where tasks are shared and people work together, those people need to know what values they are working towards. If the importance of security is not established, implementing security activities easily becomes a tug of war between the security team and the developers and operations. If the importance of security is not communicated to developers, the developers will feel that security activities are unnecessary and only slow them down. If, on the other hand, the importance of security is established, the Sec, Dev and Ops can work together to strike a balance that fits the organization's values.

As many industry publications (e.g., Puppet 2019, Gartner 2019) talk of the importance of culture and understanding DevOps as a cultural phenomenon, the CAMS principles should be taken into consideration by organizations wanting to implement DevOps. Through analyzing the articles for their mentions of the DevOps principles in relation to security, I have fused together their messages to palatable practical advice that is presented in Table 11. Through embracing all four DevOps principles and weaving security into each one, security can be built into the core of DevOps.

TABLE 11 Advice on how to build security into the core of DevOps

DevOps principle	*Practical advice*
Culture	• Develop a security-oriented culture with strong security awareness. • Create a culture with a sense of shared responsibilities, where Dev, Ops and security work together for a common goal. • Prioritize fixing of security defects.
Automation	• Automate security as much as possible but make sure the chosen tools are applicable to the environment and the technology. • Ensure pipeline security. • Acknowledge that automation enhances manual security activities, it does not replace them.
Measurement	• Develop security metrics. • Measure and monitor security constantly. • Provide a feedback loop to developers and operations on security issues and intelligence.
Sharing	• Share knowledge on security principles. • Develop incident management and involve developers in fixing security problems.

5.4 Limitations, reliability and validity

A piece of research is considered reliable, if another researcher conducting the same exact research would get the same exact results. The reliability of an SLR comes from making the systematic review process as systematic and transparent as possible. In Chapter 3 I have presented my search strategy as well as the search criteria and have done my best to adhere to those policies I have presented. However, as a solo researcher interpreting others' research from my own point of view, there is the possibility of another human being interpreting those same research articles in another fashion. Thus, I have done my best to conduct the research as reliably as possible, but due to the qualitative nature of the study and the research process needing a heavy amount of interpretation, I assume another researcher conducting the same research might have slightly differing results.

The validity of the results is influenced by the study selection process as well as the interpretation of the selected studies. In applying the search criteria, I did my best to find all studies that were relevant to the research questions and not exclude any works that would contribute to the research. To make sure my searches were complete, I performed additional manual searches into the selected databases to make sure my search strings had been successful. These manual searches proved that the searches using the search strings had been completed properly and no relevant research from the selected databases had been lost in the process.

The limited number of research available on the research subject has naturally influenced the answers to the research questions. However, as the subject of this Thesis has been to chart the current status of secure DevOps research, the number of studies only proves that there is room for more research on the subject.

5.5 Topics for future research

This literature on the current state of research on DevOps security practices has shown that there is much room for more research on the subject. The current research focuses on securing the technologies typically used in DevOps development, and very little material is available on how more traditional security practices should be affixed to the DevOps development life cycles. I echo the words of Tuma et al. (2018) who said that there is "immaturity of research in integrating security into DevOps". More research on practical aspects of adding security to DevOps is needed as the industry perceives the trend of DevSecOps to grow in the future. According to Deloitte (2019): "As the DevSecOps trend gains momentum, more companies will likely make threat modeling, risk assessment, and security-task automation foundational components of product development initiatives, from ideation to iteration to launch to operations." To understand how these security activities should be incorporated into the development life cycle, more academic research is needed. The focus should not only be on automation, though, but on the integration and interaction of the automated and non-automated activities. How to incorporate the slower and non-automated security practices into rapid DevOps development, was a question that was not answered by the reviewed works, and as such it remains a good question to answer by future research.

6 CONCLUSION

In this Thesis a systematic literature review was conducted, with the goal of understanding the current state of research of security in the context of the DevOps development method. The research answered three research questions:

- RQ1: What are the challenges of security in DevOps as reported by the authors of primary studies?
- RQ2: Which security activities are associated with DevOps in the literature?
- RQ3: How are the CAMS (culture, automation, measurement and sharing) principles reflected in secure DevOps research?

In the systematic literature review a search from four electronic databases was done in April 2019 using a search string developed for the purpose of finding relevant research. The initial search results from the databases yielded 292 results. After two rounds of filtering the papers relevant to the research questions, the number of selected papers was narrowed down to 16. Backward and forward snowballing was conducted to find any potentially relevant works. Through this step in the research process, two more articles were found. The final selection of articles for this research thus became a set of 18 articles. The reviewed articles were analyzed using typification and content analysis.

The results of the research revealed a total of nine challenges with security in the context of DevOps. The biggest challenge came from the technologies used in DevOps development, with the deployment pipeline in particular garnering a lot of mentions. Also, DevOps' goal of rapid deployments was seen as a security challenge by four articles. Striking the balance between security and fast deliveries was a cause of concern for the authors. The reviewed works recognized the importance of having a shared goal of security, set by the management level of the company, as the guiding post directing development towards security-inducing activities. Another challenge mentioned by four authors was the potential increase of insider threat. If the Dev can do Ops' work and vice versa, many employees suddenly have a much wider access to the organization's systems than before. Thus, security activities that especially fit the challenges of the automated pipeline, the faster delivery cycles and increased insider threat are needed to make the DevOps development process more secure.

To understand where the current security research is focused, the security activities mentioned in the reviewed works were mapped to the BSIMM framework. The results showed that the security activities mentioned by the reviewed articles focused heavily on securing the DevOps technologies and environments. Thus, in current research, security activities belonging to the domains of governance, intelligence and software development have not gotten much attention and offer many possibilities for future research.

The research also seeked to understand the authors' views on what DevOps is. This was done by analyzing the articles for mentions of the four principles of DevOps, which are culture, automation, measurement and sharing. The research found that the reviewed articles interpret DevOps to signify various things and do not have a single definition of its essence. According to the DevOps principles, culture and sharing are counted amongst the development method's key issues, yet the security activities reflecting these principles are sorely missing in the current research (e.g., creating a security culture, security awareness training, clear security roles in organization). According to the principles, in DevSecOps, the focus should very much be on creating a flourishing security culture that facilitates communication and sharing between the three key players: Dev, Sec and Ops. To achieve a better picture of this, more research in the less technology-oriented side of DevSecOps would be needed.

REFERENCES

Agile Manifesto. (2001). Agile Manifesto. Accessed online 15.5.2019 at https://agilemanifesto.org/.

Ahmanavand, M., Pretschner, A., Ball, K. & Eyring, D. (2018). Integrity Protection Against Insiders in Microservice-Based Infrastructures: From Threats to a Security Framework. In: Mazzara M., Ober I., Salaün G. (eds) *Software Technologies: Applications and Foundations, STAF 2018*. Lecture Notes in Computer Science, vol 11176.

Baskerville, R. (1993). Information systems security design methods: Implications for information systems development. *ACM Computing Surveys (CSUR), 25(4)*, pp. 375–414.

Bass, L., Holz, R., Rimba, P., Tran, A. B. & Zhu,L. (2015). Securing a Deployment Pipeline. *2015 IEEE/ACM 3rd International Workshop on Release Engineering (4–7)*. Florence, Italy, 2015.

Beigi-Mohammadi, N., Litoiu, M., Emami-Taba, M., Tahvildari, L., Fokaefs, M., Merlo, E. & Onut, I. V. (2018). A DevOps framework for quality-driven self-protection in web software systems. In Andrew Jaramillo and Guy-Vincent Jourdan (Eds.), *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering, CASCON '18 (270–274)*. IBM Corp., Riverton, NJ, USA.

Charles Babbage Institute. (2003). An interview with Donn B. Parker. Conducted by Jeffrey R. Yost on 14 May 2003 in Los Altos, California. Accessed online 21.2.2019 at: https://conservancy.umn.edu/bitstream/handle/11299/107592/oh347dp.pdf?sequence=1&isAllowed=y

Cois, C. A., Yankel, J. & Connell, A. (2014). Modern DevOps: Optimizing Software Development Through Effective System Interactions. *2014 IEEE International Professional Communication Conference, IPCC (1–7)*. Pittsburgh, PA, USA, 2014.

Curphey, M. (2019). Fail Fast: How Shifting Security Left Speeds Development. A DevOps.com blog, 8.2.2019. Accessed online 9.7.2019 at https://devops.com/fail-fast-how-shifting-security-left-speeds-development/.

Deloitte. (2019). *Tech Trends 2019. Beyond the digital frontier. Deloitte Insights 10th Anniversary Edition*. Accessed online 25.5.2019 at https://www2.deloitte.com/content/dam/insights/us/articles/Tech-Trends-2019/DI_TechTrends2019.pdf.

Dhillon, G. & Backhouse, J. (2001). Current directions in IS security research: Towards socio - organizational perspectives. *Information Systems Journal, 11(2),* pp. 127–153.

Diekmann, C., Naab, J., Korsten, A. & Carle, G. (2019). Agile Network Access Control in the Container Age. *IEEE Transactions on Network and Service Management, 16(1),* pp. 41-55.

Düllmann, T. F., Paule, C. and van Hoorn, A. (2018). Exploiting devops practices for dependable and secure continuous delivery pipelines. *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering, RCoSE '18 (27–30).* ACM, New York, NY, USA.

Felderer, M. & Fourneret, E. (2015). A systematic classification of security regression testing approaches. *International Journal on Software Tools for Technology Transfer, 17(3),* pp. 305–319.

Ferry, N., Solberg, A., Song, H., Lavirotte, S., Tigli, J., Winter, T., Muntés-Mulero, V., Metzger, A., Rios Velasco, E. & Aguirre, A. C. (2019). ENACT: Development, Operation, and Quality Assurance of Trustworthy Smart IoT Systems. In: Bruel JM., Mazzara M., Meyer B. (eds), *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (112-127).* Chateau de Villebrumier, France, March 5–6, 2018.

Gartner. (2019). The Secret to DevOps Success. April 11th 2019. Accessed online 22.5.2019 at https://www.gartner.com/smarterwithgartner/the-secret-to-devops-success/.

Google Trends. (2019). DevOps and DevSecOps comparison in the time period 1.1.2016-31.5.2019. Accessed online 9.7.2019 at https://trends.google.com/trends/explore?date=2016-01-01%202019-05-31&geo=US&q=devops,devsecops.

Hahn, D. (2016). How Netflix Thinks of DevOps. Accessed online 9.5.2019 at https://www.youtube.com/watch?v=UTKIT6STSVM.

Humble, J. & Molesky, J. (2011). Why Enterprises Must Adopt DevOps to Enable Continuous Delivery. *The Journal of Information Technology Management 24(8), pp. 6–12*. Accessed online 9.5.2019 at https://www.cutter.com/sites/default/files/itjournal/fulltext/2011/08/itj1108.pdf.

Jaatun, M. G., Cruzes, D. S., Bernsmed, K., Tondel, I. A. & Rostad, L. (2015). Software Security Maturity in Public Organizations. In: Lopez J., Mitchell C. (eds), *Information Security (ISC 2015)*. Lecture Notes in Computer Science, vol 9290.

Jaatun, M.G., Cruzes, D. S. and Luna, J. (2017). DevOps for Better Software Security in the Cloud Invited Paper. In: *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES '17)*. Reggio Calabria, Italy, August 29–September 1 2017.

Jaatun, M. G. (2018). Software Security Activities that Support Incident Management in Secure DevOps. In: *Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018*. Berlin, Germany, 27-30 August 2018.

Jabbari, R., Bin Ali, N., Petersen, K. & Tanveer, B. (2016). What is DevOps? A Systematic Mapping Study on Definitions and Practices. In: *Proceedings of the XP2016 (XP '16 Workshops)*. ACM, New York, NY, USA, Article 12, 11 pages.

Khan, A. (2018). DevOps Culture at Amazon. Accessed online 9.5.2019 at https://www.youtube.com/watch?v=mBU3AJ3j1rg.

Kitchenham, B. & Brereton, P. (2013). A systematic review of systematic review process research in software engineering. *Information and Software Technology, 55(12)*, pp. 2049-2075.

Kitchenham, B. A., Budgen, D. & Brereton, P. (2016). *Evidence-based software engineering and systematic reviews*. Boca Raton: CRC Press.

Lietz, S. (2016). Shifting Security to the Left. A DevSecOps blog, June 5 2016. Accessed online 9.7.2019 at https://www.devsecops.org/blog/2016/5/20/-security.

Loughman, K. (2019). The DevOps Model: Rapid Software Delivery and Incident Management. Accessed online 24.11.2019 at https://victorops.com/blog/the-devops-model-rapid-software-delivery-and-incident-management.

MacDonald, N. & Head, I. (2017). 10 Things to get Right for Successful DevSecOps. Gartner Research, 3 October 2017. Accessed online 9.7.2019 at https://emtemp.gcom.cloud/ngw/eventassets/en/conferences/lsce14/documents/gartner-io-cloud-uk-research-note-successful-devsecops-2018.pdf.

Mackey, T. (2018). Building open source security into agile application builds. *Network Security, 2018(4)*, pp. 5–8.

Mansfield-Devine, S. (2018). DevOps: Finding Room for Security. *Network Security, 2018(7)*, pp. 15–20.

McGraw, G. (2005). Bridging the Gap between Software Development and Information Security. *IEEE Security & Privacy, 3(5)*, pp. 75–79.

McGraw, G., Migues, S. & West, J. (2019). *BSIMM9*. Downloaded 20.5.2019 from https://www.bsimm.com/.

Michener, J. R. & Clager, A. T. (2016). Mitigating an Oxymoron: Compliance in a DevOps environment. *2016 IEEE 40th Annual Computer Software and Applications Conference, COMPSAC (396–398)*. Atlanta, GA, 2016.

Mohammed, N. M., Niazi, M., Alshayeb, M. & Mahmood, S. (2017). Exploring software security approaches in software development lifecycle: A systematic mapping study. *Computer Standards & Interfaces, 50(2017),* pp. 107–115.

Mohan, V. and ben Othmane, L. (2016). SecDevOps: Is it a marketing buzzword? mapping research on security in devops. In: *Proceedings of the 11th International Conference on Availability, Reliability and Security, ARES (542–547)*. Salzburg, Austria, Sep. 2016.

Myrbakken, H. & Colomo-Palacios, R. (2017). DevSecOps: A Multivocal Literature Review. *International Conference on Software Process Improvement and Capability Determination, (17–29).* September 2017.

Puppet. (2019). *2018 State of DevOps Report*. Accessed online 9.5.2019 at https://puppet.com/resources/whitepaper/state-of-devops-report.

Raj, A., Kumar, A., Pai, S. J. & Gopal, A. (2016). Enhancing Security of Docker using Linux Hardening Techniques. *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology, iCATccT, (94–99)*. Bangalore, India, 2016.

Rios, E., Iturbe, E., Mallouli, W. & Rak, M. (2017). Dynamic security assurance in multi-cloud DevOps. *2017 IEEE Conference on Communications and Network Security, CNS (467–475).* Las Vegas, NV, USA.

Schoenen, S., Mann, Z. & Metzger, A. (2018). Using Risk Patterns to Identify Violations of Data Protection Policies in Cloud Systems. In: *Service-Oriented Computing, ICSOC 2017 (296–307).*

Shahin, M., Babar, M. A. & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access, 2017(5)*, pp. 3909–3536.

Souza, E., Moreira, A. & Goulao, M. (2019). Deriving architectural models from requirements specifications: A systematic mapping study. *Information and Software Technology, 2019(109)*, pp. 26–39.

Stroud, R. (2017). 2018: The Year of the Enterprise DevOps. Forrester. October 17, 2017. Accessed online 20.6.2019 at https://go.forrester.com/blogs/2018-the-year-of-enterprise-devops/.

Tamburri, D. A., Di Nucci, D., Di Giacomo, L. & Palomba, F. (2019). Omniscient DevOps analytics. *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pp. 48–59.

Thanh, T. Q., Covaci, S., Magedanz, T., Gouvas, P. & Zafeiropoulos, A. (2016). Embedding Security and Pricacy into the Development and Operation of Cloud Applications and Services. *2016 17th International Telecommunications Network Strategy and Planning Symposium, Networks (31–36)*. Montreal, QC, Canada.

Torkura, K.A., Sukmana, M. I. H., Cheng, F. & Meinel, C. (2018). CAVAS: Neutralizing Application and Container Security Vulnerabilities in the Cloud Native Era. In: Beyah R., Chang B., Li Y., Zhu S. (eds), *Security and Privacy in Communication Networks, SecureComm 2018 (470–490)*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 254.

Tuma, K., Calikli, G. & Scandariato, R. (2018). Threat analysis of software systems: A systematic literature review. *The Journal of Systems & Software, 144(2018)*, pp. 275–294.

Ullah, F., Adam J. R., Shahin, M, Zahedi, M. & Babar, M. A. (2017). Security Support in Continuous Deployment Pipeline. *International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE*.

Ur Rahman, A. A. and Williams, L. (2016). Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices. *2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery, CSED (70–76)*. Austin, TX, USA.

Williams, L., McGraw, G. & Migues, S. (2018). Engineering Security Vulnerability Prevention, Detection, and Response. *IEEE Software, 35(5)*, pp. 76–80.

Willis, J. (2010). What DevOps Means To Me. Chef.io 16 July 2010. Accessed online 10.5.2019 at https://blog.chef.io/2010/07/16/what-devops-means-to-me.

Wohlin, C. (2014). Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. *EASE '14 Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, (1–10)*.