List of Security Concerns within Continuous Software Evolution

SIMONE PERSSON
Abstract

The amount of data being collected is increasing astronomically. Hence questions about privacy and data security are becoming more important than ever. A fast-changing culture is also reflected in the demands and requirements placed on software systems. Products and services need to evolve with the demands and feedback from customers to stay relevant on the market. Working methods and technologies have been refined to afford updating software continuously. However, rapidly changing software cause concern for the quality and level of security in the release.

This thesis is a comprehensive literature study, reviewing the challenges of ensuring secure practises for continuously evolving software. The problem solved by the thesis is lack of an overall picture of the security concerns during continuous evolution. The findings are summarised in a checklist of areas of concern for security when maintaining and updating systems with continuous practises in cloud environments.

This study shows that ensuring security, while delivering continuous releases, is a daunting task. It requires close collaboration between teams handling different aspects of software. This, in turn, entails a widening of competences to include knowledge about the work of other departments. It is concluded that personnel with this wide range of skill will be hard to acquire.

Keywords
DevOps
CD/CI - Continuous deployment / Continuous integration
Cloud Computing
Maintenance
Abstract


Studien visar att leverera säkra lösningar kontinuerligt är en svår uppgift. Det kräver nära samarbete mellan team som sköter olika delar av mjukvaruutveckling. Detta fordrar vida kompetenser som inkluderar förståelse av varandras arbete. Att finna personal med tillräckligt vida kompetenser uppskattas vara problematiskt.

Nyckelord
Devops
CD/CI - Continuous deployment / Continuous integration
Cloud Computing
Maintenance
Table of Contents

Chapter 1  Introduction ........................................................................................................1
  1.1  Background ............................................................................................................. 1
  1.2  Problem .................................................................................................................. 2
  1.3  Purpose .................................................................................................................... 2
  1.4  Goal ........................................................................................................................ 2
    1.4.1  Social benefit, sustainability and ethics .......................................................... 2
  1.5  Methods ................................................................................................................... 3
  1.6  Scope and limitations .............................................................................................. 4
  1.7  Disposition .............................................................................................................. 4

Chapter 2  Continuous evolution and challenges to security ..................................... 5
  2.1  Flexible code through continuous releases .......................................................... 5
    2.1.1  DevOps ............................................................................................................. 5
    2.1.2  Continuous Integration and Continuous Delivery .......................................... 6
    2.1.3  Automation ....................................................................................................... 6
    2.1.4  Monitoring ...................................................................................................... 6
    2.1.5  Technical challenges for secure continuous releases .................................... 6
      2.1.5.1  Shorter release cycles raise concern for quality ........................................ 6
      2.1.5.2  The irony of automation ............................................................................ 7
      2.1.5.3  Integrating security in the delivery pipeline ............................................. 7
    2.1.6  Organisational challenges for secure continuous releases ......................... 8
      2.1.6.1  Varied interpretations of the definition of DevOps ................................ 8
      2.1.6.2  Obstacles for effective collaboration ....................................................... 8
      2.1.6.3  Assessing security without the full picture ............................................. 9
  2.2  Flexible infrastructure through cloud computing ............................................. 9
    2.2.1  Definition of cloud computing ....................................................................... 9
    2.2.2  Multi-cloud strategy ....................................................................................... 10
    2.2.3  Microservices ................................................................................................. 10
    2.2.4  Containers ...................................................................................................... 10
    2.2.5  Top general threats to security within cloud computing ............................ 11
    2.2.6  Threats associated with architectural principles often used with continuous evolution .................................................................................................................. 11
  2.3  Maintenance and evolution .................................................................................... 13
    2.3.1  Classic maintenance process model .............................................................. 13
    2.3.2  Maintenance with continuous evolution ....................................................... 14
    2.3.3  Continuous Maintenance of tools and artefacts .......................................... 15
  2.4  Feedback to the study .......................................................................................... 15

Chapter 3  Research method ....................................................................................... 17
  3.1  Research style ....................................................................................................... 17
  3.2  Research Strategy ................................................................................................. 18
    3.2.1  Research instrument and data collection process ....................................... 18
    3.2.2  Inclusion and exclusion criteria .................................................................... 18
      3.2.2.1  Motivation behind choice of research criteria ...................................... 19
    3.2.3  Research phases ........................................................................................... 19
      3.2.3.1  Exploration phase .................................................................................... 20
      3.2.3.2  Interpretation phase ............................................................................... 21
      3.2.3.3  Communication phase .......................................................................... 21
  3.3  Method discussion ............................................................................................... 21
Chapter 4  List of security concerns within software evolution .......... 23
  4.1  Creating a checklist of security concerns ...................................... 23
      4.1.1  Identifying problem area ......................................................... 23
      4.1.2  Gathering and structuring information ....................................... 23
      4.1.3  Formulating the checklist ....................................................... 23
  4.2  Result ............................................................................................ 24
      4.2.1  Developers need to be well-educated in security ......................... 24
      4.2.2  Security specialists need to collaborate closely with other teams ...... 25
      4.2.3  Threats and changing conditions need to be discovered early .......... 26
      4.2.4  Security need to be integrated in the deployment pipeline ............. 26
      4.2.5  Maintenance is needed of the tools in the deployment pipeline ......... 27

Chapter 5  Conclusion ............................................................................. 29
  5.1  Research question revisited ............................................................. 29
  5.2  Validity, dependability, confirmability and transferability .................. 30
  5.3  Suggestions for future research ........................................................ 31

References .................................................................................................. 32

Appendix A ................................................................................................. 1
  1.  Personal information and the need to protect it ..................................... 1
      1.2  The right to privacy ......................................................................... 1
      1.3  Data Law of 1973 ........................................................................... 1
      1.4  Organization for Economic Cooperation and Development Privacy
           Guidelines .......................................................................................... 2
      1.5  Swedish Data Protection Act ............................................................. 2
      1.6  General Data Protection Regulation .................................................. 3

Appendix B ................................................................................................. 1
  1.  DevOps .................................................................................................. 1
      1.2  Continuous Integration ..................................................................... 1
      1.3  Continuous Delivery ....................................................................... 1
      1.4  Automation ...................................................................................... 2

Appendix C ................................................................................................. 1
  1.  Cloud Computing .................................................................................. 1
      1.1  Cloud Security Alliance top 12 threats ............................................. 1

Appendix D ................................................................................................. 1
  1.  Software Maintenance and Evolution .................................................... 1
Chapter 1 Introduction

“The Only Thing That Is Constant Is Change” (Heraclitus, n.d.). The quote by the Greek philosopher Heraclitus can be applied to all of life but seems more relevant than ever for describing the state of technology. Companies must deliver products that adapt to the need of the customers to stay relevant on the market (Hüttermann, 2012). This means that requirements constantly change for both functionality and computing resources.

We also live in an age where the need for data security is more important than ever. The amount of data generated annually is increasing with an incredible speed (Reinsel, Gantz, & Rydning, 2017). Much of the data contains information pertaining to individuals, such as their movements, habits, and all forms of personal information. Appropriate measures for protecting this data must be taken by all companies/authorities/organisations that in some way process information about individuals. Also, strict data security regulations are enforced to ensure this (Datainspektionen, n.d.-b). Ensuring security for products and services that continuously change to fit the constantly shifting requirements and demands of the market and customers is difficult (Bird, 2016).

1.1 Background

We exist in an age defined by rapid change (Sharma, 2017). The average lifespan of a company in the top 500 most widely held stocks on the New York Stock Exchange and NASDAQ is now under 20 years (Klerk et al., 2017). In 1958 the average lifespan was 60 years and it has gradually declined since then.

Many of the fastest growing companies in the world succeed by embracing innovation and finding new ways to offer customers what they want and when they want it (Sharma, 2017). The transportation company Uber (“Uber,” 2018) has in recent years disrupted the taxi industry by creating a platform for customers and registered drivers to connect and offer rides in the most efficient way possible (Blystone, 2015). The success lies in the ability to adhere to the ever-changing needs, wishes and expectations of the customers.

Products that continuously evolve are run by continuously evolving software (Hüttermann, 2012). Methods and technologies have been refined over the years to enable software to be updated in smaller iterations and in a higher pace than ever before, enabling the evolution to be driven by the feedback from the customers and people using the system (Bird, 2016). Cloud computing offers flexibility in infrastructure and computing resources, such as servers and storage, to support the ever-changing software (Linthicum, 2015).

The development culture allowing continuous evolution is successful in many ways, but concerns still remain for ensuring security in this context (Bird,
Assessing security becomes problematic when the product or service is constantly changing.

1.2 Problem

The problem is that an overall picture of the security concerns during continuous software evolution is lacking. Fast release cycles and complicated cloud environments places unique challenges on security (Bird, 2016; Rios, Iturbe, Mallouli, & Rak, 2017), and higher demands on information security puts pressure on companies to stay compliant with current security regulations (Datainspektionen, 2017). Organisations that have adopted a development culture with continuously evolving software and cloud computing, need to possess an overview of the challenges to security that can lead to non-compliance. The research question is as follows:

What areas of concern for security exist when maintaining and updating systems with continuous methods in cloud environments?

1.3 Purpose

The purpose of this thesis is to provide an overall picture of the security concerns associated with a development culture with continuously evolving software in cloud computing environments. The work will result in a list of security concerns for continuous evolution.

1.4 Goal

The goal of the study is to offer knowledge of the security challenges in a continuously evolving software in a cloud computing environment and create a foundation for future security assurance models in continuous evolution.

1.4.1 Social benefit, sustainability and ethics

The list created as the result of this study will be valuable for all organisations looking to get an understanding of what challenges are to be expected when delivering fast releases in cloud environments whilst staying compliant with current data security regulations. Ensuring secure practises can help mitigate and lessen the impact of malicious attacks. Successful attacks where incriminating data is stolen could have devastating consequences for the companies and individuals affected by the result of the disclosed information.

The General Data Protection Regulation (GDPR) is a common directive within the EU for handling of personal information (European Parliament, 2016). GDPR was enforced in May 2018. Organisations found to be non-compliant with the requirements of GDPR by a regulatory authority may be imposed an administrative fine of 20 million euros or 4 percent of the annual turnover (Datainspektionen, 2017). A substantial sum putting pressure on companies to make sure that the regulations are being followed. Guidelines for how to maintain systems in a way that ensures continuous compliance with the
requirements of GDPR will therefore be of value. More information on personal information and the laws protecting is to be found in Appendix A.

Much points to the issue of data protection being increasingly important over time. The market research firm IDC ("IDC," 2018) forecasts that the amount of data created annually by 2025 will reach 163 zettabytes, with one zettabyte being one trillion gigabytes, and the current annual creation rate being 16.3 zettabytes (Reinsel et al., 2017). They further estimate that 90 percent of this amount will need some level of security while only half of it will have sufficient security measures in place. Models for security assurance within continuous software evolution will be important to meet these challenges, and an overview of the security concerns within continuous software evolution could be used as a starting point for creating these.

Ethical considerations relevant to the research concern principles of openness for aims and objectives of the study, avoiding bias to ensure objectivity, and taking appropriate measures to avoid misleading information (Bryman & Bell, 2007). A factual account is given of the purpose, goal, and limitations of the work to avoid any deception about the objectives of the study. The review of literary sources as data collection method introduces the risk of bias since the interpretation of the text may be affected by the personal views of the researcher. Careful consideration has therefore been taken to represent the views expressed by the authors as accurately as possible, as well as being thorough in the process of distinguishing these from personal analysis of the findings. The risk of providing misleading information has been mitigated by basing the result on relevant and peer reviewed sources and representing the findings as truthfully as possible.

1.5 Methods

This study follows a qualitative research process, meaning that the focus was put on exploring and understanding the challenges to security within continuously evolving software (Håkansson, 2013). The research method was descriptive research with an inductive approach, implying that sources discussing aspects of data security in the different contexts of the study were looked at as they are, and from this finding new connections and common characteristics.

The chosen research strategy was Comprehensive Literature Review as described by Onwuegbuzie and Frels (2016) (Onwuegbuzie & Frels, 2016). The Comprehensive Literature Review represents both a tool for data collection and analysis. Reviewing relevant literature is an effective method to understand a fragmented and complicated topic as well as identifying possible gaps in available research (Mylärniemi, 2015). The result of the study was then used to create the final list of security concerns for continuous evolution. The research method is discussed further in Chapter 3.
1.6 Scope and limitations

The study is a literature review, meaning that the result is based on- and limited to the literary sources acquired on the relevant subjects. The area of study is limited to methods and technologies associated with rapid evolution of software and what challenges can be found to affect the overall security. The term evolution pertains to the software lifecycle phase commonly referred to as maintenance and evolution (Kajko-Mattsson, 2001). This means that security in the context of the remaining two major phases in the software life cycle, Software Development and Retirement, are not covered by this thesis.

It should be taken into consideration that some of the subjects have been less researched overall. Cloud computing services and platforms are evolving quickly and research on relevant threats to security may struggle to keep up.

The list created as the result of this study consists of areas of concern for security summarised from the findings in the researched literature. The points on the list do not make up a concrete set of guidelines or a model for security assurance within continuous evolution. They are more general in nature and rather provide a foundation of knowledge for what parts of the development culture may prove problematic to ensure secure practises.

The area of the study is extensive and with time being a limiting factor, it must be assumed that further research would uncover more findings than the conclusions drawn by this thesis. The list should therefore not be regarded as complete, but rather a groundwork for future research on the subject.

1.7 Disposition

The remainder of the report is structured in the following way:

- Chapter 2 establishes an extended background of the development culture for continuously evolving software and the challenges to ensure security in this context.

- Chapter 3 describes the scientific methods used in the study. Methods for data collection and analysis are presented.

- Chapter 4 explains how the findings from the literary review were translated into the finished list of security concerns within continuous software evolution and presents the result.

- Chapter 5 presents the conclusions drawn from the study. The work is reviewed from the aspects of validity, dependability, confirmability, and transferability. Suggestions for further studies on the subject are made.
Chapter 2 Continuous evolution and challenges to security

This chapter provides a theoretical framework of the technologies and methods enabling continuously evolving software and the concerns that exist for security in each context. Section 2.1 presents practises and concepts central to enabling continuous evolution. Section 2.2 explains the concept of cloud computing along with the architectural principles often used to enable continuously evolving software. Section 2.3 covers the concept of software maintenance and evolution. Section 2.4 discusses how the presented theory supports the thesis and discusses the related works.

2.1 Flexible code through continuous releases

This section presents technologies and methods for software development that enables short iterations with fast releases and what challenges they bring for security. Subsections 2.1.1-2.1.4 establishes necessary definitions and subsections 2.1.5-2.1.6 presents security challenges.

2.1.1 DevOps

As technology is advancing in an increasing pace, many organisations have opted for development cultures embracing small updates done often (Hüttermann, 2012). The Agile Manifesto was released in 2001 (Beck et al., 2001), with the central principle being to move away from the traditional waterfall-oriented models for developing and releasing software, where updates to the system where big and planned for months. The goal was to achieve an iterative approach to development and maintenance that was driven by feedback from the customer and people using the system.

The agile methods had, however, only sped up the process for developing new code. The process for testing and releasing, including tasks usually handled by a team called Operations, could not keep up (Sharma, 2017). Methods for testing the code and integrating it with the rest of the system needed to become more efficient. This paved way for a new development culture called DevOps, with the aim to bring the separate departments Development and Operations closer together (Bird, 2016; Hüttermann, 2012; Sharma, 2017).

The term DevOps is a composition of the words development and operations (Hüttermann, 2012). The term development encompasses software developers including programmers, testers and personnel handling quality assurance. The term operations represent the personnel in charge of putting software into production including system administrators, database administrators, and network technicians. DevOps can be described as practises streamlining the software delivery process by encouraging development and operations teams to work together in a collaborative way.
2.1.2 Continuous Integration and Continuous Delivery

The concepts continuous integration (CI) and continuous delivery (CD) are central to enable continuous evolution (Davis & Daniels, 2016; Hüttermann, 2012; Sharma, 2017). The motivation behind both concepts is to shorten the total cycle time from the inception of a requirement until it is delivered to the customer. The CI process integrates the newly developed parts with the current system and the CD process handles the release of the integrated result. The CI/CD process is automated through a delivery pipeline that do the different steps of testing and releasing the finished integrated work. A detailed description of the different tasks in the CI/CD process is to be found in Appendix B Section 1.1 and 1.2.

2.1.3 Automation

A key enabler to facilitate continuous integration and delivery is automation (Hüttermann, 2012). This is a process of automating tests, builds and deployments creating releases automatically. Implementing automation will increase efficiency and set up a process that is repeatable. A more detailed description of the different functions of automation for enabling continuous evolution is to be found in Appendix B Section 1.3.

2.1.4 Monitoring

Monitoring is the continuous method of collecting and storing data showing the state of the service and infrastructure (Davis & Daniels, 2016). Monitoring may cover basic information on a system level such as if a server is running or not, how much CPU is utilised, and how much space is left on different disks. It can also be done on a higher level by monitoring how long services take to complete, how many user requests are being handled, and what queries are made to the database for example. An important use of monitoring is continuously overlooking security using, for example, systems for intrusion detection, or keeping track of number of failed attempts to login.

Automated delivery should always be used with monitoring (Hüttermann, 2012). Monitoring is integral in creating the fast feedback that steers the direction of the evolution. Feedback is given through testing before the software has been released, and when it is moved into production, it continues to be provided through monitoring. Effective feedback is important for ensuring the quality of the releases.

2.1.5 Technical challenges for secure continuous releases

This section summarises the technical challenges found for ensuring security while updating software continuously.

2.1.5.1 Shorter release cycles raise concern for quality

One of the main purposes of continuous evolution is reacting faster to customer feedback and demand, which entails doing faster deployments through continuous integration and continuous delivery (Wilde et al., 2016). However, short release cycles may cause concern for the quality of the releases. In a study of the impact of shorter release cycles on software quality,
authors Khomh et al. (2012) came to discover that fewer bugs were fixed using a model for rapid release in contrast with a traditional release model (Khomh, Dhaliwal, Ying Zou, & Adams, 2012). The same conclusion was drawn by Wilde et al. in their report evaluating security for the DevOps deployment process (Wilde et al., 2016). They concluded that considering the time constraints with faster deployments, the use of automation is vital, but not all checks can be automated and the fewer number of human eyes controlling the process raises concern.

### 2.1.5.2 The irony of automation

It should be remembered that highly automated systems creates the risk of operators of said system relying too heavily on everything working by itself (Hüttermann, 2012). When something inevitably fails it becomes harder to fix it. To prevent failures, even more automation and safeguards are introduced to the system, silencing small errors that occur. The silenced errors slowly add up and eventually become increasingly catastrophic. Hüttermann (2012) calls this phenomenon “the irony of automation” (p. 43) (Hüttermann, 2012). Operators of the system need to be experts on said system and not just monitoring an automated process that works perfectly until it suddenly does not.

### 2.1.5.3 Integrating security in the delivery pipeline

Sharma (2017) stresses the importance of integrating security in the delivery pipeline (Sharma, 2017). Stakeholders in security need to work together with the development and operations teams to ensure that the delivery pipelines are secure. Fast releases should not be made at the expense of security. On the contrary, it should be used to enhance it. In a traditional model for software development is security often implemented at the end of the release cycle, right before delivery (Hüttermann, 2012). With DevOps, the intention is to have security implemented in a continuous manner. Insecurities will be found early in the life cycle by securing each small release.

The components of the toolchain in the deployment pipeline also become new surfaces for attack (Wilde et al., 2016). A successful attack on one of these components could become costly. An attacker might have the possibility to read or modify all software it comes in contact with through a compromised component.

The components and tools in the pipeline can be protected through correct use of authorisation and authentication (Wilde et al., 2016). But the case studies conducted by Wilde et al. showed that it was difficult to implement security settings ensuring “least privilege”. Tutorials for setting up the tools often suggested broad permissions to simplify the process. Wilde et al. (2016) warns that organisations may implement these tools without knowing how to properly lock down the system. The virtual environments will also have to utilise keys and certificates for transport layer security and database access for example. The process of handling these keys is hard to implement safely.
2.1.6 Organisational challenges for secure continuous releases

Several challenges to implement secure continuous evolution were found to be of an organisational nature. They are summarised in this section.

2.1.6.1 Varied interpretations of the definition of DevOps

Several studies have been made of the result of implementing DevOps at various organisations (Lwakatare, 2017). A common concern is the varied interpretations of the concept of DevOps. Despite having performed deep analysis of the goal of DevOps before implementing it in the organisation, it is not uncommon that several stakeholders turn out to have understood the concept and practices of DevOps differently.

2.1.6.2 Obstacles for effective collaboration

Traditionally, the activities carried out by the development and operations units have been kept separate from each other with separate goals and objectives (Sharma, 2017). In this case, the development unit is given the task of developing new functionality and fixing bugs, whilst the operations unit focuses on making sure that the software is running smoothly by maintaining secure infrastructure and offering customer support. This could be viewed as problematic since both units are dependent on each other for releasing new software, but they have opposing goals (Sharma, 2017; Wilde et al., 2016). Operations strive for stability, and development strives for change. If the collaboration between development and operations is not working, it will most definitely hinder the process of releasing new functionality and processing feedback.

Pang and Hindle (2016) state that it is generally known that friction tend to exist between the departments for development and operations (Pang & Hindle, 2016). The core of DevOps revolves around close collaboration between the two departments. This close collaboration leads to the line between development and operations becoming blurred. There is a risk that no proper assignment of responsibility between the departments is made, leading to the new culture not being appreciated by those affected (Lwakatare, 2017).

Blurring the line between development and operations also mean that each party need to have competence in both fields. Operations personnel may feel reluctant to giving up responsibility to the development team, if they do not believe they have the right competence needed to handle it (Lwakatare, 2017).

Jones, Noppen, and Lettice (2016) makes a similar point when concluding that a clear challenge for the adoption of DevOps is the fact that both developers and operations personnel need to widen their competences to fit the new way of working (Jones, Noppen, & Lettice, 2016). A high learning curve is to be expected when new methods, tools and technologies are introduced.
According to Sharma (2017), it is often the case that software security is viewed as a hindrance in innovation and creative change among teams developing software (Sharma, 2017). The guidelines provided to ensure compliance with relevant regulations can be perceived as restricting and hard to understand. The developers strive for innovation may stand in contrast with need for stakeholders in security to ensure robust security functions.

2.1.6.3 Assessing security without the full picture

Bird (2016) debates the difficulties in integrating security in a system that does not yet have a finished design (Bird, 2016). A common approach to development in teams with an agile development culture is to start with a version that is the simplest working implementation of the idea. Fast feedback is then intended to steer the direction of development. This is efficient since no time is wasted developing features from the initial idea that turn out not to be needed. However, it may be difficult for the team in charge of security to do proper assessment of threats when the design, requirements, and code keep changing. Properly assessing security often demands that the full picture is provided.

2.2 Flexible infrastructure through cloud computing

This section provides a definition of cloud computing along with architectural principles commonly used with continuously evolving software and the concerns they bring for security. Subsections 2.2.1-2.2.4 provide necessary definitions, and subsections 2.2.5-2.2.6 present threats to security within cloud computing.

2.2.1 Definition of cloud computing

Changing requirements and a greater need for flexibility requires fast development cycles, but also a more flexible infrastructure (Linthicum, 2015). Sufficient computing resources (such as servers and storage) need to be acquired to meet the demands of the system, before the applications can be deployed. Adapting the resources when requirements change is often costly both in terms of time and money. For this reason, many organisations are migrating to cloud-computing solutions (Linthicum, 2015).

Cloud computing allows companies to consume resources for computing provided over internet (Hurwitz et al., 2010), with the significant advantage that resources can be scaled up or down according to demand (Takabi, JosHi, & Ahn, 2011). There are three principal models for cloud software delivery (Hurwitz et al., 2010). These are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

- **Infrastructure as a Service (IaaS)** – With IaaS the hardware is provided as a service, often including the virtualisation technology needed to operate these resources (Hurwitz et al., 2010). The customer rents the infrastructural components on which they build and run their application.
• *Platform as a Service (PaaS)* – With PaaS, not only infrastructure is provided, but also a platform where customers can develop and run their applications (Hurwitz et al., 2010). This means that every part of software development resides in the cloud. From the design phase, all the way through testing, deployment and source-code management. PaaS-platforms completely eliminate the need for provisioning hardware, setting up and managing data centres and networks (Bird, 2016). Services can be scaled according to demand and the platforms also offer built in tools and functions for security and operation management.

• *Software as a Service (SaaS)* – With SaaS, software applications are hosted and provided by the cloud provider as on demand services (Hurwitz et al., 2010). The applications are usually used online with files stored in the cloud as opposed to the user's own unit. More detailed explanations of the capabilities of cloud computing is to be found in Appendix C.

### 2.2.2 Multi-cloud strategy

Many organisations today utilise a multi-cloud strategy (Bird, 2016; Earls, 2017). Definitions on the subject vary among authors, but a multi-cloud strategy implies the use of two or more cloud computing services (Earls, 2017). This is often referred to as a mix of IaaS environments, such as Amazon Web Services (AWS), Google Cloud Platform, or Microsoft Azure. The use of a multi-cloud strategy allows the possibility to choose services from different providers, which is positive since some cloud environments may be more suitable than others for specific tasks (Earls, 2017).

### 2.2.3 Microservices

Microservices are small functions that can be updated, managed, tested, and deployed independently of each other (Bird, 2016). Microservices can be used as an architectural style by developing an application as a collection of microservices working together (Lewis & Fowler, 2014). This enables developers to make changes quickly and try creative solutions which makes it especially suitable for continuously evolving software (Bird, 2016). It is possible for developers to take complete responsibility for their part of the system, including design, deployment and operations.

### 2.2.4 Containers

Containers afford applications to be portable across different environments by providing a standard way of packaging them with defined configurations and dependencies (Sharma, 2017). The business logic specific to the application is kept inside the container and completely isolated from the outside environment. The use of containers along with microservices allows microsegmentation (Bird, 2016). Microsegmentation means that each microservice can run, and be managed, in its own runtime environment.

The use of containers lends itself especially effective for the concepts of continuous integration and continuous delivery (Vaughan-Nichols, 2018).
This is because containers make it possible to create local environments exactly like the live production environment. Identical development environments can be set up on the same host and in turn be used to carry out necessary testing.

2.2.5 Top general threats to security within cloud computing

Several general concerns for software security concern problems for implementing proper functions for authorisation and authentication. A full list of the general threats to security in cloud computing is to be found in Appendix C.

2.2.6 Threats associated with architectural principles often used with continuous evolution

With multi-cloud applications, not only the security of the individual components must be considered, but also the integrated security of the whole product (Rios et al., 2017). This includes the communication and data flow between components.

Bird (2016) mentions three common challenges associated with working with microservices (Bird, 2016):

- **Difficulty to identify complete attack surface** – The attack surface is the sum of each part of the system vulnerable to intrusion. The attack surface of a single microservice might be very small, but when you have a large system consisting of many microservices, the attack surface might be hard to see and understand as a whole.

- **No clear trust boundaries** - There are no clear boundaries where access control, and rules for authentication and authorisation need to be imposed, as with tiered web-applications. Trust boundaries must instead be implemented consistently for every component.

- **Not following conventions** - While developers and teams can work more independently, it is still important to standardise strategies for logging and auditing for example. If different methods are used in separate parts of the system, it will most likely lead to confusion and frustration.

The biggest container platform on the market today is Docker (Bird, 2016; Sharma, 2017). Docker performs operating system (OS) virtualisation allowing independent containers to run in a single instance of a Linux virtual machine. This saves much of the overhead that is normally needed to start and run virtual machines. With virtual machines, a hypervisor is used between the OS of the host computer and the virtualised guest OS. The hypervisor provides the guest OS with a virtual platform and manages its execution. Emulating hardware in this way is heavy on system requirements. Docker containers, on the other hand, can utilise a shared operating system, saving the resources otherwise needed to virtualise hardware. Figure 1 shows how containers are isolated but share OS and other resources, in contrast to virtual machines who have them separated.
Figure 1. Containers allow resources to be shared contra virtual machines. Adapted from “Using Docker” by Mouat (2015).

Mouat (2015) presents 5 concerns for security that should be considered when using Docker (Mouat, 2015):

- **Kernel exploits** – With containers, the kernel is shared between all the containers running on the host. This means that a weakness in the kernel exposes all containers to attack.

- **Denial of Service attacks** – The shared kernel resources also makes the system vulnerable to Denial of Service attacks. One container can be made to monopolise important sources such as memory and user IDs, starving out the others.

- **Container breakouts** – Containers provide isolation, but breakouts are still possible. It becomes important to be careful with what privileges are granted since they are not name spaced. This means that if a process breaks out of the container, it will have the same privileges as it did inside of the container.

- **Poisoned images** – The Docker images provide the correct runtime environment. An attacker can gain access to both the host and the data if a compromised image is used. Docker provides tools to discover and manage this risk, but they must be used correctly.

- **Compromising secrets** – Containers use secrets, such as usernames, passwords or keys to gain access to databases or services. An attacker that successfully obtains the secret can also gain access to that resource. The need to protect secrets to access sources become especially important for microservice architectures since containers start and stop often.

The cloud platforms provide built-in functions handling security and operations management. However, it must be known where the responsibility of the cloud provider ends and the owners of the system begin (Bird, 2016).
2.3 Maintenance and evolution

This section focuses on the maintenance phase of software development, which is entered after the product has been delivered and started being put to use by the customer (Kajko-Mattsson, 2001). At this stage new functionality and corrections to the current software will be taken care of through the maintenance process. Section 2.3.1 presents a classic model for maintenance and evolution. Section 2.3.2 discusses the role of maintenance in the context of continuously evolving software. Section 2.3.3 introduces the concept of continuous maintenance of tools and artefacts and its importance for secure continuous evolution. More detailed information on software maintenance and its role within the software lifecycle is to be found in See Appendix D.

2.3.1 Classic maintenance process model

Kajko-Mattson presents the model over a theoretic maintenance process shown in Figure 2. The process starts with the phase Modification Request Reporting. Here is a request for a modification of the system made using a Modification Request Report. After that two different phases are entered in parallel: The Modification Request Analysis and the Release Planning and Supervising.

During the phase for Modification Request Analysis the request is assessed by a maintainer who makes a judgement on wither or not it will be possible to satisfy the request. If the request is deemed to be possible to satisfy the maintainer also makes a suggestion on how to implement the change. The Modification Implementation phase can then be entered where the changes are made. During the Release Planning and Supervising phase decisions are made for the release in which the change will be implanted. The final phase is Release Delivery where the release containing the change is delivered.

![Figure 2. Evolution and maintenance process and its phases. Adapted from “Motivating the Corrective Maintenance Maturity Model” (CM3) by Kajko-Mattson (2001).](image)
2.3.2 Maintenance with continuous evolution

Bird (2015) denotes that continuously evolving software puts an end to the classic view of maintenance (Bird, 2015). The change, he means, come from the view of development being shifted from running projects to building and running services. Adding new functionality, changes, support work, deployment and so on is all done by the same people, working together.

There are several positive side-effects to this approach to maintenance (Bird, 2015). Real feedback from production can be used steer the development priorities. As large projects get broken down into individual corrections or added features that can be finished quickly and pushed to production, development and maintenance can be valued equally. What is prioritised depends on what is currently most important for the business and what is needed for the system to run.

However, with small changes constantly being made cannot security be assessed as a whole right before deployment (Bird, 2016). Security must instead be integrated completely in the development work. A solution for this, suggested by Bird (2016), would be to include experts on information security in the development and operations teams. The tendency is however for there not being enough personnel with this expertise along with sufficient skills in programming to work with code and design. This means that more responsibility for security must be placed on personnel in development and operations. Figure 3 shows the model Bird suggests for change management in a DevOps culture where security is integrated in every step of the process from idea to deployment of a bugfix or added feature.

![Diagram of DevOps cycle](image)

**Figure 3.** DevOps cycle for bugfix or adding new feature. Adapted from “DevOpsSec Securing Software through Continuous Delivery” by Bird, 2016.
Continuous Maintenance of tools and artefacts

Continuous evolution relies heavily on continuous practises as shown in the previous chapter. The continuous processes rely on numerous different tools and services taking care of integration, testing and deployment. Pang and Hindle (2016) emphasises the need for a process maintaining these services (Pang & Hindle, 2016). An example they provide is the need to free up storage taken up by retained builds and test results created in the CI/CD process. An automated process for continuous maintenance (CM) can here be used to remove irrelevant artefacts and compress the parts that need to be kept to reduce storage space.

Pang and Hindle (2016) define CM as an extension of CI/CD and other continuous processes used for creating and deploying new versions of a system (Pang & Hindle, 2016). The purpose of CM is to maintain the long-term sustainability of the application. Three specific areas where CM should be utilised is mentioned:

- **Build process** – CM is needed to maintain artefacts created in the build process such as meta-data, logs and executables.
- **Test process** – CM is needed to maintain artefacts created in the test process such as virtual machines, database instances, containers, test data and logs.
- **Production** – CM is needed to maintain artefacts created when running the application such as the different logs. An automated maintenance process could be used to first analyse the logs, compact and save them for a certain amount of time before deleting them.

Pang and Hindle (2016) concludes that CM could benefit the DevOps way of working by introducing automated CM processes that handle for example removal of old containers, and virtual machines (Pang & Hindle, 2016).

Feedback to the study

The concerns for security presented in this chapter has been used to formulate checklist for security concerns for continuous evolution. The related works on which the literature study is based on, presents security concerns in the context of each respective subject. No related work was found addressing the overall picture of security concerns for continuously evolving software with cloud computing.

Much of the related works discusses security concerns for continuously evolving software from the perspective of DevOps. The *DevOps Adoption Playbook* by Sharma (2017) and *DevOps for Developers* by Hüttermann (2012) describes the technical challenges with development practises enabling continuous releases of updates to the system. They also discuss difficulties for adopting the practises in an effective way. *DevOpsSec: Securing software through continuous delivery* by Bird (2016) provides an understanding of the biggest challenges for ensuring security with continuously evolving software.
Much has been written on the challenges to security in cloud computing. The focus of this thesis has, however, been to provide a picture of the threats most relevant to architectural principles most relevant for continuously evolving software. Rios et al. with their report *Dynamic security assurance in multi-cloud DevOps* was valuable for its discussion on the unique challenges of using cloud resources from different providers. Bird (2016) contained a chapter dedicated to concerns for security in cloud computing. *Using Docker* by Mouat (2015) provided an interesting discussion on the aspect of security when using a container-based architecture.

Motivating the Corrective Maintenance Maturity Model by Kajko-Mattsson (2001) provided a representation of the classic model for maintenance and evolution. It was vital for creating a basis of comparison with the version of maintenance and evolution used for continuously evolving software provided by Bird (2016).
Chapter 3 Research method

This chapter presents the research method adopted for the study. Section 3.1 presents the research style. Section 3.2 covers the research strategy, including methods for data collection, research instrument and research phases. Section 3.3 provides a short discussion of the research method.

3.1 Research style

This study follows a qualitative research process with a realistic core philosophical assumption. The qualitative research method is best used for understanding meaning, opinions and actions while using these to formulate a tentative hypothesis (Håkansson, 2013). A realistic philosophic assumption accepts that objects in the reality are known which implies the idea of reality being independent from the human mind. A quantitative type of research would not have been suitable since the goal of those methods are to utilise experiments and testing, resulting in mathematical or statistical data, and using this to support or disregard a theory (Håkansson, 2013).

The chosen research method was descriptive research with an inductive approach. The descriptive research method does not focus on the causes behind a situation, but rather examines and describes the characteristics of it (Håkansson, 2013). The focus of the method is to look at a representation of matters as they are, and from this find new connections, and common characteristics. An inductive approach is generally best suited for data collected with qualitative methods. The collected data is used to gain an understanding of the subject by observing it (Håkansson, 2013). A descriptive research method with an inductive approach was considered suitable for this project since the aim was to study what threats to security have been observed in previous studies and what methods were suggested to mitigate them, whilst accepting the information found as factually correct.

For qualitative research with an inductive approach, Håkansson (2013) suggests quality assurance through applying and discussing validity, dependability, confirmability and transferability (Håkansson, 2013). Validity in qualitative research is defined as an assurance that “the research has been conducted according to existing rules” (s. 8), and that the information is verified to have been correctly perceived (Håkansson, 2013). Dependability is ensured by evaluating the correctness of the result by examining the data collection process, analysing the collected data, as well as the result of the research. Confirmability ensures that personal values or assessments has not had an impact on the result. With transferability the aim is to describe the work in a way so that it can be used by others. How these principles were fulfilled is discussed in Chapter 5 Section 5.2.
3.2 Research Strategy

This section reviews the strategy followed. Section 2.2.1 covers the research instrument and process of data collection. Section 2.2.2 presents the criteria for inclusion and exclusion of literature. Section 2.2.3 presents the different phases of the study.

3.2.1 Research instrument and data collection process
The research strategy followed was comprehensive literature review (Onwuegbuzie & Frels, 2016). The strategy can be described as both a method for data collection and analysis, meaning that the research instrument chosen for the study is literature review. Literary sources where gathered and analysed according to the descriptive research method, meaning that information about the different aspects of the study was compiled to find common characteristics, and meaning in the already existing data.

In qualitative research projects, the purpose of collecting data is to provide evidence for the ideas being investigated (Polkinghorne, 2005). The evidence lies in the accounts other people have given on these ideas and the researcher analyses this evidence creating an overall description of the matters involved. The search for relevant literature has mainly been carried out using Google Scholar. The broad subjects touched by this study, such as cloud computing and Devops had several books written on different aspects of the topic. Scientific papers were useful with the narrower subjects such as observed challenges with security in the different areas touched by the study. Other forms of publications such as scientific articles and reports from major organisations such as IDC and CSA were a great entryway into each topic. Some of the search terms are listed below. They have all resulted in thousands of matches but have been narrowed down based on the inclusion and exclusion criteria reviewed in Section 3.2.2.

- DevOps
- Maintenance process
- Continuous Integration, Continuous Delivery
- Cloud computing data security challenges

3.2.2 Inclusion and exclusion criteria
Before the search for literature can be initiated must the criteria for what sources to include be established (Condron, 2018). These are commonly referred to as inclusion and exclusion criteria. Extra caution must be taken when formulating these as to not introduce bias. Table 1 shows the criteria for inclusion that has been used in the data collection process, stating what must be true for the source to be included in the study. Table 2 shows the criteria for exclusion, stating what immediately disqualifies a source from the study.
Table 1. Criteria for inclusion in the literature review.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>The source must have been published in the last 5 years, if describing current common methods and practises.</td>
</tr>
<tr>
<td>Language</td>
<td>They source must be written in either English or Swedish.</td>
</tr>
<tr>
<td>Reported outcomes</td>
<td>The source must be using objective measures.</td>
</tr>
<tr>
<td>Peer review</td>
<td>The source must have been peer reviewed.</td>
</tr>
</tbody>
</table>

Table 2. Criteria for exclusion in the literature review.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicate reports</td>
<td>If the same study is presented in several reports will the most complete version be chosen, and the others excluded.</td>
</tr>
</tbody>
</table>

3.2.2.1 Motivation behind choice of research criteria

Practicing active source criticism became an important task in the assessment of the validity of different sources that the result in based on. The inclusion and exclusion criteria were developed to ensure this.

Older sources than 5 years have been excluded with the aim being to represent a modern development culture. Looking at the way cloud computing is used for example has changed much in recent years and using old sources in this case would not lead to a relevant result. Sources describing classic process models is not affected by the criteria since the purpose of using these is not present current practises but rather show what has changed since then.

Sources written in other languages than Swedish or English have been disregarded due to the researcher only being proficient in these. Translating sources written in other languages would be timewise costly and could result in the content being misinterpreted.

Sources that have not been peer reviewed or where the reported result is not based on objective measures have been disregarded due to the low confirmability of the content.

3.2.3 Research phases

The comprehensive literature review, as described by Onwuegbuzie and Frels, (2016) is divided into seven steps, that in turn are subdivided into three phases as illustrated by Figure 4 (Onwuegbuzie & Frels, 2016).
Figure 4. The seven steps of the comprehensive literature review. Adapted from “Seven Steps to a Comprehensive Literature Review: A Multimodal and Cultural Approach” by Onwuegbuzie & Frels, 2016.

3.2.3.1 Exploration phase

Each step in the exploration phase is investigative in nature (Onwuegbuzie & Frels, 2016). Step 1 explores the topic with a broad focus, finding areas of interest. The research area was discovered through two observations further motivated in Chapter 1: (1) Software needs to evolve quickly to fit the needs of the customer, (2) Data security is more important than ever. Further research based on these observations lead to the following areas of interest:

- Challenges to security within agile software methodology
- Challenges to security within cloud computing
- The role of software maintenance within continuous evolution.

Step 2 involves what information should be selected for the study and what should be excluded (Onwuegbuzie & Frels, 2016). A basic foundational knowledge of the different areas was needed to first be acquired in order to understand the challenges to security. After this was the criteria for inclusion and exclusion formulated.

Step 3 concerns finding an effective method for organising and storing the sources (Onwuegbuzie & Frels, 2016). The citation tool Zotero version 5.0 (“Zotero,” 2018) was used to organise and store the sources. A bibliography with relevant subfolders was created for simple organisation.

Step 4 and 5 furthers focuses the area of study, systematically adding and excluding sources (Onwuegbuzie & Frels, 2016). Challenges to security in the
context of the chosen areas of interest was researched here. The decision to include or exclude a source from the study was made based on the research criteria specified in Section 3.2.2.

3.2.3.2 Interpretation phase
The interpretation phase initiates the process of analysing the selected information that was gathered in the exploration phase (Onwuegbuzie & Frels, 2016). The different inferences are combined to form a coherent narrative.

With literary review as a research method is evidence gathered through written sources (Polkinghorne, 2005). This means that that the literary sources become the ground on which conclusions are drawn. The evidence does not however lie directly in the exact words on the page, which is why methods like counting how many times a word appears in the text for example, is not an effective approach. The task of the researcher is instead to identify the meaning in the text expressed by the authors.

3.2.3.3 Communication phase
The findings are formulated and communicated to the appropriate audiences in communication phase (Onwuegbuzie & Frels, 2016). The goal of this phase is to contribute to the cycle of knowledge by making the result of the study understood and available to the right people. The findings were sorted and summarised into the finished list of security concerns within software evolution, in a way that would be easily understood and accessed to relevant parties. The communication phase also involves letting the thesis be peer reviewed to help improve the validity and confirmability.

3.3 Method discussion
With the research strategy being comprehensive literature review, the task of the researcher is to identify the meaning in the text expressed by the authors (Onwuegbuzie & Frels, 2016). This means that the evidence should be considered indirect, since it is dependent on the interpretation of the person conducting the study (Polkinghorne, 2005). When reviewing literature, the person conducting the study must remember that the result could be used as a base for future research and impact potential stakeholders in the field. Bearing this in mind, the study has been conducted with the intention of creating a result that is as truthful and relevant as possible. The Validity, dependability, confirmability and transferability of the study are further discussed in Chapter 5.
Chapter 4 List of security concerns within software evolution

This chapter presents and discusses the result of the study. Section 4.1 explains how the findings from the literary review were translated into the finished list of security concerns within continuous software. Section 4.2 presents the result.

4.1 Creating a checklist of security concerns

This section presents how the work was carried out to create the list of security concerns within software evolution.

4.1.1 Identifying problem area

The task of formulating the problem to be solved by this study has been a constant process throughout the project. Systematically gathering and reviewing different sources allowed trends to be identified. These trends, and a need to understand specific concepts, drove the direction of the study. Figure 5 shows an overview of how the problem was identified. The red field shows the problem to be solved with this study.

4.1.2 Gathering and structuring information

The search within each subject of the study started by gathering the basic technical definitions needed to understand the challenges associated with security. Once a basic understanding of the topic was obtained was a more focused search was initiated to find the respective issues related security. The findings were then summarised in text.

4.1.3 Formulating the checklist

By studying the summarised threats to security gathered about the different subjects could trends be identified in that several authors had mentioned similar problems. These trends were then given a suitable title and description.

Figure 5. Overview of how the different parts of the study relates to each other.
4.2 Result

The study of relevant theory on security challenges resulted in five areas of concern for security in a development culture for continuous evolution. These are presented in Figure 6 and described in detail in Section 4.2.1-4.2.5.

4.2.1 Developers need to be well-educated in security

One of the biggest challenges for ensuring security in continuously evolving software is the wide competences required of the developers. These are (1) high learning curve, (2) Tempted to allow wider permissions than necessary, (3) Knowledge of the overall security of the application.

High learning curve - When researching what challenges the adoption of a development culture enabling continuous evolution meant for software companies, several sources discussed a high learning curve with the introduction of new technologies, tools and methods (Jones et al., 2016; Lwakatare, 2017).

Tempted to allow wider permissions than necessary - When looking at specific challenges to security, was the complexities in setting up the tools in
the deployment chain with sufficient mechanisms for firewall and authentication/authorization mentioned (Wilde et al., 2016). The process was made easier by allowing wider permissions, which was suggested by tutorials available online on the subject.

Wider permissions do however affect the overall security in a negative way. Four out of the twelve top security threats in cloud computing mentioned by Cloud Security Alliance (2016) involved stolen credentials (Cloud Security Alliance, 2016). If permissions are kept to a minimum, the damage that can be caused if an attacker gets hold of said credentials is also kept to a minimum.

Limiting permissions was also mentioned when discussing the top threats for docker architectures (Mouat, 2015). Containers cannot guarantee isolation, and if a process manages to break out of a container it will keep its initial permissions. Meaning that if the process had root access in the container, it now has root access to the entire host. It becomes important that the personnel in charge of setting up the tools in the pipeline have sufficient knowledge of the system and risk to security certain permissions entail.

Knowledge of the overall security of the application - Overall, the short release cycles mean that methods for evaluating aspects concerning security must be completely integrated in release process (Bird, 2016). This means that developers need to have a wide competence in how to work with complicated tools and environments as well as how the services or products they are developing can be exploited by others.

4.2.2 Security specialists need to collaborate closely with other teams
The classic definition of DevOps refers to enhanced collaboration between development and operations (Davis & Daniels, 2016; Hüttermann, 2012; Sharma, 2017). However, it becomes apparent that security must be an equally integrated part of the process in order to ensure secure releases.

- Miscommunicated guidelines for security - Developers blindly following guidelines provided by the security specialists to ensure compliance with security regulations, may result in them feeling frustrated and hindered in their work (Sharma, 2017). A closer collaboration between developers and security specialists could help create a deeper understanding of what constitutes high security in an application. This deeper understanding would in turn help developers not to view security as a hindrance but instead find more creative ways to develop secure solutions.

- Difficulty of assessing security without the finished design – A clear problem for security specialists working with continuously evolving software is the difficulty of assessing security of a system that does not have a finished design (Bird, 2016). Instead of following the initial idea of the features of a product, the culture of continuous evolution promotes priorities for development being driven by feedback from
users. Since the product is constantly evolving, security specialists have no plan or document to assess from the aspects of security. Security must instead be considered through every step.

- **Wasted effort with securing code that will soon be scrapped** - Developers need to experiment in order to find creative solutions (Bird, 2016). Effort may be wasted if security personnel secure code that might soon be thrown out and not used anymore. Close collaboration and good communication between developers and personnel with the task of securing code is encouraged to minimise this waste.

### 4.2.3 Threats and changing conditions need to be discovered early

Threats in the production environment must be identified and acted upon as quickly as possible, and process of looking for these must be constantly ongoing (Rios et al., 2017).

- **Effective monitoring** - Monitoring is an important part of generating fast feedback to discover threats and act on them quickly (Davis & Daniels, 2016; Hüttermann, 2012).

- **Monitoring functions developed alongside regular development** - Integrating monitoring should not be viewed as a separate task to the rest of development. Services, and the functions taking care of monitoring them, should be developed side by side to ensure that all parts are covered (Hüttermann, 2012). Effective tools for monitoring can be helpful with this task (Rios et al., 2017).

### 4.2.4 Security need to be integrated in the deployment pipeline

Continuous releases requires an automated release process (Lwakatare, 2017; Sharma, 2017). Automated releases are implemented by the use of a deployment pipeline taking care of integration, testing and all forms of quality assurance (Hüttermann, 2012; Sharma, 2017).

- **Sufficient level of security must be ensured** - A great deal of confidence is put on the deployment pipe. Making sure that the tests and quality assurance carried out in the pipe is sufficient becomes an important issue (Pang & Hindle, 2016; Sharma, 2017). Completely integrating security requirements with the deployment pipeline is a daunting task (Wilde et al., 2016). Experts in security need to work together with development- and operations teams on this issue (Sharma, 2017).

- **The irony of automation** - Introducing an increasing amount of automation may lead to smaller errors being silenced through different safeguards making the results from eventual system errors more critical (Hüttermann, 2012). Operators of a highly automated system may also be less equipped to handle a failure if too much confidence is put on everything working perfectly.
4.2.5 Maintenance is needed of the tools in the deployment pipeline

The tools in the pipeline become new surfaces for attack (Wilde et al., 2016). This should be considered a significant threat since a successful attack might enable the attacker to read or modify all the software the compromised component comes in contact with.

- Continuous maintenance of tools and artefacts - It was suggested creating a processes for continuous maintenance, with the purpose of taking care of the development environment and deployment tools (Pang & Hindle, 2016). Continuous maintenance can also be used to free up space and get rid of artefacts and container images created in the automated tests.

- Control is needed of the creation and disposal of containers - Systems with a container-based architecture need to take special care of keeping track of created container images and making sure to get rid of old ones (Mouat, 2015). One of the top five security concerns when using Docker was making sure that the container images in use are safe and have not been tampered with. An attacker can introduce a poisoned container and using this may compromise both the data and the host. Only containers that are up to date should be used to mitigate the risk.
Chapter 5 Conclusion

We live in an age defined by rapid change where software needs to evolve with the customers ever-changing needs and wishes. The huge amounts of data created annually are expected to increase tremendously in the coming years and brings with it the challenge of sufficiently protecting it.

Agile methods have been refined over the last decade to enable rapid releases, allowing development to be steered by the feedback from customers and users. Cloud computing offers the flexible infrastructure and computing resources needed to support it. Difficulty of ensuring security in this context however still remains an issue. The problem dissected by this study is the lacking overall picture of the security concerns during continuous software evolution. The purpose was to help provide this overall picture by creating a list of security concerns within continuous software evolution. The goal of the study was to offer knowledge of the challenges of assuring security for continuously evolving software, providing a foundation for future models for security assurance within continuous software evolution. The list will be valuable for organisations looking to get an understanding of what challenges is to be expected when evolving software continuously whilst staying compliant with current data security regulations.

The study followed a qualitative research process with research instrument and method for data collection being comprehensive literature review. Books and scientific reports were studied to find new connections and common characteristics. The findings were summarised into the points on the list of security concerns within continuous software evolution.

The result of the study is limited to the literary sources acquired on the relevant subjects. It should be remembered that some of the subjects touched upon have been less researched overall. Cloud computing platforms and technologies are evolving rapidly and researching what impact they have on security must be a constant ongoing process. The resulting list of security concerns could therefore not be considered complete. The area of study is vast and further studies would undoubtedly uncover more findings on the topic.

5.1 Research question revisited

This section revisits the research question and summarises the result. The research question was the following:

What areas of security concern exist when maintaining and updating systems with continuous methods in cloud environments?

The study identified five areas of concern for security. These were both technical and organisational in nature. The overall issue that seems to affect most of the areas of concern is the fact that this kind of development culture demands personnel with wide competences. The classic process model for
maintenance and evolution, where releases were few and far apart, allowed time for security specialists to locate eventual vulnerabilities and take steps to mitigate them before release. A development culture where releases are made continuously, demands security to be integrated completely in the development and maintenance process. This means that security specialists must work closer with the development team. But in order for developers to be creative in developing effective and secure solutions, must they be well versed in the topic of security. If security threats are not correctly communicated to- or understood by the developers, there is a risk that they view the security regulations and best practises to ensure high security as a hindrance standing in the way of creative solutions.

As focus is shifted from working on large projects to developing and maintaining small services, the culture of continuous evolution is calling for developers to take full responsibility for their own creations. This means that they must have sufficient knowledge of security threats and how to mitigate them as well as duties relating to operations such as integrating and deploying the software correctly. Finding developers with this range of skill and knowledge will undoubtedly become problematic.

The threats on the technical side concerned challenges mostly relating to the difficulties in implementing effective automation. Setting up the tools needed in the deployment pipeline is challenging which may tempt to allow wider permissions than advisable. The tools also become new surfaces for attack that could become costly. Processes for taking care of the development platforms, artefacts and container images created in the release process must not be forgotten.

Caution should also be taken for what level of automation is implemented. Operators of a system should always be experts on said system to know what steps need to be taken when failures eventually occur. It should also be remembered that automation has a tendency to breed automation, where more safeguards are added to avoid failures, suppressing smaller errors and making problems with the system increasingly critical.

5.2 Validity, dependability, confirmability and transferability

When deriving evidence from the analysis of literature there is a risk the meaning in said texts has been misinterpreted by the person conducting the study. To ensure validity of the study and that the results are correctly understood, has the work been peer reviewed by several parties.

Dependability of the study has been ensured by evaluating the correctness of the result by examining the data collection process, analysing the collected data, as well as the result of the research. The data collection method, further discussed in Chapter 3, prioritised peer reviewed literature to ensure accuracy in the conclusions drawn from them. The collected data is analysed in Chapter 4 where the result is discussed.
Confirmability has been safeguarded by deriving evidence from peer reviewed literature and not basing any part of the result on personal values. The conclusions drawn from the study can be used by any organisation developing cloud products using methods for continuous development. The general nature of the findings ensures good transferability.

### 5.3 Suggestions for future research

This study resulted in a list of security concerns within continuous software evolution. The area of the study is extensive and further studies could be dedicated to extending the checklist with more findings.

Acquiring feedback from organisations working with continuously evolving software would be an effective way to investigate if the findings are correct or if the list can be improved. The list could also be reworked into a more concrete set of guidelines for security assurance within continuous software evolution.
References


Klerk, E., Kersley, R., Bhatti, M., & Vair, B. (2017). Disruptive forces in Europe: A Primer, 49.


Appendix A

1. Personal information and the need to protect it

Privacy is a basic human right. It can be viewed as an expression of human dignity and the self-government of individuals. But since it can conflict with other human rights and values, it can perhaps not be viewed as an absolute right. The freedom of the press, freedom of research, and the Swedish openness principle are examples of areas where collisions may occur. However, a need to protect the very core of privacy has long been an issue resulting in different principles, guidelines and laws that should be followed if personal data are to be collected or processed.

1.2 The right to privacy

An early source discussing the right to privacy for the individual is an article in the December 15, 1890 issue of the Harvard Law Review, written by attorney Samuel D. Warren and Louis D. Brandeis (Warren & Brandeis, 1890). The article was written with the background of the recent technical advancements at the time with the possibility to infringe on the individual’s “right to be let alone”. (p.193) Examples of these were development in photography (in the 1840s taking a picture of someone meant that the object had to remain motionless for around 10-60 seconds to get a clear picture, making it hard to photograph someone surreptitiously), along with new technology for reproducing scenes and sounds as well as the growing phenomenon of sensationalist journalism. With this Warren and Brandeis (1890) points out the risk of "what is whispered in the closet shall be proclaimed from the house-tops." (p. 195). By acknowledging man’s spiritual nature, feelings, and intellect, the rights of the individual needed to be broadened from focusing on the subjects right to physical property, to include all property, meaning the intangible as well as the tangible (Warren & Brandeis, 1890).

1.3 Data Law of 1973

In the year of 1973 the world’s first data law was enforced in Sweden. With the advancing development of the mainframe computer, worry grew that the new technology could be used to gather and exploit information about private individuals (Riksdagsförvaltningen, 1973). The law stated that no data registers containing personal information relating to an individual may be created or used without permission from the Swedish Data Protection Authority.
1.4 Organization for Economic Cooperation and Development Privacy Guidelines

In 1980 the Organisation for Economic Cooperation and Development (OECD) (“OECD,” 2018) strived to achieve common directives for data protection within EU with their publication “Recommendations of the Council Concerning Guidelines Governing the Protection of Privacy and Trans-Border Flows of Personal Data” (OECD, 1980). The guideline contains basic principles for data protection and is summarised below:

1. **Collection Limitation Principle**: Data should only be collected with the knowledge and consent of the involved, and through fair and lawful means.

2. **Data Quality Principle**: Privacy is of course best protected if no personal data is collected at all. The information collected should therefore be relevant, suitable for the purpose, and not excessive.

3. **Use Limitation Principle**: Personal data should not be revealed or made available to other parties or be used for other purposes than the ones specified, without the consent of the involved or by the authority of law.

4. **Security Safeguards Principle**: Personal data should be sufficiently protected from unauthorised access, use, modification, destruction or disclosure.

5. **Openness Principle**: Information about what personal data is collected and to what purpose should be kept in the open. Information about the party gathering the personal data should also be made available.

6. **Individual Participation Principle**: Data subjects should have the right to intervene in the processing of its personal data. For example, to find out what information an organisation/company/authority has stored about them and have the right to challenge it. If challenge is successful the individual should have the right to have the information deleted, corrected or in some way object to the processing of the data.

7. **Accountability Principle**: A controller of data should be appointed the task of making sure that the guidelines are followed.

The basic principles for data protection were all included in The EU Data Protection Directive of 1995 (Datainspektionen, n.d.-a). Each European country was tasked with enforcing their own interpretation of the directive.

1.5 Swedish Data Protection Act

The Swedish Data Protection Act (PuL) (Riksdagsförvaltningen, 1998) replaced the 1973 Data Law in October 1998. The law is based on the EU Data Protection Directive adopted in 1995 which has the purpose of protecting the personal integrity of all EU citizens (Datainspektionen, n.d.-d). The act of handling personal data in PuL comprises for example collecting, registering, organisation, storage, processing, distribution, and disposal.

What laws in PuL apply differs depending on the structure of the data. If the personal data is stored in a register or some form of database, the information is regarded as structured. If the data occurs in running text, such as in emails
or simple lists for example, then it is defined as unstructured (Datainspektionen, n.d.-d). Most of the regulations in PuL do not apply for unstructured data if it does not violate the integrity of the registered individual. This is commonly referred to as “rule of abuse” (“missbruksregeln”, swe) (Riksdagsförvaltningen, 1998).

This means that handling of data in a colloquial and unstructured manner, can be done somewhat without restriction (Datainspektionen, n.d.-e). The purpose of this is to facilitate everyday handling of personal data that is not threatening to hurt the integrity of the involved. To determine if a piece of personal data is sensitive or incriminating to the integrity of the individual, an analysis needs to be made case by case, looking at what context the information will be used, and for what purpose. One must also consider how big dispersion of the information is expected, and what the risks are that it may spread more than anticipated. All possible outcomes from handling of data must be taken into account to make sure that the interests of the registered is protected.

1.6 General Data Protection Regulation

The General Data Protection Regulation (GDPR) (European Parliament, 2016) replaces the national interpretations of the EU Data Protection Directive for all EU-countries in May 2018. The regulation will affect all companies, organisations and authorities that in some way handle personal data. The purpose of having each country in the EU sharing the same data law is to create a uniform and equal level of protection of personal data and to make sure that the free flow of information within the EU is not hindered (Datainspektionen, n.d.-b). Another purpose for creating a new data protection regulation is to modernise the EU Data Protection Directive of 1995, adapting it to the current digital landscape.

A big part of the rules instated by GDPR are already present in the Swedish Data Protection Act. Organisations can process personal data relevant to the service performed with the knowledge and consent of the data subject, providing that the right level of security is ensured. The Swedish Data Protection Authority summarise the following as rules not present in the soon to be replaced Data Protection Act (Datainspektionen, n.d.-c):

- **Data Portability** – Data subjects have the right to obtain and reuse the personal data they have supplied for their own purposes across different services.

- **Impact Assessment** – Before processing sensitive information, should an assessment of the risks to the data subject and how they can be minimised be made.

- **Notification of a personal data breach to the supervisory authority** – At the event of a security breach or unexpected loss of data should a report be made to the supervisory authority (in Sweden the Data Protection Authority) within 72 hours. In some cases, the data subjects also need to be notified.
• **The Data Protection Officer role** – Organisations and authorities handling sensitive personal information or stocktaking of the behaviour of individuals, need to appoint one staff member the role as Data Protection Officer with the duty of overseeing the data protection practices.

• **Penalty for non-compliance** – Organisations found to be non-compliant with the requirements of GDPR may be imposed an administrative fine with the sum depending on several factors such as the severity of the infringement.

• **The simplified rules for handling unstructured data (missbruksregeln) is reversed** – The Swedish Data Protection Act contains simplified requirements for handling personal data in unstructured contexts such as simple lists and running text. This is not the case with GDPR, meaning all forms of handling and storing data is affected by the rules.
Appendix B

1. DevOps

Studying the definition of DevOps made by different sources makes it apparent that it cannot be pinned down as specific process or technology (Davis & Daniels, 2016). Authors Davis and Daniels in the book Effective DevOps (2016) define it rather like a mindset where intentions and issues is communicated between teams, adjusting dynamically to fulfil the goals of the organisation.

The concepts continuous integration (CI) and continuous delivery (CD) are considered central to DevOps (Davis & Daniels, 2016; Hüttermann, 2012; Sharma, 2017). The motivation of both concepts being to shorten the total cycle time from the inception of a requirement until it is delivered to the customer.

1.2 Continuous Integration

The delivery of a software system usually involves several teams of developers that develop different parts of it (Sharma, 2017). Usually the different parts of the system need to communicate with each other to perform its services. This means that when one component is created or changed, it needs to be integrated with the rest of the system. In a traditional development process this task is usually performed after the final component is built. This process can often become costly when issues with integrating with other components may result in major changes having to be made. Something to help mitigate this problem, and help to reveal issues with integration early, is to perform integrations more often and subsequently testing the integrated result (Hüttermann, 2012; Sharma, 2017). The process of performing integration at a regular interval is commonly referred to as continuous integration.

1.3 Continuous Delivery

Continuous delivery has the goal to distribute the new functionality created by the developers out to customers and users (Hüttermann, 2012; Sharma, 2017). At the end of every build, the component or application is brought to the next phase of the application delivery lifecycle which is Quality Assurance (QA). To check if the build is ready to be distributed to the production system, it is first tested in a fabricated staging area designed like the production environment. The process of doing this regularly is referred to as continuous delivery.

With DevOps, both development and operations are involved with QA (Hüttermann, 2012). Hüttermann (2012) points out that it is impossible to guarantee the quality of an application if you do not have control over it. If you do not have the permissions needed to change what is defected, you cannot ensure its quality. He promotes QA done with a holistic approach where
changes can always be made to any part of the application/component, middleware or infrastructure. This unavoidably brings development and operations closer together.

To implement continuous delivery must a delivery pipeline be created (Sharma, 2017). The delivery pipeline is an automated toolchain where a series of mostly automated steps handles the code with little or no manual intervention. One set of tools builds the software and deploys it to the testing area. The testing area is provided by another set of tools. Tests are run on the software and if they pass, it can be delivered to the production environment.

A build-environment would exist in a manual deployment process as well. The parts that are new to continuous deployment is the separate components of the toolchain. Wilde et al. (2016) describes these as being:

- **the Provisioning Service** - the service that creates the instance of the virtual environment.
- **the Tool Store** – contains code for tools needed for the virtual environment.
- **the Artefact Store** – contains the build artefacts needed for the virtual environment.
- **the Deployment Service** – the service that keeps track of which application belongs in which virtual environment and mediates between this and the artefact store.
- **the Deployment Agent** – takes care of tasks needed to start the application in the virtual environment. For example, starting services and copying files (Wilde et al., 2016).

### 1.4 Automation

While the adoption of DevOps has much to do with changing the cultural mindset by bringing development and operations closer together, automation plays an important role on the technical side. Lwakatare (2017) names the following functions of automation in a DevOps workflow (Lwakatare, 2017):

- **Mechanism for standardisation** – Automation creates a standard for the development and delivery process.
- **Reducing human-error** – Reducing mistakes happening because of human error when assessing software manually.
- **Documenting activities** – Managing and documenting activities leading to faster feedback.
Appendix C

1. Cloud Computing

The traditional way of providing IT services is letting the computation take place on physical hard drives and onsite servers (Hurwitz, Bloor, Kaufman, & Halper, 2010). The networking, storage, hardware and software is set up by the IT provider and specialised for the client. With the cloud model, the computing is done on remote servers usually kept by a third-party hosting company with the infrastructure virtualised across the globe. The exact place where the computations and data are made is often not known.

Internet based utility computing has been widely used by the public since the mid-90s through email-services, search engines, self-publishing and social media platforms such as YouTube, Twitter and Facebook (Hurwitz et al., 2010). These consumer-centric services lay a foundation for the core concept of the modern-day cloud computing.

In an article for IEEE Security and Privacy Magazine, authors Takabi, JosHI and Ahn (2011) summarise five principal characteristics of cloud computing:

• on-demand self-service
• ubiquitous network access
• location-independent resource pooling
• rapid elasticity
• measured service

(p. 24)
With rapid elasticity referring to the possibility to quickly scale resources up or down, and measured service indicating that it is the cloud service providers that optimise and control the computing resources (Takabi et al., 2011).

1.1 Cloud Security Alliance top 12 threats

Cloud Security Alliance (CSA) (“Cloud Security Alliance,” 2018) is a not-for-profit organisation with a stated mission to “promote the use of best practices for providing security assurance within Cloud Computing, and to provide education on the uses of Cloud Computing to help secure all other forms of computing.” In 2016 they published a report summarising the twelve biggest security threats in cloud computing (Cloud Security Alliance, 2016). These were:

1. Data Breaches

An incident where confidential data is viewed by or stolen by a person who was not authorised to do so is called a data breach. It may be the result of directly targeted attack, human error, vulnerabilities in the code or poor security practices. The risk of a data breach is of course not exclusive to cloud environments, but since the customers need to rely on a third party handling
their data, a great confidence in the security provided by the cloud provider is needed.

2. Insufficient Identity, Credential and Access Management

According to CSA, unauthorised access to data is often the result of insufficient identity, credential and access management. Data breaches can be prevented by, for example, multifactor authentication, strong password usage, a constant automated rotation of cryptographic passwords, keys and certificates.

3. Insecure interfaces and application programming interfaces

To interact with the cloud services, the customers need to have access to different user interfaces (UIs) and application programming interfaces (APIs). CSA emphasises that since management, monitoring and provisioning are all handled by these interfaces, the security and availability of the cloud services depend on these.

4. System Vulnerabilities

CSA defines system vulnerabilities as bugs in the code that can be exploited by attackers to infiltrate the system. With the increased use of single instances of software running on a server to serve several tenants (software multitenancy), systems from different organisations are placed closer together and given access to the same memory and resources. This opens new areas for attacks.

5. Account Hijacking

With account hijacking, the attacker utilises stolen credentials to, for example, manipulate data, snoop on activities and transactions, or return falsified information. According to CSA, cloud service instances might become a new base for attackers since the attacker will have access to critical areas of cloud services.

6. Malicious Insiders

CSA points out that a system that only depends on the cloud service provider for security are at risk for malicious insiders. Even though it probably is not the biggest threat, it should still be taken into consideration. To control that no one has unnecessary access to data or parts of the system is of great importance.

7. Advanced Persistent Threats

Advanced persistent threats (APTs) is a type of cyber-attack that infiltrates the system by creating a foothold in the infrastructure and blending in with the normal network traffic. APTs often achieve their goals over an extended time, adapting to the security measures put in place to protect against them.
8. Data Loss

Data stored in the cloud is under the risk of being lost for several reasons according to CSA. It could be accidentally deleted by the cloud service provider, lost due to physical damage to servers caused by fires or earthquakes, or deleted or stolen through malicious attacks. Proper measures must therefore be taken to back up all data.

9. Insufficient Due Diligence

CSA emphasises the need for organisations to perform due diligence when adopting cloud technologies and choosing providers.

10. Abuse and Nefarious Use of Cloud Service

CSA warns against free cloud service trials with fraudulent account sign-ups and poorly secured cloud service deployments. Cloud computing resources can be utilised to launch denial-of-service attacks, phishing campaigns or email spam for example.

11. Denial of Service

Denial-of-service attacks are meant to prevent users from being able to access the application or data. The cloud service is forced to utilise unwarranted amounts of system resources such as processor power, disk space or network bandwidth causing the system to slow down and not being able to respond.

12. Shared technology vulnerabilities

For cloud service providers to be able to offer scalable services, certain infrastructure and platforms need to be shared, which may have a negative impact on security. Moderations to prevent breaches in shared resources should be implemented. CSA suggests multifactor authentication on all hosts, host-based intrusion systems (HIDS) and Network-based Intrusion Detection Systems (NIDS).
Appendix D

1. Software Maintenance and Evolution

The practice of software maintenance is as old as software development. However, Kajko-Mattson (2001) describes maintenance as the neglected stepsister to Software Development, since more focus and research tends to be put on development (Kajko-Mattsson, 2001). For most software organisations, maintenance is however a dominating cost factor that should not be overlooked. Pang and Hindle (2016) agree that maintenance processes tend to be overlooked (Pang & Hindle, 2016). They are often integrated into the development process at a later stage, after the application runs into issues with sustainability.

Maintenance is considered one of the major phases of the software life cycle (Kajko-Mattsson, 2001). The three major phases are: Software Development, Operation, Software Evolution and Maintenance and Retirement. During the software development phase, the software is designed and implemented from scratch according to requirements from the customer. When the product has been delivered to the customer, the operation phase and evolution and maintenance phase is entered simultaneously. As the customer organisation runs the system, the developer organisation supports the daily operation as well as further developing and maintaining it. The product is evolved with new functionality and corrects problems encountered during operation. When the decision is made for a system to no longer be in use, it enters the retirement phase.

Software maintenance is normally sorted into different categories. The IEEE standard uses the following groupings: corrective-, adaptive-, perfective-, and preventative maintenance. IEEE also uses subgroups according to Figure 2 (ISO/IEC, 1999)

<table>
<thead>
<tr>
<th>Correction</th>
<th>Enhancement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proactive</td>
<td>Preventative maintenance</td>
</tr>
<tr>
<td>Reactive</td>
<td>Corrective maintenance</td>
</tr>
</tbody>
</table>

Figure 2 IEEE categorisation of different types of maintenance.