Continuous Security in DevOps environment: Integrating automated Security checks at each stage of continuous deployment pipeline

A Master's Thesis submitted for the degree of “Master of Science”

supervised by
Dr., Lawrence STAPLETON, Ph.D., MA, BA Mod (Hons), CPIM

MOHAMMED JAWED, B.E

11728338

Vienna, 24.04.2019
Affidavit

I, MOHAMMED JAWED, B.E, hereby declare

1. that I am the sole author of the present Master’s Thesis, "CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT: INTEGRATING AUTOMATED SECURITY CHECKS AT EACH STAGE OF CONTINUOUS DEPLOYMENT PIPELINE", 95 pages, bound, and that I have not used any source or tool other than those referenced or any other illicit aid or tool, and

2. that I have not prior to this date submitted the topic of this Master’s Thesis or parts of it in any form for assessment as an examination paper, either in Austria or abroad.

Vienna, 24.04.2019

_______________________
Signature
Acknowledgments

Foremost, I would like to acknowledge my indebtedness and render my sincerest thanks to my academic supervisor, Dr. Lawrence Stapleton (Ph.D., MA, BA Mod (Hons), CPIM). His responsive guidance, expert advice and feedback have been invaluable throughout this thesis work. Thank you so much, I am fortunate to work under your supervision during this thesis work.

My sincere thanks goes to Prof. Dr. Peter Kopacek (Academic Director) for his patience and nurturing me in the Engineering management areas, and to program assistance Ms. Sandra Wagner for providing cordial and timely support.

On the professional grounds, I am grateful to PARTEE, Michael Scott (Information security & Systems engineering Leader at International Atomic Energy Agency (IAEA), Vienna) for his comments and valuable suggestions, which have contributed greatly to the improvement of the thesis.

It would not be complete without to express my gratitude to I owe a special thanks to my AMMI and ABBU who supported me and helped me throughout my life. Ammi, Abbu I do not know how to thank you enough for providing me with the opportunity to be where I am today.

At last, I do not know how to begin with saying thank you to my soul mate, my dearest wife and my best friend, Tamanna. I love you for everything, for being so understanding and for putting up with me through the toughest moments of my life. I thank God for enlightening my life with your presence. You are the best thing that has ever happened to me.

Thank you all,

Mohammed Jawed
Abstract

In this digital transformation, when the world is witnessing record security breaches, and hackers are getting bolder and more sophisticated, the rapid development and delivery of software is not enough. Organization must also secure the applications.

To get the secure application out of the delivery team hands and in front of customers, organization must prioritize security and make it a standardized part of a daily operational Procedure. This is only possible when an organization integrates security checks and practices throughout the software development lifecycle to better defend their applications and protect their brand values.

This thesis outlines a secure DevOps delivery workflow to achieve continuous security by infusing security controls, tools, compliance, best practices, and automated security checks at each stage of the software development cycle so that all the security testing is done seamlessly through a continuous delivery pipeline.

The secure DevOps delivery workflow designed in this thesis will allow organizations to make security a standardized part of their daily operational procedures. The results will be a resilient organization and culture of “everyone responsible for security,” “Scaling through automation,” and “Measurable outcomes.” Moreover, it will save effort by building a better, faster, cheaper and more secure product while retaining the focus on delivering customers-centric features. This frees up resources from fighting with security-related bugs and compliance requirements and allows organization to fend off more attacks, leading to an overall more protected system.

The secure seven-stage workflow release gateway presented in this thesis are Requirements, Plan, Secure Develop, Build, Test, Deploy and Continuous Monitoring – with fully integrated security checks at each stage. It is applicable to all DevOps enabled organization and serves to improve productivity, organizational security knowledge, shift security to left, and transform the application release cycle to produce more secure applications.

As part of the definition and to facilitate the adoption of the secure seven-stage workflow for DevOps, additional areas of guidance are provided:

- **The Secure DevOps work value stream**, to help DevOps team members evaluate the proposed seven-stage gateway workflow and provide everyone in the value stream with the fastest possible feedback about the security of what they are creating, enabling them to quickly detect and correct security problems as part of their work, which enables learning and prevents future errors.

- **Security testing DevOps toolchain**, set of security tools integrated within Continuous deployment pipeline to auto checks security at each stage of the workflow, to help the delivery team identify and fix flaws earlier in the delivery process before flaws exposed to the public without impeding agility and this shift security to the left.

- **Secure DevOps workflow model** by building security into every stage of the secure development life cycle, from the security requirements stages onwards, and this makes everyone responsible for security.

**Contributions of this thesis consist of:** a secure seven-stage continuous security workflow infused with security controls, compliances and best practices; flow of work across continuous security value stream; Anatomy of continuous security assurance model; Toolchain to infuse automated security checks during CI/CD pipeline.

**Keywords:** agile; DevOps; DevSecOPs; continuous security; security Toolchain; continuous integration; continuous delivery; security testing; secure development; continuous monitoring; cybersecurity; Secure DevOps;
## Contents

List of Figures ........................................................................................................................................ vi
List of Tables .......................................................................................................................................... vii
List of Abbreviation .............................................................................................................................. viii
1 Introduction .......................................................................................................................................... 1

2 Literature Review ................................................................................................................................. 5
  2.1 Cybersecurity ..................................................................................................................................... 5
  2.2 Agile software development methodology ....................................................................................... 8
  2.3 DevOps ............................................................................................................................................ 9
  2.5 Agile Development and traditional Cybersecurity in DevOps: Challenges ..................................... 13

2.6 Summary .......................................................................................................................................... 14

Chapter 3 Continuous Security workflow: Solution to secure DevOps environment ............................. 16
  3.1 Requirements .................................................................................................................................... 18
    3.1.1 Security Requirements for web applications ............................................................................. 19
    3.1.2 OWASP Application Security Verification Standard (ASVS) .................................................... 19
      3.1.2.1 Security requirement defined by OWASP ASVS ................................................................. 20
    3.1.3 Security requirements for Release gate ..................................................................................... 21
    3.1.4 Common Vulnerability Scoring System (CVSS) metrics ......................................................... 21
    3.1.5 Benefits of having Security requirement for a base of our Continuous security workflow ........ 24
  3.2 Plan ................................................................................................................................................ 25
    3.2.1 Security Design principle and explanation ................................................................................. 25
    3.2.2 Security Properties and description ........................................................................................... 26
    3.2.3 OWASP top 10 proactive controls for secure architecture and design ....................................... 27
    3.2.4 Data Masking For Privacy (DMFP) ............................................................................................ 28
    3.2.5 Threat Modeling ......................................................................................................................... 29
      3.2.5.1 What are we building (Diagram)? ....................................................................................... 30
      3.2.5.2 What can go wrong (Identify Threats)? .............................................................................. 30
      3.2.5.3 What are we doing to defend against the threat? (Mitigate) .............................................. 31
      3.2.5.4 Did we do a good Jobs (Validation)? ................................................................................... 32
      3.2.5.5 Benefit of using threat modeling ......................................................................................... 32
      3.2.5.6 EoP(Elevation of Privilege) card game .............................................................................. 33
    3.2.6 Secure architecture review .......................................................................................................... 33
    3.2.7 Security controls assessment for cloud computing ................................................................. 34
  3.3 Develop .......................................................................................................................................... 34
    3.3.1 Team Activities .......................................................................................................................... 35
3.3.1.1. Secure coding best practices .................................................................35
3.3.1.2 Security Knowledge Base........................................................................36
3.3.1.4 Open Source Firewall Prevention ..........................................................36
3.3.2 Pre-commit checks .....................................................................................37
  3.3.2.1 IDE plugin (Secure Coding Analysis) ......................................................37
  3.3.2.2 Threat modeling analysis .....................................................................38
  3.3.2.3 Code Review .........................................................................................38
3.3.3 Commit-time checks ..................................................................................38
  3.3.3.1 Source Code Scanning for Malware detection .........................................39
  3.3.3.2 Secure code build ................................................................................40
  3.3.3.3 Secure compiling ..................................................................................40
  3.3.3.4 Security Unit testing ............................................................................40
  3.3.3.5 Static application security testing (SAST) .............................................41
3.3.4 Putting all automated steps together in Develop Stage ...............................42
3.4 Build.............................................................................................................42
  3.4.1 Secret Information Leakage Testing (SILT) ................................................43
  3.4.2 Software Composition Analysis (SCA)......................................................44
  3.4.3 Security Unit testing .................................................................................45
  3.4.4 Static Application security testing (SAST) .................................................46
  3.4.5 Digital Signing .........................................................................................46
3.5 Test ................................................................................................................47
  3.5.1 Dynamic Application Security Testing (DAST) ..........................................48
    3.5.1.1 to summaries DAST technique and analysis .........................................48
    3.5.1.2 Taxonomy of Dynamic application security testing tools .................49
  3.5.2 Interactive Application Security Testing (IAST) ........................................49
    3.5.2.1 Advantage of using IAST in security testing phase to drive continuous security .................................................................50
    3.5.2.2 Taxonomy of Interactive application security testing tools ................51
  3.5.3 Security Test best practices ...................................................................51
3.6 Deploy ..........................................................................................................51
  3.6.1 Securing Infrastructure Configuration .....................................................52
    3.6.1.1 Taxonomy of Securing Infrastructure Configuration tools ................52
  3.6.1 Hardening the environments ..................................................................52
3.7 Monitoring ....................................................................................................53
  3.7.1 Web Application Firewalls (WAFs) ..........................................................54
  3.7.2 Runtime Application Self-Protection (RASP) ...........................................54
    3.7.2.1 The problem RASP addresses .........................................................55
3.7.2.2 How RASP works ........................................................................................................55
3.7.3 Threat intelligence ........................................................................................................55
3.7.4 Taxonomy of Security Monitoring Tools .......................................................................56
4 A flow of work in a continuous security value stream .........................................................57
5 Anatomy of Continuous Security workflow ........................................................................58
6 Taxonomy of continuous security toolchain to secure DevOps environment .....................59
7 Discussion and Conclusion .................................................................................................61
   7.1 Quantitative Research Method: Survey ......................................................................61
   7.2 Analysis of the Responses: Discussion ......................................................................63
   7.3 Conclusion ...................................................................................................................65
8 Future Work .........................................................................................................................68
Appendix A – PowerShell Script to hook Metadefender API within CI/CD Pipeline ..............69
Appendix B – Script to use signtool.exe for digital signing of Code .....................................71
Appendix C – Owasp Application Security Verification Standard 3.0 .................................73
Appendix D – Survey Questionnaire ......................................................................................77
Bibliographic .........................................................................................................................79
List of Tables

Table 1 category of activities into DevOps Life cycle stages ........................................17
Table 2 Three aspects of security requirements .................................................................19
Table 3 CVSS Basic Metric Group's overview (CVSS, v3.0) .................................................23
Table 4 Saltzer and Schroeder’s Design Principles (Richard, 2012) ......................................26
Table 5 Security Properties and description (Fotis, 2018) ....................................................26
Table 6 Security controls ........................................................................................................27
Table 7 Proactive controls .....................................................................................................28
Table 8 Different type of Data masking and differences .........................................................29
Table 9 STRIDE Description ..................................................................................................31
Table 10 STRIDE mitigation approach ..................................................................................32
Table 11 Software security-testing types ..............................................................................47
Table 12 DAST technique and analysis ...............................................................................48
Table 13 Taxonomy of DAST Tools .....................................................................................49
Table 14 Advantages of using IAST in security testing phase ..............................................50
Table 15 Taxonomy of IAST tools .......................................................................................51
Table 16 Security test best practices ....................................................................................51
Table 17 Taxonomy of securing infrastructure configuration tools ......................................52
Table 18 Taxonomy of Security Monitoring Tools ...............................................................56
Table 19 Distribution of respondents ..................................................................................62
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASVS</td>
<td>Application Security Verification Standard</td>
</tr>
<tr>
<td>CAIQ</td>
<td>Consensus Assessments Initiative Questionnaire</td>
</tr>
<tr>
<td>CD</td>
<td>Continuous Delivery/Deployment</td>
</tr>
<tr>
<td>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>CIS</td>
<td>Centre for Internet Security</td>
</tr>
<tr>
<td>CSA</td>
<td>Cloud Security Alliance</td>
</tr>
<tr>
<td>CSC</td>
<td>Cybersecurity Culture</td>
</tr>
<tr>
<td>CVSS</td>
<td>Common Vulnerability Scoring System</td>
</tr>
<tr>
<td>DAST</td>
<td>Dynamic Application Security Test</td>
</tr>
<tr>
<td>DSC</td>
<td>Desired State Configuration</td>
</tr>
<tr>
<td>FedRAMP</td>
<td>Federal Risk and Authorization Management Program</td>
</tr>
<tr>
<td>GDPR</td>
<td>General Data Protection Regulation</td>
</tr>
<tr>
<td>GPL</td>
<td>General Public License</td>
</tr>
<tr>
<td>HIPAA</td>
<td>Health Insurance Portability and Accountability Act</td>
</tr>
<tr>
<td>IAST</td>
<td>Interactive Application Security Test</td>
</tr>
<tr>
<td>ISO27001</td>
<td>Information Security Management System</td>
</tr>
<tr>
<td>MASVS</td>
<td>Mobile Application Security Verification Standard</td>
</tr>
<tr>
<td>NCP</td>
<td>National checklist program</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>OECD</td>
<td>Organisation for Economic Co-operation and Development</td>
</tr>
<tr>
<td>OTM</td>
<td>Operation threat modeling</td>
</tr>
<tr>
<td>OWSAP</td>
<td>Open Web Application Security Project</td>
</tr>
<tr>
<td>PCI DSS</td>
<td>Payment Card Industry Data Security Standard</td>
</tr>
<tr>
<td>PIA</td>
<td>Privacy Impact Assessment</td>
</tr>
<tr>
<td>RASP</td>
<td>Runtime Application Self-protection</td>
</tr>
<tr>
<td>SAST</td>
<td>Static Application security testing</td>
</tr>
<tr>
<td>SCA</td>
<td>Software Composition Analysis</td>
</tr>
<tr>
<td>SCAP</td>
<td>Security Content Automation Protocol</td>
</tr>
<tr>
<td>SDLC</td>
<td>Security Development Lifecycle</td>
</tr>
<tr>
<td>SIEM</td>
<td>Security information and event management</td>
</tr>
<tr>
<td>SILT</td>
<td>Secret Information Leakage Testing</td>
</tr>
<tr>
<td>STRIDE</td>
<td>Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege</td>
</tr>
<tr>
<td>TI</td>
<td>Threat intelligence</td>
</tr>
<tr>
<td>USD</td>
<td>Unified Security Dashboard</td>
</tr>
<tr>
<td>WAF</td>
<td>Web Application Firewalls</td>
</tr>
</tbody>
</table>
1 Introduction

In today’s digital world, data breaches, exploitation of vulnerabilities and potential threat are becoming all too common, and it is increasing with alarming scale and multiplying exponentially.

In September 2017, the U.S.-based firm Equifax revealed a massive data breach due to a vulnerability in a website application. This breach affected 143 million customers. This attackers stole sensitive data which included names, birth dates, social security numbers, addresses, and even driver’s license numbers (Steve Brown, 2018). Not to forget the case of Uber in November 2016, in which hackers stole information related to as many as 57M rider and driver accounts.

In a more recent incident, on Feb 25th, 2019, verifications.io, an email address service provider, suffered a data breach due to a public-facing 150GB-sized MongoDB instance which was left unprotected without a password, resulted in 763 million unique email address and personally identifiable information (PII) being leaked (Bob Diachenko, 2019).

In the same month Feb 2019, SenseNets a Shenzhen-based artificial intelligence company that operates facial recognition systems in China, has exposed crucial personal information of 2.5 million people after leaving a database unprotected (Danny Bradbury, 2019).

Additionally, the recent global ‘success’ of the wannacry ransomware (May 2017), which utilized a well-known, but unpatched, system flaw, demonstrate the consequences of ignoring security as a business priority (DevSecOPs, 2017).

Based on the CRN report (Michael Novinson, 2018), breaches are not only increasing but also changing in nature. They are not only limited to government agencies and fortune 500 customers, but go beyond this to include third-party contractors, solution providers, and even security vendors. These security threats are multiplying exponentially.

At this time, the world is witnessing record security breaches, and hackers are getting bolder and more sophisticated -- and smarter. In all those breaches the data remains the primary hacker target, Microsoft predicts that by 2020 data volumes online would be 50 times greater than it is today. There are 111 billion lines of new software code being produced each year — which will include billions of vulnerabilities that can be exploited (Steve Morgan, 2016).

The issue of information security is becoming more crucial in today’s information age as the privacy and security issues of information resources face many challenges due to several factors such as the development of Information Technology (Gebrasilase and Ferede, 2011). According to one of the cybersecurity services provider (Bulletproof annual cybersecurity report, 2019), the Top 5 common issues found in 2018 was:

1) Missing patches and out of date software
2) Default or poor password
3) Cross-site scripting (XSS)
4) SQL Injections
5) Brute force attacks

On one hand, security is a huge challenge that can have dire consequences if improperly handled and not taken security as the number one priority to address. As per the report, the cybercrime in its various forms is expected to cost the world more than US$6 trillion by 2021 (Steve Morgan, 2016).
CHAPTER 1. INTRODUCTION

When companies suffer a data breach, they do not only incur the cost of data damage and destruction, stolen money, IP theft, business disruption, and reputational harm. Other costs, such as legal and PR fees, drops in share price, interruptions to e-commerce, loss of customers and competitive advantages can also affect organizations (DevSecOps, 2017). To deal with cybercrime in 2015, U.S President Barak Obama declared a national emergency (CORY and ELISE, 2015).

**On the other hand**, as software continues to ‘eat the world,’ high velocity IT becomes the foundation of competitiveness in the modern marketplace governed by shorter times to market. Every business should become an agile and innovative software delivery machine in order to survive Which leads us to the enterprise IT paradox: Go faster and innovate. But always stay secure (DevSecOps, 2017).

Companies are increasing pace by combing agile and DevOps to build software to better satisfy customers by delivering what customers need, when they need it. It is critical to go with the pulse of fast-moving markets and match customers’ demands by adopting Agile and DevOps to cut the time-to-market by third or more.

DevOps enables an agile delivery methodology by automating the delivery of minimally viable products, Once a minimum viable product is delivery, it is regularly refined based on user and customer feedback. DevOps is the new world of developing, releasing, and updating applications for a growing number of enterprises (Dave Meltzer, 2018). However, the more technology-centric our organizations become and the faster they go, the higher the chance that a hacker will find that one vulnerability that will suck away all that hard-earned customer value (Jason Bloomberg, 2017).

Even though an organization has adopted the DevOps they are struggling with the shortage of InfoSec (information security) people.- The ratio of engineers in development to operations, and to InfoSec is 100:10:1. When InfoSec people are outnumbered, without automation and integrating information security into the daily work of Dev and Ops, InfoSec can only perform compliance checking, which is opposite of security engineering (Gene Kim, 2017). InfoSec people will be distressed due to being stuck in the “checkbox trap” of just checking compliance, and many InfoSec people the same challenges that developers and operations teams did before the birth of DevOps.

Due to the disconnection of InfoSec from the DevOps movement, security has become a stand-alone part of the corporate IT organization. InfoSec become self-siloed and disassociated from the broader organizational objectives they are tasked with protecting, and keeps the business side of organizations at arm’s length. Without an understanding between the various departments that create a business, and a holistic security strategy, many companies defaulted to relying solely on regulatory compliance for the sake of maintaining the status quo versus placing focus on proactive security measures that protect the organization responsibility, mitigate risk, and adapt to an ever-changing world (Grant Wernick, 2019).

To make matter worse, InfoSec is inundated with alerts that can be eventually categorized as false positives, which results in teams chasing down logs and other events only to find that there is not an actual threat. Meanwhile, alerts that do pose a real danger may not be investigated fast enough -- or at all. The threat landscape is growing exponentially, and bad actors are more creative than ever-think Mirai, botnets, and unique malware. It is increasingly difficult for defenders to keep up, let alone get ahead of these threats (Jen Andre, 2018).

Despite DevOps teams utilising numerous tools and methods to build and deploy their infrastructure and their own custom applications, multiple issues remain in enterprises where security in the development life cycle is either ignored, overlooked, or not seen as a primary concern. As a result, many enterprises and even some of the major DevOps software vendors are beginning to realise that incorporating security practices is a must-have, not a nice-to have (Murray Goldschmidt and Michael McKinnon, 2016).
CHAPTER 1. INTRODUCTION

The crucial question is how to secure the products and minimize these security issues while maintaining the value of these accelerated development cycles.

The golden answers to the above question are to: “Integrate security early in the development cycle;” “ensure everyone responsible for security;” “Shift security to the left;” and “add security checks at each stage of the release cycle;”

One practice to achieve it is to integrate security controls and automated checks throughout the software development lifecycle. This will help the organization to address new risk exposure and identify security issues early in the development process rather than after a product is released or in experiments phase.

Therefore, the goal of this thesis is to achieve continuous security by designing a secure delivery workflow release gates infused with automated security checks at each gate early in the development cycle to keep the DevOps agility at pace and get the organization out of the checkbox trap and to secure DevOps delivery.

The continuous security workflow designed in this thesis will allow organizations to make security a standardized part of their daily operational procedure and results in a resilient organization and culture of “everyone responsible for security,” “Scaling through automation,” and “Measurable outcomes.” Moreover, it will save team effort to put into building a better, faster, cheaper and more secure product with a focus on delivery of customers centric features instead of fighting with security-related bugs and compliance issues as well as to fend off more attacks. All of this leads to an overall more protected system.

The continuous security workflow consists of seven stages, which are, Requirements, Plan, Secure Develop, Build, Test, Deploy and Continuous Monitoring. Each stage is infused with security controls, tools, compliance, best practices, and automated security checks so that all the security requirements and controls can be implemented seamlessly through a continuous delivery pipeline.

This thesis aims to achieve Continuous security in DevOps environment, by integrating security checks at each stage of the software development cycle in CI/CD pipeline without affecting speed or agility.

![Continuous Security in DevOps](image)

The continuous security workflow uses the following tactics to solve some of the main challenges faced by DevOps-enabled organizations:

1. Security as code.
2. Securing DevOps by designing a secure workflow delivery model.
5. Shifting security to the left of CI/CD assembly line.
CHAPTER 1. INTRODUCTION

This thesis begins with the systematic review of the existing literature on cybersecurity and security culture within the agile software development methodology and the DevOps culture.

Secondly, formulate the current challenges into agile development and DevOps practices.

Finally, Design of an effective workflow using agility and DevOps automation practices by integrating security checks at each stage of the software delivery process.

This thesis work is organized in eight chapters supported with appendix section and structured as follows:

Chapter 1: This chapter includes an introduction and highlights the importance of having security in DevOps practice by Securing the DevOps workflow, the aim and goal of this thesis.

Chapter 2: This chapter covers the literature review on cybersecurity and security culture, agile software methodology, and DevOps. The following chapter also covers challenges faced by DevOps cultural organization regarding getting secure products out in the market.

Chapter 3: This chapter covers the design of a secure DevOps delivery workflow to achieve continuous security in details, which includes purpose, security controls, best practices, and security toolchain at each stage.

Chapter 4: This chapter defines a security value stream to visualize the flow of work to provide visibility and insight across all stages of security checks in securing the DevOps.

Chapter 5: This chapter includes an Anatomy of the Secure DevOps workflow with security controls. The purpose of this chapter is to provide summary to the delivery team to view all seven-stages of continuous security in DevOps environment.

Chapter 6: This chapter will summarise the security toolchain used in chapter 3. The toolchain includes commercial and free to use tools. Having all toolchain and practices in single view will help the team to refer and practices efficiently.

Chapter 7: This chapter is a discussion and definition of goals, aims, and purpose of in detail. This chapter also includes results of survey, which was derived after implementing Secure DevOps workflow in DevOps enable Organization across various projects delivery lines.

Chapter 8: This chapter summarizes future work to enhance the DevOps delivery processes, such as the use of Artificial Intelligence and Big Data to secure the product predictably.

Also included are a few appendices, which contain custom scripts, codes and reference tables.
2 Literature Review

2.1 Cybersecurity

Dictionary meaning of Cybersecurity: “The state of being protected against the criminal or unauthorized use of electronic data, or the measures taken to achieve this.”

Cyber Security is a whole set of procedures and systems protecting computer systems, and networks from the intentional and unintentional damages or dangers in the cyberspace through services like confidentiality, integrity, authentication, availability, non-repudiation, auditing, and digital signature (Asif and Hussain, 2010).

“It encompasses a broad range of practices, tools, and concepts related closely to those of information and operational technology security. Cybersecurity is distinctive in its inclusion of the offensive use of information technology to attack adversaries (Darko and William, 2017).”

The most preferred definition of cybersecurity as stated by International Telecommunications Union (ITU,) : “Cybersecurity is the collection of tools, policies, security concepts, security safeguards, guidelines, risk management approaches, actions, training, best practices, assurance and technologies that can be used to protect the cyber environment and organization and user’s assets. Organization and user’s assets include connected computing devices, personnel, infrastructure, applications, services, telecommunications systems, and the totality of transmitted and/or stored information in the cyber environment. Cybersecurity strives to ensure the attainment and maintenance of the security properties of the organization and user’s assets against relevant security risks in the cyber environment. The general security objectives comprise the following”:

- Availability
- Integrity, which may include authenticity and non-repudiation
- Confidentiality.

According to Grant Wernick, Co-Founder & CEO of Insight Engines: “Cybersecurity was born in the shadows by three-letter acronym agencies-NSA, CIA, DoD, etc.- as an effort to combat a new type of villain who could produce massive damage with minimal risk. A new era of battles was fought on the digital theatre with an enemy into whose eyes we could not see; instead, we would catch a glimpse of the enemy through the zeros and ones of the virtual world (Grant Wernick, Feb 2019).”

Cybersecurity has become a significant concern in recent years. By looking at the current state of cybersecurity, leading industry analysts are forecasting this trend will continue for the near future depending on the sensitivity of the assets being compromised. The number of records exfiltrated, or the type of data that is breached, cyberattacks can easily cost enterprise-level organizations hundreds of millions of dollar in mitigations, legal cost, and business loss. As per one of the report, the cybercrime in its various forms is expected to cost the world more than **US$6 trillion** (Steve Morgan, 2016).

Cybersecurity factors can be categories into,

- Network Security,
- Application security
- Information security
- Operational security
- Disaster recovery and business continuity
- End-user education
As per the Forbes report, Cybersecurity threats affect all industries, regardless of size. The industries that reported the most cyber-attacks in recent years are healthcare, manufacturing, finance, and government (Steve, 2016).

“All the industries are highly integrated technology and required IT to run their business, and in results, information security is becoming a focal point for designing, developing and deploying software applications. Ensuring a high level of trust in the security and quality of these applications is crucial to their ultimate success. Information security has, therefore, become a core requirement for software applications, driven by the need to protect critical assets and the need to build and preserve public trust in computing” (Lynn and Rossouw, 2008).

Many of the security problems that people face today, such as security breaches and data theft, are caused by security vulnerabilities in application source code (Tyler, et.all, 2018).

The more technology-centric our organizations become and the faster they go, the higher the chance that a hacker will find that one vulnerability that will suck away all that hard-earned customer value (Jason Bloomberg, 2017).

Further, Prof Basie Von Solms in (2010) categories the development of Information Security from the early 1980s up to the present time into five different periods of waves (von Solms, 2010),

1. **First wave - The technical wave**
   *The First Wave was up to the early 1980s.*
   The First Wave of Information Security was dedicated to the mainframe environment, with dumb terminals and centralized processing (von Solms, 2010).

2. **Second wave - The management wave**
   *The Second Wave was from the early 1980s up to the middle 1990s.*
   The development of distributed computing, and later the Internet, WWW, and E-commerce lead to second wave (von Solms, 2010). It is characterized by a growing management realization of and involvement with the importance of information security, supplementing the Technical Wave (von Solms, 2000).

3. **Third wave - The institutional wave**
   *The Third Wave was from the middle 1990s up to about 2005.*
   The Third Wave consisted of the need to have some form of standardization of Information Security in a company, and aspects like best practices, certification, an Information Security culture and the measurement and monitoring of Information Security became important (von Solms, 2006, p165).

4. **Fourth wave - The Governance wave**
   *The Fourth Wave started about 2005* and is called the Information Security Governance Wave. This wave reflects the development of Information Security Governance as a result of the emphasis on good Corporate Governance. Therefore it is defined as the process of the explicit inclusion of Information Security as an integral part of good Corporate Governance, and the maturing of the concept of Information Security Governance (von Solms, 2006,p168).

5. **Fifth wave - The Cyber Security Wave**
   The Fifth Wave, which is called the Cyber Security Wave, started in about 2006.

   The Internet is debatably one of the greatest inventions ever developed by mankind, but it has brought with it extremely serious risks. Implementing any Internet-based system means announcing yourself to the rest of the world, thereby providing an opportunity for cybercriminals to attack the system. Cybercriminals are leveraging the growing use of the Internet by companies to deliver services to their clients to commit crime of immense proportions. Malware, phishing,
spoofing and other techniques used by such criminals are making the life for any Internet user extremely risky (von Solms, 2010, p5-6).

Additionally, Von Solms in (2010) mentioned in his paper, These 5 waves are not ‘blocks’ which started and then ended at a specific point in time – rather they represent new developments, which started in a certain period and placed new emphasis on aspects related to Information Security during the last 30 to 40 years, and should, therefore, be seen as existing in parallel with each other (von Solms, 2010, p2).

Overall, Cybersecurity is essential to govern the conducts and manners of interacting with computer systems from suspicious behavior. In a world where even our kitchen appliances and cars are connected to the internet, the opportunities are endless for cybercriminals to cause chaos (Mindcore, 2018).

Another aspect of dealing with cybercrime is to create a security culture within an organization.

Security Culture

As recorded under ENISA (2017): “Cybersecurity Culture (CSC) of organizations refers to the knowledge, beliefs, perceptions, attitudes, assumptions, norms and values of people regarding cybersecurity and how they manifest in people’s behavior with information technologies. Cybersecurity Culture is about making information security considerations an integral part of an employee’s job, habits, and conduct, embedding them in their day-to-day actions. Adopting the right approach to information security enables a resilient Cybersecurity Culture to develop naturally from the behaviors and attitudes of employees towards information assets at work (A.Martins and J.Eloff, 2002).

A successful Cybersecurity Culture shapes the security thinking of all staff (including the security team), improving resilience against all cyber threats, especially when initiated through social engineering (McKinsey, 2011), while avoiding imposing burdensome security steps that prevent staff from effectively performing their key business functions (Gerald V.Post and Albert Kagan, 2007).”

A more extensive definition for information security culture given by (Adèle da Veiga, 2008), “An information security culture is defined as the attitudes, assumptions, beliefs, values, and knowledge that employees/stakeholders use to interact with the organization’s systems and procedures at any point in time. The interaction results in acceptable procedures at any point in time. The interaction results in acceptable or unacceptable behavior (i.e. incidents) evident in artifacts and creations that become part of the way things are done in an organization to protect its information assets”.

According to A. Martins and J. Eloff (2006), Information security culture can be seen as the following:

- A set of information security Characteristics that the organization values,
- The assumption about what is acceptable and what is not in relation to information security,
- The assumption about what information security behavior is encouraged and what is not,
- The way people behave towards information security in the organization.

Prof Basie Von Solms (2000) suggested, "A culture of information security must be created in a company, by instilling the aspects of information security to every employee as a natural way of performing his or her daily job" (Von Solms, 2000).

Thomas Schlienger and Stephanie Teufel (2002), Also supported this “Security culture should support all activities in a way, that information security becomes a natural aspect in the daily activities of every employee” (Schlienger and Teufel, 2002, P-7).

To summarise this section, “Security culture is the behavior in an organization that contributes to the protection of data, information, and knowledge” (Dhillon, 1997).
A certain level of information security culture is already present in every organization using IT, but this culture could be a threat if it is not on an acceptable level. (A. Martins and J. Eloff, 2002, P-12).

2.2 Agile software development methodology

Agile is a conceptual framework of a group of software development methods to build software iterative and incrementally so that the development is aligned with the changing business needs. In agile software development methods, the ideas and solutions evolve through collaboration and cross-functional teams to deliver software faster and more comfortably.

The dictionary definition of ‘agile’ includes the ability to move quickly and is well coordinated. Alternatively, it can refer to the ability to think quickly, solve problems and have new ideas (dictionary, 2019).

According to Agile Alliance organization (agile alliance, 101): “Agile software development is an umbrella term for a set of frameworks and practices based on the values and principles expressed in the Manifesto for Agile Software Development and the 12 Principles behind it. Solutions evolve through collaboration between self-organizing, cross-functional teams utilizing the appropriate practices for their context.”

According to Jim and Alistair (2001), “the agile methods recognize people as the primary drivers of project success instead of practice they use in agile methods (Jim Highsmith, 2001”).

Agile software development methods deliver working software in smaller and more frequent increments by collaboration and the self-organization.

The name “agile” came about in 2001, it was created by seventeen of the leading thinkers in software development, with the goal of turning lightweight methods such as DP and Dynamic Systems Development Method (DSDM) into wider movement that could take on heavyweight software development process such as waterfall development and methodologies such as the rational unified process (Gene, et.al. 2016). The focal values honoured by the seventeen agilists are presented in the following (Pekka et al., 2002, P.98):

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

The author Georgios Papadopoulos did a study on the benefits from “Moving from traditional to agile software development methodologies also on large, distributed projects.” They provided evidence by the analysis of a case study that agile software development methodologies perform better than traditional methodologies in large, distributed projects. Improvements are observed on the quality and the customer perception of the end product, while agile methodologies allow for requirement changes even late in the project. At the same time, they also build better communication and collaboration in the team which results in enhanced relations between team members and improved employee satisfaction metrics (Georgios, 2014).

What makes a development method an agile one? (Pekka et al., 2002) This is the case when software development is,

- **Incremental** (small software releases, with rapid cycles)
- **Cooperative** (customer and developers continue working together with close communication)
- **Straightforward** (the method itself is easy to learn and to modify, well documented)
- **Adaptive** (able to make last moment changes)
There are several agile software development methodologies. However, scrum is one of the most popular frameworks for implementing agile. So popular that many people think Scrum and agile are the same thing. They are not. “Both Agile and Scrum are terms used in project management. The Agile methodology employs incremental and iterative work beats that are also called sprints. Scrum, on the other hand, is the type of agile approach that is used in software development. In other words, many frameworks can be used to implement agile, such as Kanban for example, but scrum has a unique flavour because of the commitment to short iterations of work (visual-paradigm)”.

![Figure 2 A Generic Agile Development Cycle (Source: visual-paradigm)](image)

Miller, the author of “The people, make the process: commitment to employees, decision making, and performance” (Miller, 2001) gives the following characteristics to agile software processes from the fast delivery point of view, which allow shortening the life cycle of projects:

1. Modularity on the development process level
2. Iterative with short cycles enabling fast verifications and corrections
3. Time-bound with iteration cycles from one to six weeks
4. Parsimony in development process removes all unnecessary activities
5. Adaptive with possible emergent new risks
6. Incremental process approach that allows functioning application building in small steps
7. Convergent (and incremental) approach minimizes the risks
8. People-oriented, i.e., agile processes favor people over processes and technology
9. Collaborative and communicative working style

Agile brought continuous improvements and an iterative approach to software development. The results were an increase in development throughput and improved customer satisfaction. However, development was still in a silo as were other stakeholders.

### 2.3 DevOps

DevOps is a software production practice build on the ideas of lean manufacturing and agile software development. DevOps aims to help fix the disconnect between software development activities (Planning,
design, development, and deployment) and attempts to introduce greater collaboration and automation technologies that work to streamline the process of creating software products (Ebert, et al., 2016)(Fitzgerald, et all, 2017)(Cois, 2014).

DevOps is a cultural and professional movement that stresses communication, collaboration and integration between software developers and IT operations professionals (Donna, 2018). The resulting improved workflow provides businesses the flexibility to change, and change quickly, without sacrificing the quality and reliability of their business services.

It is unions of People, Process and Technology to enable continuous delivery of value to the customers. DevOps is a term that stresses the strong interdependence between development and operations. DevOps is culture and movement that incorporates many different areas inside an organization such a sharing knowledge and ideas, working together as one team instead of separation, automating workflows and routine task amongst others (M.huttermann, 2012).

As per the definition of (Dyck et al. 2015) "DevOps is an organizational approach that stresses empathy and cross-functional collaboration within and between teams- especially development and IT operations-In software development organization, in order to operate resilient systems and accelerate delivery of changes."

DevOps is a movement- a new way of thinking and working; it is not a single standard, framework, a book or a method. It is primarily focused on cultural changes that highlight collaboration, communication, and interaction between the team, especially the development team with the operations team. Another aspect of DevOps is its focus on Quality assurance. In the minimum realization of the DevOps culture from a development perspective.

The author Gene Kim has mentioned in his book DevOps handbook;”” DevOps is the outcome of applying the most trusted principles from the domain of physical manufacturing and leadership to the IT value stream. DevOps relies on bodies of knowledge from Lean, Theory of constraints, the Toyota production system, resilience engineering, learning organization, safety culture, human factors, and many others. The foundation of DevOps can be seen as being derived from lean, the theory of constraints, and the Toyota kata movement; many also view DevOps as the logical continuation of the agile software journey that began in 2001” (Gene Kim, et al., 2016).

**Goals of DevOps:** Velocity, Quality & application performance.

“The objective of DevOps is to move beyond agile methodologies and enhance collaboration among developers and operations teams from application design through deployment with a heavy emphasis on automation that should improve deployment and time to market.”

DevOps embracing its philosophy of Teamwork, coordination, agility, shared responsibility and it is the pursuit of maximizing interoperability.

**DevOps values (CALMS):**

*John Willis and Damon Edwards In 2010, during the first DevOps day coined the DevOps value, Later the Jez Humble added ‘L’ for Lean in this (John Willis, 2012).*

**Culture:** It includes communication, collaboration, and behavior- how we work together.

**Automation:** Automation includes tools that enable the automation of tasks like testing and deploying software. Release management, configuration management, and monitoring and controls tools all enable automation.
**Lean:** Lean is focused on incremental improvements and splitting the work into small batches. The small batches allow us to release frequently as we develop. This is important so that we can get real user feedback on our work and learn from it so that we can pivot and adjust according to what you learn as needed (Stephanie Herr, 2018).

**Measurement:** If we cannot measure something, we do not know how it is performing, and we cannot improve it. A successful DevOps implementation will measure everything that is relevant.

**Sharing:** For DevOps to be successful, we need to create a culture of sharing—both ideas and problems. Sharing will improve communication and collaboration, and it will help the organization to learn and get better. Sharing can be thought of like a feedback loop.

DevOps, Group of super heroes of an organization, who, despite their differences work together towards shared vision and common purpose.

DevOps is not a framework or methodology in and of itself. It does not stand-alone. DevOps adopts and leverages multiple frameworks and methodologies such as agile, lean, and ITSM.

**DevOps Principles:**

DevOps principles are built on three ways to frame the processes, procedures, and practices of DevOps, as well as the prescriptive steps.

1. **The First way - (Flow)**
   
   *The first way enables fast left-to-right flow of work from Development to Operations to the customer* (Gene Kim, et al., 2016, p11).

2. **The Second Way - (Feedback)**
The second way enabled the fast and constant flow of feedback from right to the left at all stage of software delivery value stream (Gene Kim, et al., 2016, p12).

3. The Third Way- (Continual experimentation and learning)

The third way enables the creation of a generative, high-trust culture that supports a dynamic, disciplined, and scientific approach to experimentation and risk-taking, facilitating the creation of organizational learning, both from our successes and failure (Gene Kim, et al., 2016, p12-13).

Below mentioned few of the vital DevOps practices to enable IT organization to deliver high quality to the customer.

Continuous Integration

A development practice that requires developers to integrate code onto a shared repository on a daily basis.

Continuous integration (CI) represents a paradigm shift. Without continuous integration, software is broken until somebody proves it works, usually during a testing or integration stage. With continuous integration, software is proven to work (assuming a sufficiently comprehensive set of automated tests) with every new change—and team know the moment it breaks and can fix it immediately. The teams that use continuous integration effectively are able to deliver software much faster, and with fewer bugs, than teams that do not (Humble Jez, and David Farley, 2010).

Continuous Delivery

A methodology that focuses on making sure software is always in a releasable state throughout its lifecycle.

Continuous Delivery (CD) means to deploy new software to production, with the differing factor from traditional software deployment being the frequency of deployment, which can happen multiple times every day (Claps, et.al, 2014). “Continuous delivery enables businesses to reduce cycle time so as to get faster feedback from users, reduce the risk and cost of deployments, get better visibility into the delivery process itself, and manage the risks of software delivery more effectively”(Jez Humble and Joanne Molesky, 2011).

Continuous Deployment

Continuous deployment is the next step of continuous delivery (Carl Caum, 2013 ). It is the process that takes validated Features from a staging environment and deploys them into the production environment, where they are readied for release (scaledagileframework, 2018).
CHAPTER 2. LITERATURE REVIEW

Continuous deployment eliminates the human safeguards against unproven code in live software. It should only be implemented when the development and IT teams rigorously adhere to production-ready development practices and thorough testing, and when they apply sophisticated, real-time monitoring in production to discover any issues with new releases (Margaret Rouse, 2018).

2.5 Agile Development and traditional Cybersecurity in DevOps: Challenges

Agile software development provided speed to development and collaboration with the customer, but somehow it left Operations (Ops) to be included in workflow and this disconnect lead to conflict and inefficiency. This conflict resulted in slow time to market, longer deployment cycle and technical debt and fragile application.

According to the founder of StreamStep Clyde Logue, “Agile was instrumental in development regaining the trust in the business, but it unintentionally left IT operations behind. DevOps is a way for the business to regain trust in the entire IT organization as a whole (Aymen El Amri, 2016)”.

To break down the current silos between Development and Operations (Ops), DevOps arose from new reality; and bridges the gap between Dev and Ops team by providing a culture/process movement.

However, this process fails to include security into the work of Agile-DevOps workflow also, which leads to security team hiding somewhere in a cave without knowing business needs.

Current DevOps practices include security checks at the end of the agile development process; this resulted in a break on faster delivery of product to the customers. Besides, with this workflow approach, one of the most significant issues for organizations today is how they can defend themselves from potential cyber-attacks. The cost of fixing security issues and uncovered bugs grow as the development cycle advances.

Most organizations fail to integrate their security programs into their development efforts; they are still using traditional, manual security methods profoundly impairs the speed and agility of DevOps, resulting in lengthy security compliance activities and many vulnerabilities late in the delivery life cycle (Alan Crouch, 2017). This means security methods have to be more agile, and these agile security methods have to be understood by security teams and accepted by development teams to make sure they contribute meaningfully to the DevOps movement without hampering their development speed and service delivery (M.Goldschmidt and M. McKinnon, 2016)(Myrbakken H. and Colomo-Palacios R., 2017).

In the current state, some of DevOps enable organization has integrated security program with their workflow, but they are still struggling to bring this into reality to cope with the speed of Agility of DevOps and empower security, as shown in Fig.6.

The motivation of work of this thesis is the outcome of the survey conducted by an “information technology research and advisory company- 451 research” (DevSecOps Realities and Opportunities, 2018) and published in April 2018. The most important question and the driving force for this research is:
What are the most significant application security testing challenges inherent in continuous integration/continuous delivery (CI/CD) workflows?

1. The most critical issue identified was a lack of automated, integrated security tools for CI/CD (61%).
2. (56%) respondent “Inconsistent approach” as the next most significant challenge.
3. Almost half ~48% respondent that security slow the release process down.
4. Another critical challenge highlighted above is false positive (46%); the respondent suggested that a higher number of false positive can drown out the benefits security scan in CI/CD pipeline.
5. Also, 36%, Developer resistance to implement security with CI/CD pipeline.

Further mentioned in the surveyed, Challenges of integrating security into CI/CD release processes center on a lack of automated, integrated security tools, inconsistency and the noise of false positives. These are real issues for enterprise organizations (DevSecOps Realities and Opportunities, 2018).

2.6 Summary

In today’s cyberspace that created a data-driven world, information security is top priorities for companies across industries. Without having safeguard, the smart and bad person can enter into our cyberspace to breaches the data and can damages organization resources even extreme to physical damage.

As von Solms (2010) mentioned in Fifth Wave - The Cybersecurity Wave, The Internet has handed the criminal side an extremely useful way of committing their crimes, and to us, our greatest challenge – to ensure that such crimes are prevented from happening (von Solms, 2010, p5).

Agile brought continuous improvement and an iterative approach to software development. The result was an increase in development throughput and improved customer satisfaction. However, development was still in a silo as were other stakeholders. Today’s, in most organizations, it is mainly applied in Dev.

The DevOps practices automated the delivery process of agile development teams, with a focus on processes and automated tools. The DevOps 3 ways and collaborative culture shift teardown the organizational silos and help to bring more features and improvements to market faster (Vikas Hazrati, 2010) by emphasizes on continuous deployment, resulting in a significant improvement to development output, better collaboration between teams, faster time to market, improved overall productivity, and enhanced customer satisfaction, to name a few (Mark Robinson, 2019). However, in the fast-moving delivery process, they kept the security at the end of the agile development process.

Mark Robinson (2019) further mentioned in his blog post that, what good will all of these positives benefits of DevOps shift do for the company if team is not prioritizing security. Focusing on leveraging DevOps to
improve delivery workflow while ignoring security issues is like trying to push water uphill with a rake (Mark Robinson, 2019).

Therefore, the question is,

Is there a way for DevOps enable organization to embrace security without affecting agility?

YES, the secret to this question is, use the continuous deployment of DevOps practice to enable continuous security by,

- Automated security checks at each stage of agile delivery.
- Shift the security to the left into delivery pipeline
- Make everyone responsible for security and
- Set of Security checks toolchain

this thesis work “Continuous security in DevOps environment” will empower the security team to come out of their cave and guide the application, development, and operation team to deliver the secure product without affecting time to market.

Co-founder of rugged software, Joshua Corman says, “Developers write code assuming the only task is to make it perform a function. However, that can lead to programs riddled with vulnerabilities that can, in turn, lead to economic damages, lost data and lost productivity (Tim, 2010)”. Therefore, the solution is to embrace DevOps CI/CD practice to integrate security checks into the daily work of development and operations, so that security is part of everyone’s job, every day.

In the current state, IT organization are still facing Challenges of integrating security into CI/CD release processes center on a lack of automated, integrated security tools, inconsistency and the noise of false positives (DevSecOps Realities and Opportunities, 2018).

In Chapter 3, This thesis used DevOps practice of continuous delivery pipeline to design secure seven-stage agile development process; each stage is baked with automated security checks, best practices and Tools to reduce security risk, rather than added as a layer on top of the agile delivery process. Moreover, this will be the goal of the thesis to achieve “Continuous Security in DevOps environment” by Integrating automated security checks at each stage to get the ideas securely from whiteboard to experimentation.

According to Bruce Schneier (2000), “Security is a process, not a product” (Bruce Schneier, 2000).

In the end, Continuous security practice will not only improve the security of products but also will create processes that are easier to audit and that attest to the effectiveness of controls.

As stated by Dale Walker and Clare Hopping : “Cybersecurity is not merely finding a solution to a problem, or indeed the problem itself. Cybersecurity is the process through which your business should go through in order to protect itself against evolving threats. This does include the tools and technologies needed to fight security threats, and also to maintain compliance, but it also includes the processes through which everyone in your organization should go through in order to make sure nothing slips through the cracks” (Dale Walker and Clare Hopping, 2019).
Chapter 3 Continuous Security workflow: Solution to secure DevOps environment

DevOps provides an opportunity to reduce security risk if security is integrated into the continuous delivery/deployment pipeline using Continuous loop of iterations process.

The primary objective of this chapter is to present the workflow of continuous security by integrating security controls, compliance, and automated checks seamlessly into DevOps culture so that developers never have to leave their continuous integration or continuous deployment toolchain environment.

Continuous security assurance process to be integrated from the beginning of the Security development life cycle (SDLC) life cycle and assessed with each new iteration by providing a feedback loop.

To outline the security controls and practices to embed in DevOps to achieve continuous security assurance, the thesis has considered SEVEN stages of agile software delivery workflow as shown below diagram.

![Figure 7 Stages in DevOps to infuse for Continuous security assurance](image)

The embedded security controls and practices used in the above seven stages are categories into In-line activities and out-of-band activities.

**In-line activities:** Completely automated activities and embedded into CI/CD pipeline without any manual intervention.

**Out-of-band activities:** Activities that are driven by the team process and cannot be completely automated.

<table>
<thead>
<tr>
<th>Stage/Gate</th>
<th>Activities Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>out-of-band activities</td>
</tr>
<tr>
<td>Plan</td>
<td></td>
</tr>
<tr>
<td>Develop</td>
<td>In-line activities and out-of-band activities</td>
</tr>
<tr>
<td>Build</td>
<td>In-line activities</td>
</tr>
<tr>
<td>Test</td>
<td></td>
</tr>
<tr>
<td>Deploy</td>
<td></td>
</tr>
<tr>
<td>Monitor</td>
<td>In-line activities</td>
</tr>
</tbody>
</table>

16
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

Table 1: category of activities into DevOps Life cycle stages

While defining and describing the various activities of integrating security, the workflow checks should be transparent to all stakeholder and preserve the agility and speed of DevOps.

The SEVEN stages workflow shown above (Fig. 7) and discussed in details below section focuses on how continuous security assurance workflow can transform an application's development cycle to create more secure application.

However, before this chapter design secure stage of continuous security workflow it is essential we should define security check toolchain, which will be used during security checks at each step of seven stages.

Continuous security toolchain

In software, a toolchain is the set of programming tools that is used to perform a complex software development task or to create a software product, which is typically another computer program or a set of related programs (Wikipedia, 2019).

In general, the tools forming a toolchain are executed consecutively so the output or resulting environment state of each tool becomes the input or starting environment for the next one, but the term is also used when referring to a set of related tools that are not necessarily executed consecutively (Wikipedia, 2019)(elinux, 2013)(nongnu, 2012)(Imran Saed, et.,all , 2015).

According to author Jen Andre, “Machines or tools are fantastic at handing a series of repetitive tasks, while humans are great at the deriving context from data. Why not offload this repetitive task of security validation to machines and allow humans to focus on innovation and improvement?” (Jen Andre, 2018).

A toolchain in this paper has been constructed by integrating a set of tools and practices to implement continuous security workflow in DevOps environment, a full list of continuous security toolchain used in this chapter is listed under chapter 6 for a reference.

It uses the below criteria while finalizing the list of toolchain and practices to get the continuous security rolling in DevOps environment.

1. Selected tool for security validation within the workflow should exposed full functionality via APIs or command line.
2. Telemetry from production security controls is delivered back to development teams to inform application update.
3. Chosen practices should be industry best practices.
4. Selected security controls are supporting automation activities so as not to impede DevOps agility. Therefore, it must be programmable and integrated into the CI/CD pipeline.
5. Seamless integration with existing DevOps environment.
6. Selected security controls/Tools should be easily integrated within Unified Security Dashboard. In addition, respect bug tracking to record any security events for the team to prioritize.
7. Single source of truth for all security controls/tools.
8. Integrated security workflow should bring security right into the engineer’s world by integrating tools into the engineer’s world.

Continuous security workflow

Each stage of continuous security workflow baked with security controls and tools are explained in details in below sections.
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

3.1 Requirements

Security requirement stage, the first step in the Continuous Security workflow is a goal set out for an application.

To explain this in a non-technical word, requirements are about defining what under validation can do or be and what level of security is expected from the system concerning a known and unknown threat, malicious attack.

The requirements defined for an organization should be clear, consistent, testable, and measurable to deploy secure software effectively.

Any organization wants to ensure their security baseline is stable then they have to incorporate security requirement as the first step in the workflow.

![Figure 8 Security Requirement in continuous security](image)

**Purpose:**

- Meet the security goals for an application
- To build appropriate security controls to protect digital assets with the balance between security quality and time-to-market.
- Help an organization to build consensus among various stakeholders (Development teams, Operations team, Security teams, management, Auditors and so on). Moreover, to resolve a conflict of interest among various stakeholder, for example; business teams would like to get the product into the market as soon as possible even though there are security defects, while the security team may not endorse the product release (Tony Hsu, 2018).
- Create a foundation for creating secure software.

To create a baseline of Security requirement standard at this stage of Continuous Security workflow, we will consider three aspects of security requirements defined by “Owasp Application Security Verification Standard” as summarised below.
Above listed requirements can be used by the DevOps team as well as by the consumer to define what a secure application is.

### 3.1.1 Security Requirements for web applications

To validate web application security requirement, we will consider OWASP Application Security Verification Standard (ASVS), as this provides a basis for testing web application technical security controls, as well as any technical security controls in the environment, that are relied on to protect against vulnerabilities such as Cross-Site Scripting (XSS) and SQL injection.

In this paper, we will use the OWASP ASVS standard to establish a level of confidence in the security of Web applications.

Additionally, ASVS can be used by the developer as a list of security requirements, and by QA team as a checklist to do verifications and assess the security level of the applications.

OWASP Application Project source is available online free to use.

### 3.1.2 OWASP Application Security Verification Standard (ASVS)

At the time of writing this thesis and during the implementation phase the version 3.0 were available for the public use.

“ASVS version 3 focuses on “what” to verify, and leaves “how” to verify it to the developers. As such, the Application Security Verification Standard can simply be a yardstick for developers and application owners to determine what degree of confidence they have in the security of a particular web app. It can likewise guide developers around what to build into security controls to meet specific security requirements. Finally,
organizations can use it during the procurement process as a basis for specifying web app security verification requirements in contracts” (Bhaumik Shah, 2017).

3.1.2.1 Security requirement defined by OWASP ASVS
The OWASP Application Security Verification Standard (ASVS) defines three levels of the security requirement,

Summaries below diagram showing OWASP ASVS requirement mapping concerning a visual representation of simple web architecture:

![Figure 10 OWASP ASVS levels for security requirement](image1)

![Figure 11 OWASP ASVS requirement mapping](image2)
Security requirement used during development of a web application project, defined by “Owasp Application Security Verification Standard 3.0” is available under Appendix-C.

While working on defining Security requirement standards for a web application, it was experienced that few of the card gamification is very useful to promote awareness of security controls and risks and identify website security requirement among DevOps drove team members.

It is worth sharing in this paper:

**OWASP Cornucopia**
A card game to assist the DevOps team in identifying security requirements in agile development.

**OWASP SnakesAndLadders**
To promote awareness of application security controls and risks among DevOps team members.

### 3.1.3 Security requirements for Release gate

The continuous security workflow during the lifecycle of application eventually will pass through multiple release gates, and it is very critical to set up security quality criteria at each release gates. The software quality criteria at each release gate will provide sufficient input necessary for deciding for a go/no go. The object of this gate is to improve the quality instead of pinpointing the issues.

To make this practice sustainable, it has been found during the implementation that team should first start with high-priority security criteria and then iteratively add few more security criteria at each stage of release gate. Security team can play a very vital role as a coach for the DevOps team to support and help the team to mitigate security issues and meet a higher standard of security set by organizations.

This thesis will also discuss various Security release gate used for each stage of continuous security workflow in details during subsequent section.

However, the team has to collaboratively certify the security of an application once it passes through each stage of release gate and It has been noticed that most of the time during the point of release review there is an argument within the team over the decision to move to the next stage or not. For example, a development team may think it is a ‘minor issue to proceed to the next stage, while the operation team may consider it a high-risk issue’ (Tony Hsu, 2018).

To handle this situation, we have used the Common Vulnerability Scoring System (CVSS) metrics to measure the risk associated with any vulnerabilities found at each stage of the release gate.

As Peter Drucker stated, “If you cannot measure it, you cannot improve it.”

### 3.1.4 Common Vulnerability Scoring System (CVSS) metrics

Common Vulnerability Scoring System (CVSS) is an industry standard for assessing security vulnerabilities. Using CVSS organization can evaluate the degree of risks posed by found vulnerabilities so that mitigation efforts can be planned and prioritized.

Mainly provides a way for an organization to capture the principal characteristics of vulnerability, and produce a numerical score reflecting its severity. The numerical score can then be translated into a qualitative representation (such as low, medium, high, and critical) to help organizations(CVSS, v3.0)
accurately assess and prioritize their vulnerability management processes as defined by FIRST.Org, a US-based non-profit organization.

Summarized below is CVSS affords there essential benefits which are documented by First.org (CVSS, v3.0):

1. It provides standardized vulnerability scores. When an organization uses a standard algorithm for scoring vulnerabilities across all IT platforms, it can leverage a single vulnerability management policy defining the maximum allowable time to validate and remediate a given vulnerability.
2. Next, it provides an open framework. Users may be confused when a vulnerability is assigned an arbitrary score by a third party. With CVSS, the individual characteristics used to derive a score are transparent.
3. Finally, CVSS enables prioritized risk. When the environmental score is computed, the vulnerability becomes contextual to each organization and helps provide a better understanding of the risk posed by this vulnerability to the organization.

The Base metric group represents the intrinsic characteristics of a vulnerability that are constant over time and across user environments. It is composed of two sets of metrics: the Exploitability metrics ((CVSS, v3.0) (Attack vector, Attack Complexity, Privileges required, and User Interaction) and the Impact metrics (Confidentiality, Integrity, Availability) (CVSS, v3.0).

The Temporal metric group reflects the characteristics of a vulnerability that may change over time but not across user environments. For example, the presence of a simple-to-use exploit kit would increase the CVSS score, while the creation of an official patch would decrease it (CVSS, v3.0).

The Environmental metric group represents the characteristics of a vulnerability that are relevant and unique to a particular user’s environment. These metrics allow the scoring analyst to incorporate security controls which may mitigate any consequences, as well as promote or demote the importance of a vulnerable system according to her business risk (CVSS, v3.0).

To cover the essential metrics group (base metric group) to evaluate a security issue by answering the following question under Base metric group.

<table>
<thead>
<tr>
<th>Base Metric group</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack Vector(AV)</td>
<td>Reflects the path by which vulnerability exploitation is possible by an attacker.</td>
</tr>
</tbody>
</table>
### CVSS Basic Metric Group's overview (CVSS, v3.0)

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack Complexity (AC)</td>
<td>Metric describes the conditions beyond the attacker's control that must exist in order to exploit vulnerability:</td>
</tr>
<tr>
<td></td>
<td>- Low</td>
</tr>
<tr>
<td></td>
<td>- High (more work for the attacker)</td>
</tr>
<tr>
<td>Privileges Required (PP)</td>
<td>Metric describes what level of privileges an attacker must possess before successfully exploiting the vulnerability:</td>
</tr>
<tr>
<td></td>
<td>- None</td>
</tr>
<tr>
<td></td>
<td>- Low</td>
</tr>
<tr>
<td></td>
<td>- High</td>
</tr>
<tr>
<td>User Interaction (UI)</td>
<td>Whether the attacker will need another user to participate in the attack for it to succeed.</td>
</tr>
<tr>
<td></td>
<td>- Not Required</td>
</tr>
<tr>
<td></td>
<td>- Required</td>
</tr>
<tr>
<td>Scope(s)</td>
<td>An idea of how badly an exploited vulnerability can impact other components or resources beyond the privileges directly associated with it.</td>
</tr>
<tr>
<td></td>
<td>- Unchanged</td>
</tr>
<tr>
<td></td>
<td>- Changed</td>
</tr>
<tr>
<td>Confidentiality (C)</td>
<td>How much authority an exploited vulnerability provides. Will any confidential information is stolen?</td>
</tr>
<tr>
<td></td>
<td>- None</td>
</tr>
<tr>
<td></td>
<td>- Low</td>
</tr>
<tr>
<td></td>
<td>- High</td>
</tr>
<tr>
<td></td>
<td>Confidentiality refers to limiting information access and disclosure to only authorized users, as well as preventing access by, or disclosure to, unauthorized ones.</td>
</tr>
<tr>
<td>Integrity (I)</td>
<td>This metric measures the impact on the integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of information and how much data corruption an exploited vulnerability makes possible,</td>
</tr>
<tr>
<td></td>
<td>- None</td>
</tr>
<tr>
<td></td>
<td>- Low</td>
</tr>
<tr>
<td></td>
<td>- High</td>
</tr>
<tr>
<td>Availability (A)</td>
<td>Availability metric measures the impact on the availability of the impacted component resulting from a successfully exploited vulnerability.</td>
</tr>
<tr>
<td></td>
<td>- None</td>
</tr>
<tr>
<td></td>
<td>- Low</td>
</tr>
<tr>
<td></td>
<td>- High</td>
</tr>
</tbody>
</table>

CVSS project has provided an online calculator (CVSS Calculator, V3.0) to calculate a score against vulnerabilities. A vulnerability is typically given a base score in CVSS, which is a rating from zero to 10.
that gives an idea of how easy it is to exploit a vulnerability and how damaging it can be. Some vulnerabilities are also given temporal and environmental scores which modify the base score, but many are not (CVSS Scores, 2018).

<table>
<thead>
<tr>
<th>Rating</th>
<th>CVSS Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.0</td>
</tr>
<tr>
<td>Low</td>
<td>0.1 - 3.9</td>
</tr>
<tr>
<td>Medium</td>
<td>4.0 - 6.9</td>
</tr>
<tr>
<td>High</td>
<td>7.0 - 8.9</td>
</tr>
<tr>
<td>Critical</td>
<td>9.0 - 10.0</td>
</tr>
</tbody>
</table>

The base score provides a metric that is reasonably accurate and easy to understand (what information the score is conveying). However, to get the more deep information about the vulnerabilities then it is recommended to compliment CVSS scores with Threat intelligence. Threat intelligence would provide a more profound understanding of how and why threat actors are targeting specific vulnerabilities and ignoring others.

3.1.5 Benefits of having Security requirement for a base of our Continuous security workflow
- Providing confidence that software is secure when requirements are linked to verification
- Making security measurable
- Improve compliance
- Provide a blueprint for software development
- Making sure that confidential communications and data are kept private
- A unified metric standard for vulnerabilities
- Having this in place will ensure that-
  - Unauthorized malicious programs do not infect the application or component
  - Communications and data are not intentionally corrupted
  - System (people and application) are protected against destruction, damage, theft, or surreptitious replacement

Not incorporating the requirement stage in continuous security inevitably results in insecure software. Since software development follows a series of processes (the blueprint), it is of paramount importance that security requirements are determined alongside the functional and business requirements (Puneet Mehta, 2010).

It has been observed that the majority of projects started without having Security requirement stage and directly jump to the planning phase and those same applications last struggle to keep security issues at bay.

Before we move to the next stage, I would like to state a famous Chinese proverb:

“The best time to plant a tree was 20 years ago. The second best time is now” – Chinese Proverb
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

3.2 Plan
Security begins at planning before a single line of code is developed. Security Plan is aimed at reducing the risk and threat to the company’s sensitive, personal data. Planning stage in Continuous security model can mitigate security threats against an organization, as well as help to protect the integrity, confidentiality, and availability of data. Plan stage consists of secure architect and design.

The Security Architecture and Design domain contain the concepts, principles, structures, and standards used to design, implement, monitor, and secure operating systems, equipment, networks, applications, and those controls used to enforce various levels of confidentiality, integrity, and availability.

In simple word, Security Design is “Set of concept or principles,” and Security architecture is “Sets of resources or frameworks to implement the concept (design)”

3.2.1 Security Design principle and explanation
As per Saltzer and Schroeder’s 1975 Design Principles (Richard, 2012)

<table>
<thead>
<tr>
<th>Principle</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open design</td>
<td>The design should not be secret</td>
</tr>
<tr>
<td>Fail-safe defaults</td>
<td>Base access decisions on permission rather than exclusion</td>
</tr>
<tr>
<td>Least privilege</td>
<td>No more privileges than what is needed.</td>
</tr>
<tr>
<td>Economy of mechanism</td>
<td>Keep the design as small and straightforward as possible.</td>
</tr>
<tr>
<td>Separation of privileges</td>
<td>Where possible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.</td>
</tr>
<tr>
<td>Complete mediation</td>
<td>Every access to every object must be checked for authority.</td>
</tr>
<tr>
<td>Least common mechanism</td>
<td>Minimize the amount of mechanism common to more than one user and depended on by all users.</td>
</tr>
<tr>
<td>Psychological acceptability</td>
<td>It is essential that the human interface is designed for ease of use so that users routinely and automatically apply the protection mechanisms correctly.</td>
</tr>
<tr>
<td>Work factor</td>
<td>Compare the cost of circumventing the mechanism with the resources of a potential attacker.</td>
</tr>
</tbody>
</table>
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

3.2.2 Security Properties and description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality</td>
<td>Data is only available to the people intended to access it.</td>
</tr>
<tr>
<td>Integrity</td>
<td>Data and system resources are only changed in appropriate ways by appropriate people.</td>
</tr>
<tr>
<td>Availability</td>
<td>Systems are ready when needed and perform acceptably.</td>
</tr>
<tr>
<td>Authentication</td>
<td>The identity of users is established.</td>
</tr>
<tr>
<td>Authorization</td>
<td>Users are explicitly allowed or denied access to resources.</td>
</tr>
<tr>
<td>Nonrepudiation</td>
<td>Users cannot perform an action and later deny performing it.</td>
</tr>
</tbody>
</table>

Above listed table was presented by author Fotios (Fotis) Chantzis (Fotis, 2018). Secure architecture design is more about the security controls of whole systems, which includes:

**Security controls**

- Security by Design
- Security by Principle

### Security by Design
- Unauthorized access to the system.

According to OWASP, security by design principles are the following:
- Minimize attack surface area
- Establish secure defaults
- The principle of least privilege
- The principle of defense in depth
- Fail securely
- Do not trust services
- Separation of duties
- Avoid security by obscurity
- Keep security simple
- Fix security issues correctly

### Privacy by Design
- An authorized process of privacy data.

Referring to the OECD Privacy Principles, the term privacy by eight principles define the design:
- Collection Limitation Principle
- Data Quality Principle
- Purpose Specification Principle
- Use Limitation Principle
- Security Safeguards Principle
- Openness Principle
- Individual Participation Principle
- Accountability Principle

<table>
<thead>
<tr>
<th>Primary concern</th>
<th>Security by Design</th>
<th>Privacy by Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principal</td>
<td>Unauthorized access to the system.</td>
<td>An authorized process of privacy data.</td>
</tr>
<tr>
<td></td>
<td>According to OWASP, security by design principles are the following:</td>
<td>Referring to the OECD Privacy Principles, the term privacy by eight principles define the design:</td>
</tr>
<tr>
<td></td>
<td>• Minimize attack surface area</td>
<td>• Collection Limitation Principle</td>
</tr>
<tr>
<td></td>
<td>• Establish secure defaults</td>
<td>• Data Quality Principle</td>
</tr>
<tr>
<td></td>
<td>• The principle of least privilege</td>
<td>• Purpose Specification Principle</td>
</tr>
<tr>
<td></td>
<td>• The principle of defense in depth</td>
<td>• Use Limitation Principle</td>
</tr>
<tr>
<td></td>
<td>• Fail securely</td>
<td>• Security Safeguards Principle</td>
</tr>
<tr>
<td></td>
<td>• Do not trust services</td>
<td>• Openness Principle</td>
</tr>
<tr>
<td></td>
<td>• Separation of duties</td>
<td>• Individual Participation Principle</td>
</tr>
<tr>
<td></td>
<td>• Avoid security by obscurity</td>
<td>• Accountability Principle</td>
</tr>
<tr>
<td></td>
<td>• Keep security simple</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Fix security issues correctly</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security Controls</th>
<th>Security by Design</th>
<th>Privacy by Design</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Access control</td>
<td>• Cookie</td>
</tr>
<tr>
<td></td>
<td>• Unsuccessful login attempts</td>
<td>• Anonymity</td>
</tr>
<tr>
<td></td>
<td>• Session control</td>
<td>• Consent</td>
</tr>
</tbody>
</table>
Team can use a list of security techniques provide by OWASP Top Ten Proactive Controls 2018 to helps developers build secure software and to develop defensive techniques and controls in an application. These below-mentioned techniques should be applied proactively at the early stages of continuous security model to ensure maximum effectiveness. In addition, the best place to integrate this would be at the Plan stage.

### 3.2.3 OWASP top 10 proactive controls for secure architecture and design

OWASP has defined Top 10 proactive controls and it most to be followed by all information technology organization (OWASP TOP 10, 2018).

<table>
<thead>
<tr>
<th>Proactive controls</th>
<th>Security references and tools</th>
</tr>
</thead>
</table>
| C1: Define Security Requirements | • OWASP Application Security Verification Standard (ASVS)  
• OWASP Mobile Application Security Verification Standard (MASVS)  
• OWASP Top 10                                                               |
| C2: Leverage Security Frameworks and Libraries | • OWASP Dependency Check  
• OWASP Dependency Track                                                                                   |
| C3: Secure Database Access       | • OWASP Cheat Sheet: Query Parameterization  
• Bobby Tables: A guide to preventing SQL injection  
• CIS Database Hardening Standards                                                                               |
| C4: Encode and Escape Data       | • OWASP Cheat Sheet: XSS Prevention - Stopping XSS in  
• OWASP Cheat Sheet: DOM-based XSS Prevention  
• OWASP Cheat Sheet: Injection Prevention                                                                  |
| C5: Validate All Inputs          | • OWASP Cheat Sheet: Input Validation                                                                                      |
| C6: Implement Digital Identity   | • NIST Special Publication 800-63 Revision 3 - Digital Identity Guidelines                                        |
| C7: Enforce Access Controls      | • Omada  
• RSA  
• OWASP ZAP                                                                                                      |
| C8: Protect Data Everywhere     | • OWASP Cheat Sheet: Transport Layer Protection  
• Ivan Ristic: SSL/TLS Deployment Best Practices  
• OWASP Cheat Sheet: HSTS                                                                                         |
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

- OWASP Cheat Sheet: Cryptographic Storage
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: IOS Developer - Insecure Data Storage
- OWASP Testing Guide: Testing for TLS

| C9: Implement Security Logging and Monitoring | OWASP AppSensor Detection Points
 | OWASP Log injection
 | OWASP Log forging
 | OWASP Code Review Guide |

| C10: Handle All Errors and Exceptions | CWE 209: Information Exposure Through an Error Message
 | CWE 391: Unchecked Error Condition
 | Error-Prone
 | Chaos Monkey |

Table 7 Proactive controls

3.2.4 Data Masking For Privacy (DMFP)

Data masking is a method of creating a structurally similar but inauthentic version of an organization's data that can be used for purposes such as software testing and user training (Margaret Rouse, 2009).

Data masking is a very critical privacy process or method and must be part of the security planning stage. Using Data masking process, an organization can obfuscate sensitive data to protect it.

Data masking process is needed to:

- Make production data available in the test and development environment to allow the DevOps team to test upgrades, patches, and fixes.
- To build and test the new features for customers ensuring existing functionality does not break we have to impersonate development environment close to that of the production environment to achieve the desired production like behavior.

Find below two common technique to achieve data masking during implementation,

<table>
<thead>
<tr>
<th>Category</th>
<th>Key Difference</th>
<th>Sub-category</th>
<th>Technique</th>
<th>Applications scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymization</td>
<td>Anonymization is the destruction of</td>
<td>Randomization</td>
<td>Noise addition</td>
<td>Numeric data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Permutation</td>
<td>Numeric data needs to be reversible</td>
</tr>
</tbody>
</table>
3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

<table>
<thead>
<tr>
<th></th>
<th>Different Privacy</th>
<th>Big data statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generalization</td>
<td>Aggregation &amp; K-anonymity</td>
<td>Big Data Statistics</td>
</tr>
<tr>
<td></td>
<td>L-diversity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T-closeness</td>
<td></td>
</tr>
<tr>
<td>Pseudonymization</td>
<td>Encryption</td>
<td>Data needs to be reversible</td>
</tr>
<tr>
<td></td>
<td>Hash</td>
<td>Fixed length value</td>
</tr>
<tr>
<td></td>
<td>Tokenization</td>
<td>Financial sector to replace card ID number</td>
</tr>
</tbody>
</table>

Table 8 Different type of Data masking and differences

3.2.5 Threat Modeling

Threat modeling is a security-focused design activity; it has proven to be one of the best return on investment activities for identifying and addressing design flaws before their implantation into code.

As Adam Shostack mentioned in his book, “Threat modeling is about using models to find security problems. Using a model means abstracting away many details to provide a look at a bigger picture, rather than the code itself. You model because it enables you to find issues in things you have not built yet and because it enables you to catch a problem before it starts. Lastly, you threat model as a way to anticipate the threats that could affect you” (Adam Shostack, 2014).

We will use threat-modeling practice to develop a shared understanding of the product’s system design and service architecture to discover and correct design-level security problems. This understanding will help us to be proactively in threat mitigation or at least prioritize it using unified bug tracking tool.

Threat modeling activities is an essential part of the agile software Development Lifecycle.

To optimize and achieve maximum business security effort that will lead us to effective communication and collaboration across the DevOps team, we can use threat modeling across software development and operations.

Application threat modeling: Application threat modeling should focus solely on the application for which it is created.

Operational threat modeling: OTM (Operation thread modeling) covers infrastructure components such as servers, databases, DMZ, Firewall, and load balancer. It help the DevOps team to mitigate inherent threat in the infrastructure of business while aligning with business strategy and budgeting.

The goals of the threat modeling activities are to improve the,
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

- security of designs,
- To document the security design activity, and
- To teach about security as people work through the process.

**Microsoft Threat modeling tool** will be used to achieve the desired secure design model for the application and operational level. The result of Threat model will be shared in the Unified Security Dashboard (USD) as well as in unified Bug tracking tool.

Threat modeling practice helps us to document the following four key questions:

**Diagram**
- What are we building?

**Identify Threats**
- What can go wrong?

**Mitigate**
- What are we doing to defend against threat?

**Validate**
- Did we do a good job?
- Validate Steps 1-3

*Figure 17 Thread modeling questions*

### 3.2.5.1 What are we building (Diagram)?
To answer this question, we will use the Data flow diagram (DFD) using the Microsoft threat model tool. To draw what we are working on, during this process all team members can learn how a system works.

We have created a DFD diagram of a simple application as shown in below screenshot,

*Figure 18 Data flow diagram-using threat modeling tool*

### 3.2.5.2 What can go wrong (Identify Threats)?
This is a bit of "research" activity during the activities we can find the main threat that applies to our application referred to above DFD.

To approach this question, we will use STRIDE structure, which is an acronym for **Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege** then we will maps threats to the properties that guard against them.
During the continuous security workflow, we will use the Threat modeling tool to identify these threats, as shown in the below screenshot:

![Threat Category in the threat-modeling tool](image)

We have also used EoP(Elevation of Privilege) card game to find threats in the diagram drawn previously. It is a good team game to get team members to familiarise with different types of threat can existing within our application design.

To better formulate the question (Identify Threats) let me elaborate this using the STRIDE model and simplifies the overall security conversations.

<table>
<thead>
<tr>
<th>Threat category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Involves illegally accessing and then using another user's authentication information, such as username and password (Threats - Microsoft Threat Modeling Tool, 2017)</td>
</tr>
<tr>
<td>Tampering</td>
<td>Involves the malicious modification of data or information flow (Threats - Microsoft Threat Modeling Tool, 2017).</td>
</tr>
<tr>
<td>Repudiation</td>
<td>The process to refuse to the author of something that happened, tamper with logs to deny what has been done.</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Involves the exposure of information to individuals who are not supposed to have access to it. (Threats - Microsoft Threat Modeling Tool, 2017)</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Deny service to valid users, Potential Excessive Resource Consumption to overload servers or infrastructure.</td>
</tr>
<tr>
<td>Elevation of Privilege</td>
<td>An unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the entire system (Threats - Microsoft Threat Modeling Tool, 2017).</td>
</tr>
</tbody>
</table>

Table 9 STRIDE Description

3.2.5.3 What are we doing to defend against the threat? (Mitigate)

Once we have the layout and we know the issue, then it would be easy to defend our applications against those threats.
Few of examples that we can use to mitigate those threats:

<table>
<thead>
<tr>
<th>Threat</th>
<th>Security Property</th>
<th>Mitigation approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Authentication</td>
<td>• Password, certificates, multi-factor authentication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Digital signature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Token</td>
</tr>
<tr>
<td>Tampering</td>
<td>Integrity</td>
<td>• Digital Sign</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Hashing(SHA256)</td>
</tr>
<tr>
<td>Repudiation</td>
<td>Non-repudiation</td>
<td>• Secure Logging</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Auditing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Digital Sign</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Authentication</td>
</tr>
<tr>
<td>Information disclosure</td>
<td>Confidentiality</td>
<td>• Encryption</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Permission/access control list</td>
</tr>
<tr>
<td>Denial of service</td>
<td>Availability</td>
<td>• load balance, buffer, message queue</td>
</tr>
<tr>
<td>Elevation of privilege</td>
<td>Authorization</td>
<td>• Permission/ACL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Input validation</td>
</tr>
</tbody>
</table>

Table 10 STRIDE mitigation approach

3.2.5.4 Did we do a good Jobs (Validation)?

Finally, we can carry out a retrospective activity by repeating over the work we have done to check quality, feasibility, progress, and planning and the easy part is checking over our work:

- Look at the diagram. Does it represent the system well?
- Look at the list of threats. Did we find at least five threats per thing in the diagram, including data flows that connect systems?
- Did we file a bug per threat?

We will track a list of threats in bug tracking tool with mitigation as well as this will be integrated with unified security dashboard for transparency and visibility.

3.2.5.5 Benefits of using threat modeling

Benefits of using threat modeling as mentioned below by OWASP threat modeling (Threat Risk Modeling-OWASP)

- Build a secure design
- Efficient investment of resources; appropriately prioritize security, development, and other tasks
- Bring Security and Development together to collaborate on a shared understanding, informing the development of the system
- Identify threats and compliance requirements, and evaluate their risk
- Define and build the required controls.
- Balance risks, controls, and usability
- Identify where building control is unnecessary, based on acceptable risk
- Document threats and mitigation
- Ensure business requirements (or goals) are adequately protected in the face of a malicious actor, accidents, or other causes of impact
- Identification of security test cases/security test scenarios to test the security requirements
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

3.2.5.6 EoP(Elevation of Privilege) card game

Elevation of Privilege, a game designed to draw people who are not security practitioners into the craft of threat modeling. Delivery team will find this game very fascinating and fun way to draw developer or tester attention toward the security issue of the existing application controls vulnerable to threats or any missing security consideration.

![Elevation of Privilege Card](image)

*Figure 20 The '4 of Tampering' card from Elevation of Privilege*

3.2.6 Secure architecture review

OWASP ASVS (Application Security Verification Standard) checklist can be handy to evaluate the existing Web application security architecture.

OWASP ASVS assessment results can be presented as a diagram (fig.21), that can be published over Unified security dashboard(USD) and corresponding issues can be posted to the bug tracking tool.

The diagram will provide team members with an overview of which security areas need further improvements:

![OWASP ASVS Diagram](image)

*Figure 21 Result of OWASP ASVS assessment*
3.2.7 Security controls assessment for cloud computing

For self-assessment of cloud computing, we can use consensus assessments initiative questionnaire v3.0.1 (this is the latest version available at the time of writing a thesis). CSA (cloud security alliance) CAIQ (consensus assessments initiative questionnaire) consolidated most security standards (including ISO 27001, FedRAMP, NIST 800-53 R3, and PCI DSS) into a self-assessment questionnaire (Tony Hsu, 2018).

![Figure 22 consensus assessments initiative questionnaire](image)

3.3 Develop

In this stage, we will bring the ideas into existence based on blueprint provide in the previous two stages. We will automate most of the activities in this stage and later stages to speedup DevOps pipeline.

Main driver force at this stage would be the Development team. Therefore, it is critical from a security point of view to facilitate and provided all necessary resources, knowledge, and tools to them to develop secure software with minimum defects or bug.

During the development process, it is more cost effective and efficient to fix bugs in the earlier stages rather than later ones. The cost increases exponentially as the software moves forward in the SDLC (Arvinder Saini, 2017).

The Systems Sciences Institute at IBM (Dawson, et.al,2010) has reported that the cost to fix a bug found after development phase was around 15 times more costly than during design and up to 100 times more than one identified in the maintenance phase. Bugs introduced during the design phase, if not handled during the early stages of development, cost more as they can have a severe impact and are more complicated to resolve (Arvinder Saini, 2017). That is a clear indication that the financial impact of not addressing security during the development process can quickly become cost prohibitive.

To integrate security at this stage of DevOps pipeline and make continuous security we will separate this into three different processes.
3.3.1 Team Activities

According to Forrester Research (Forrester), “Effective developer education program can reduce vulnerabilities by \(~25\%)” (Forrester).

If a new application meets functionality requirements but causes a security breach in the process, the advantages to the business of that application may be outweighed by the costs. We cannot call our code or application great if it is not secure. That is why the responsibility for software security has shifted to developers at many organizations. On a practical level, this means that, as a developer, developers are shouldering more daily responsibility for testing and remediation of security vulnerabilities as code makes its way through the software development lifecycle (SDLC). For this reason, developers must be empowered to indeed own application security as a function of overall application quality.

To get a secure code out of developer's hands, it is indispensable to compile industries best coding practices and knowledge base.

3.3.1.1. Secure coding best practices

As cybersecurity risks steadily increase, application security has become an absolute necessity. That means secure coding practices must be part of every developer’s skill set. How we write code, and the steps we take to update and monitor it, have a significant impact on our applications, an organization, and our ability to do our job well (Veracode, 2018). Secure coding best practices must be incorporated into all life cycle stages of an application development process.

*Few of the Industry best practices that can be useful to start with,*

- OWASP Secure Coding Practices
- Secure coding best practices handbook guide resource
- Secure coding guidelines(Security in .NET)
- OWASP Developer Guide Reboot
- Android Application Secure Design/Secure Coding Guidebook
- SEI CERT Oracle Coding Standard for Java
- OWASP Code Review Guide v2
3.3.1.2 Security Knowledge Base

In DevOps, an organization should provide a platform to allow developers to think from the attacker’s perspective. Below is mentioned a few of the widely used knowledge base by various organizations for secure coding/practice knowledge.

- OWASP SAMM
- Microsoft Security Development Lifecycle (SDL)
- Open Reference Architecture for Security and Privacy
- Security Knowledge Framework
  - Train developers in writing secure code
  - Security support pre-development (Security by design, early feedback of possible security issues)
  - Security support post-development (Double check your code using the OWASP ASVS checklists)
  - Code examples for secure coding
  - View Demo (Username: admin Password: test-skf)
- AppSec Knowledge Base by Veracode
- OWASP Teammentor
- SANS Software Security Training
- CERT Secure Coding Training
- Elevation of Privilege (EoP) card games
- OWASP ASVS assessment
- OWASP Cornucopia
- OWASP SnakesAndLadders

An organization should train developers to develop defensible applications.

3.3.1.4 Open Source Firewall Prevention

Most of the software developers knowingly or unknowingly downloaded open source components from open source platforms like GitHub, Maven and other Open source code repositories to use during development without realizing or validating for vulnerabilities of those components. As per Nexus info graph (NexusFirewall infographics, 2019), One in 16 open source component downloads contains a known security vulnerability.

To proactively prevent developers from downloading known vulnerable code from open source code repositories into secure development cycle we have to implement an "Open Source firewall."

To secure DevOps, “Nexus Firewall” tool is a perfect fit to prevent any known vulnerabilities entering into our secure development cycle.

Reason for selecting “Nexus Firewall” for Continuous security toolchain,

- To create value early in the development process by automatically blocking defective components.
- Easily identify good components from bad components and block the bad ones from entering into Artifactory repository through customized policies.
- Enable a continuous audit, or take unwanted components out of distribution with quarantine.
- Prevent vulnerable Java, JavaScript, .Net, PyPi, RubyGems, and RPM components from entering our pipeline.
- Automatically stop risky components from entering into secure development chain.
3.3.2 Pre-commit checks

Pre-commit checks, it consists of steps to complete before the developer checks code into the source code repository. Various checks under these activities will enable development teams to run scans in their IDE while performing code and perform the assessment of threat modeling and architecture analysis.

3.3.2.1 IDE plugin (Secure Coding Analysis)

IDE (Integrated Development Environment) plugin tool. Provide inline security analysis ‘Just in Time’ in the dev environment as the developer writes code. Plug-in kind of tool which is more in-built for a developer to learn and correct security issues while bringing product ideas into reality. Rather than scanning for security issue after the code is written and committed for the continuous integration process.

- **DevSkim**
  DevSkim is a framework of IDE extensions and Language analysers that provide inline security analysis in the dev environment as the developer writes code. It is designed to work with multiple IDEs (VS, VS Code, Sublime Text), and has a flexible rule model that supports multiple programming languages. The idea is to give the developer notification as they are introducing a security vulnerability in order to fix the issue at the point of introduction and to help build awareness for the developer (Github - Microsoft/devskim).

- Checkmarx CxSAS
- Veracode Greenlight
- Synopsys SecureAssist
- FindBugs
- Fortify visual studio Plugin

For IDE plugin, we have chosen “**DevSkim**” due to easy to use and its intellisense characteristics for secure code check, and it is just like a spellchecking tool as shown in below screenshot.

![Figure 24 IDE plugin for secure code analysis](image-url)
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

3.3.2.2 Threat modeling analysis
An organization should encourage Developers to perform threat analysis before committing the changes to the source repository.

Threat modeling analysis will help developers to identify threats and vulnerabilities in the early stage, which guarantees that a secure application is being built, saving both revenue and time.

The team can use the automated tool to perform threat-modeling analysis, few of free to use as well as commercial threat modeling tool as mentioned below,

- OWASP Threat Dragon
- Microsoft Threat Modeling
- ThreatModeler

3.3.2.3 Code Review
We should keep this activity as optional, and the developer should choose this path only when they feel to get the code review or peer review done. However, it should be mandatory for all high-risk code not for all code.

We can use a few of the tools for automated code review purpose,

- Azure Repos
- Review Assistant
- Review Board
- Phabricator
- Collaborator
- Doxygen
- Crucible
- Gerrit

Once the code review is completed, the changes will be queued for commit-times checks.

To represent Code review activities in the logical workflow,

3.3.3 Commit-time checks
The next steps in the Implementation stage of Continuous Security are commit-time checks. The CI (Continuous Integration) workflow fully automatically triggers this activity as soon as the developer commits check-in to local branch repository. The changes will be committed to the repository only when

Figure 25 Code review- Logical workflow
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

the resulting code is evaluated to guarantee its Security, stability, integrity, and quality defined in the workflow. Any security issues found during these activities will be reported back to the developer by creating an email alert, and the corresponding commit will be rejected (not commits in the repository).

In day to day coding activities, we called it gated commit (Just before the Build stage).

**Gated Commit:** Gated commit is performed before committing to a central repository by merging changes made with the head of the master branch and carrying out tests to see if the changes do not yield a broken build.

These activities provide more rapidly feedback to the developer, and quick attention to any critical and high-security issues found with the committed code.

These activities include checks such as,

*3.3.3.1 Source Code Scanning for Malware detection*

We have included this into commit-time checks with CI pipeline to validate for any malware, threats, and vulnerabilities in source repositories. This step will detect and prevent commit of unknown files, executable or binaries before we proceed to next activities in our pipeline with the policy of “Any file could be infected”, ”Any file could be attempting to exploit a vulnerability to compromise a network” (MetaDefender - Advanced Threat Prevention Platform).

We have used MetaDefender API tool integrated with our CI/CD pipeline due to its Multiscanning features that include around 30+ anti-virus scan which will help us to prevent file-born threats. We can easily hook MetaDefender API with Azure DevOps CI/CD pipeline.

In case any Vulnerabilities reported by MetaDefender, the Gated commit build will reject the changes and status will be in the failed state.

At this step, a PowerShell script is written to hook MetaDefender API to scan files within the CI/CD pipeline, custom script is available under Appendix-A.

The result of Scan performed by MetaDefender using custom PowerShell

![Figure 26 Result of source scan restored from Metadefender API using a custom PowerShell script](image-url)
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

3.3.3.2 Secure code build
At this step, the CI pipeline will validate the compilation of source code to make sure that there is no compilation error. At the end of the compilation, it will generate deployable and testable artifacts also.

To secure the build pipeline, we should treat all warnings reported in this step as the error, and the CI pipeline should fail the build even in the state of warning.

3.3.3.3 Secure compiling
The secure compiling step will validate compiler/linker settings and other security-relevant binary characteristics needed to mitigate any initial code exploitation.

We can use below tools to validate this automatically with CI/CD pipeline,

**Binskim**: A correctness and security checker for Windows portable executables and Linux ELF binaries. BinSkim scans binaries to ensure they have been compiled securely,

These includes:

- **Use of Outdated Compiler Tool Sets** - Binaries should be compiled against the most recent compiler toolsets wherever possible to maximize the use of current compiler-level and OS-provided security mitigations (Binskim, 2018).
- **Insecure Compilation Settings** - Binaries should be compiled with the most secure settings possible to enable OS-provided security mitigations, maximize compiler errors and actionable warnings reporting, among other things (Binskim, 2018).
- **Signing issues** - Signed binaries should be signed with cryptographically strong algorithms (Binskim, 2018).

**CheckSec**: Designed to test what standard Linux OS and PaX security features are being used.

It integrates **Binskim** executable with Commit-time check to validate our source code output to help the developer to validate any compiler/linker issue before committing to the main branch.

BinSkim output shown below,

3.3.3.4 Security Unit testing
Security unit test along with regular unit tests should be performed on individual classes and methods to ensure that they adequately satisfy their API contracts with other classes as well as security requirement of written code functionalities.
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

Executing a security test at this step of the pipeline at very early in the development cycle allows faster feedback to the developer. Only the high category test cases written for the high-risk module should be executed at this stage for faster feedback to the developer just before the commit of code. We will execute all the unit test cases during a full fledge Build stage.

The commit will not be performed until unless the CI pipeline should no show green against executed test cases.

Example of security unit test cases category for the web application at this stage are:

- Input validation & Encoding
- Special character validation
- Error messages
- Authorizations
- Manipulation

The main objective of security unit tests is to validate that code is being developed in compliance with secure coding standards requirements.

3.3.3.5 Static application security testing (SAST)

Static application security test or static security analysis discovers security issues by scanning the code for instances of known coding flaws against a set of rules. It is one of the most valued activities in any mature SDL process, and this auto activity must have in Continuous security pipeline. Having this at an early stage of SDL exposed developers to potential security issues within their application code and which results in shorter feedback to the developer.

This automated activity is time-consuming, as it has to scan all the code line by line for security assessment. To shorten the scan time and the feedback loop to the developer, we will limit our static security analysis to check only on the set files that change or newly added which include,

- SQL Injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- XML eXternal Entity Injection (XXE)

Through analysis will be performed during the Next stage of continuous security pipeline that is Build stage.

Tools available to perform automated static security analysis,

- .NET Security Guard
- Security Code Scan
- Puma Scan
- SonarQube
- Fortify Static Code Analyser

We have selected Fortify Static Code Analyser as static security analysis tool because of its thoroughness and its ability to facilitate collaboration and easy to integrate with CI/CD pipeline.
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

3.3.4 Putting all automated steps together in Develop Stage

Figure 28 putting all automated steps together in Develop Stage

Source Code changes or added will get committed to source repository upon the success of all auto steps in this stage. However, any security risk reported at this stage will reject the commit, and no changes of code will be pushed to the repository.

At this stage of the workflow, the goal is to provide the development team with shorten and faster feedback on their work so that developers are notified whenever they commit changes that are potentially insecure.

On the success of all steps in this stage, it will auto trigger the next stage that is the “Build” stage in CI/CD pipeline of continuous security practice.

3.4 Build

The CI/CD pipeline will trigger the secure build stage after the successful commit of source code in the repository. At this stage, our pipeline will perform advanced and in-depth automated security validation and securing the artifacts, such as full security unit test, more deep level of SAST (Statics Application Security Test), Sensitive Information leakage Test, software composition analysis, Digital Signing and Hashing of artifacts, and scanning of the container.

Failure of any validation at this stage will,

- Auto trigger an alert to the application team member of critical and high-risk issues with details.
- Auto Create a work item in the bug tracking tool for feedback and priorities.
- Metrics and reports generated at this stage will be feed to security-unified dashboard and
- Any high-risk and critical security issue reported will stop the CI/CD pipeline from moving further.
This stage is critical from the security point of view as the status of this stage will decide whether generated artifacts are secure enough to move to the testing stage or not.

![Security checks during Build Stage in CI/CD pipeline](image)

**Figure 29 Security checks during Build Stage in CI/CD pipeline**

### 3.4.1 Secret Information Leakage Testing (SILT)

SILT is also one of the most critical steps to check with CI/CD pipeline to maintain continuous security practice within DevOps. During the development phase, the developer unintended hard code or leave the secret key information and potential secrets in the source code which provides backdoor entry to reason to exploit by the cybercriminal. In modern application development, this is the most common and severe problem faced by an organization.

To check this kind of secret leakage and help developers to detect such unintentional mistake, we will integrate a tool to search source code repository during CI/CD pipeline for hardcoded secrets like keys (e.g., AWS Access Key, Azure Share Key or SSH keys), passwords, API key. The main idea of integrating such a tool with continuous security practice is to detect any potential secret leaks before attackers can leverage them.

Compiled below, few of the tools to use as secret information leakages test,

- **TruffleHog**: Searches through git repositories for secrets, digging deep into commit history and branches. This is effective at finding accidentally committed secrets. This tool is more natural to shove into a DevOps pipeline.

- **DumpsterDiver**: Analyse significant volumes of various file types in search of hardcoded secrets keys or passwords. Additionally, it allows creating simple search rules with underlying conditions (e.g., reports only CSV file including at least ten email addresses). It can analyze any text file; git repositories as well as it can unpack and analyze the content of provincial archives (e.g., .zip or .tgz) for secret keys or password search. It is free to use tool.

- **Credential Scanner (aka CredScan) by Microsoft**: Tool developed and maintained by Microsoft to identify credential leaks such as those in source code and configuration files. Some of the commonly found types of credentials are default passwords, SQL connection strings and Certificates with private keys (Microsoft/credscan).

We have chosen DumpsterDiver(free to use the tool) and CredScan(easy to integrate with Azure DevOps Pipeline) as continuous security tools chain to search for secret keys leakage within Source repository as well as the target location.
3.4.2 Software Composition Analysis (SCA)

Modern applications are mostly “assembled,” not developed, and developers often download and use known vulnerable open-source components and frameworks (Nutanix, 2017).

Sonatype estimates 6% of downloads from Maven are of known vulnerable components and, in an earlier study, found that 71% of production applications contained at least one OSS component with known security flaws classified as "severe" or "critical" (F.Rashid, 2013).

CNET has reported that programmers are copying security flaws into the software. Developers do not write all of their code. They routinely borrow code from others, and they are not checking the code for security flaws. This widespread practice opens the door for hackers to have a broad impact with just a few exploits (Steve Morgan, 2015).

Known vulnerabilities in open source libraries or functions in source code are considered to be parts of the OWASP Top 10, and it should be identified at an early development stage.

The best way to stop entering these known vulnerabilities into DevOps pipeline would be to integrate a tool within our CI/CD pipeline to scan all open source libraries and function and make DevOps practice secure to follow continuous security model.

The more popular an open source libraries or functions is, the higher the value to hackers of exploiting a vulnerability found in it.

This is where SCA(Software composition Analysis) tools come into play. SCA(Software composition Analysis) toolchain within CI/CD pipeline scan and check for open source libraries or functions in the source code that have known vulnerabilities and licensing compliance like GPL (General Public License). It is must-have a tool under continuous security model.

The following tools will help us to scan for any known vulnerabilities found in open-source components:

- OWASP Dependency Check
- WhiteSource Tool
- SonaTypeNexus
- BlackDuck

The OWASP Dependency-check and White Source tool will be part of continuous security practice. Dependency check as it is free utility and widely used by open source community. Besides, the white source is a commercial tool, and it is seamlessly integrated fully into the build process, it supports all programming languages; CI tools, or development environments. It works automatically, continuously, and silently in the background, checking the security and licensing of open source components against WhiteSource’s regularly updated definitive database of open source repositories.
3.4.3 Security Unit testing

At the build stage, the security unit test should be configured to execute against all source code. Secure CI/CD pipeline should have a code coverage of 80-100%. Anything less than should be treated as a failure of the unit test.

Any security issue reported during unit test validation should stop the build moving further in steps.

At this stage, the security test suite will validate both positive and negative requirements for security controls such as:

- Identity, Authentication & Access Control
- Input Validation & Encoding
- Encryption
- User and Session Management
- Error and Exception Handling
- Auditing and Logging

It is always advisable to choose one free tool along with one commercial tool.
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

3.4.4 Static Application security testing (SAST)
Since we have already run SAST in the earlier commit-time checks, however during the build stage, we have to run more comprehensive rule against source code using SAST tool compare to commit-time checks (gated build). SAST tool also configured to validate for OWASP Top 10 most critical security risk for an application. Any risk reported by the SAST tool should break the build stage and alert the team. Using this tool during the build will help an organization to secure the foundation of the Application.

We will continue using the same tool (Fortify Static Code Analyser) as used during commit-time checks. Having Fortify Static Code Analyser as SAST tool integrated into enables remediation of vulnerabilities earlier in the secure development lifecycle.

Tool for continuous security: Fortify Static Code Analyser

3.4.5 Digital Signing
On successful completion of previous steps in this stage, we will digitally sign all output binaries, checksum (SHA256 or SHA512) for artifacts and the corresponding checksum will be stored securely in the database or encrypted file to validate authenticity and Integrity of deployed artifacts during monitoring stage.

Digital signing or code signing is the process of digitally signing executables and scripts to confirm the software author and guarantee that the code has not been altered or corrupted since it was signed. The process employs the use of a cryptographic hash to validate authenticity and integrity (Code Signing-Wikipedia).

The organization can use this feature to integrate code-signing tool with build stage to auto sign all the artifacts at the end of this stage. We can enable monitoring tool in production to monitor the integrity of the deployed package, and this will make sure that no one has tampered with the deployed/running artifacts in production or the software supply chain.

On Windows, we can enable Authenticode to sign the application’s code and artifacts. It also validates the artifacts has not been tampered with since it was signed and published.

For security tighten, we can enable policy across environments (Dev, Test, Staging, and Production) to allow running only trusted applications (only digital sign code).
A custom PowerShell script, which developed during this implementation to sign all the artifacts during build stage digitally. The script is available under Appendix-B for reference.

3.5 Test

Moving to the next stage of the Continuous security workflow, the CI/CD pipeline will automatically pick the latest ‘good’ build artifacts and deploy it to Test or staging environment to identify whether the security features of a software implementation are consistent with the design and to mimic users behavior before an application is released into production. This way we can make sure that services and product remain secure from the moment they hit the production environment. The environment (staging or Test) at this stage is the most representative of the production environment.

Software security testing fall under two different types,

1. Security functional testing
2. Security vulnerability testing

<table>
<thead>
<tr>
<th>Security functional testing</th>
<th>Security vulnerability testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>To ensure whether software security functions are implemented correctly and consistent with security requirements specification.</td>
<td>To discover security vulnerabilities (flaws in system design, implementation, operation, management) as an attacker.</td>
</tr>
</tbody>
</table>

Security functional validation has already been done in the previous stage (Develop & Build) of the pipeline using SAST tools. At this stage of the pipeline, the continuous workflow will be perform testing activities.
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

that will cover functional validation as well as vulnerability testing to complement SAST testing. Such as Dynamic Application Security Test (DAST), Interactive Application Security Test (IAST), SQL Injection checks and Fuzz Testing. Security testing also called penetration testing.

The ultimate goal for having security checks at this stage is to help development teams to remediate critical security issues as soon as they are introduced. Moreover, it will empower the QA team also to identify potential misuses and abuse of the application functionality.

All vulnerabilities identified during this stage will stop the step by moving further in the pipeline. The tool used in this stage will gather metrics and pass on to the Unified security dashboard (USD), and immediately create defects in the bug tracking system.

“Many organizations have significant network security in place, but it is not enough as 84% of all cyber-attacks are happening on the application layer,” said Tim Clark, Head of Brand Journalism at SAP (Steve Morgan, 2015). We have to look for methods, techniques, and tools that can easily be integrated into our CI/CD pipeline through API exposer of the command line.

Few of security testing methods and techniques to perform in an automated way while the code is in running state (pre-production stages) are analyzed and summarized as follow,

3.5.1 Dynamic Application Security Testing (DAST)

Dynamic Application Security Testing (DAST) takes place once applications have entered in test stage and it is black box security testing technique. DAST activities are a type of black hat or black box testing.

As per Gartner, Dynamic application security testing (DAST) detect conditions indicative of a security vulnerability in an application in its running state (Gartner IT Glossary).

DAST finds security vulnerabilities, threats, and weakness by employing fault injection technique and by feeding malicious data on an app by using application's exposed interfaces, requests, responses, scripting, data injection, sessions, authentication, and more. It helps to find common vulnerabilities such a SQL injection, cross-site scripting, OS command injection, and buffer overflow.

DAST can also cast a spotlight in runtime problems that cannot be identified by static analysis, for example, authentication and server configuration issues, as well as flaws visible only when a known user logs in (Sherif Koussa, 2018).

DAST can easily be integrated with UI testing tool like selenium to mimic real user action or manual testing actives to records and analysis real-time vulnerabilities, attack on an application.

3.5.1.1 to summaries DAST technique and analysis

| DAST is a black box security testing | • The tester does not know the technologies of frameworks that the application is built on.  
| • The application is tested from the outside in.  
| • This type of testing represents the hacker approach. |
| DAST Requires a running application | • It does not require source code or binaries.  
| • It analyses by executing the application. |
| DAST finds vulnerabilities towards the end of the SDLC | • There are high chances that existing vulnerabilities can be revealed after the development cycle is complete. |

Table 12 DAST technique and analysis
3.5.1.2 Taxonomy of Dynamic application security testing tools

<table>
<thead>
<tr>
<th>Commercial</th>
<th>Open Source or free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortify WebInspect</td>
<td>Arachni Scanner</td>
</tr>
<tr>
<td>InsightAppSec</td>
<td>Vega</td>
</tr>
<tr>
<td>WhiteHat Sentinel Dynamic</td>
<td></td>
</tr>
</tbody>
</table>

Table 13 Taxonomy of DAST Tools

As per the tooling criteria, OWSAP ZAP selected from open source tooling list and WebInspect from commercial tooling category to be best suited for DAST activities. Both tools can operate through functional automation tests like selenium or coded UI test to analyze vulnerabilities at runtime or operation time.

![ZAP tool driven by Selenium within CI/CD Pipeline for Dynamic analysis for an application](image)

3.5.2 Interactive Application Security Testing (IAST)

IAST (Interactive Application Security Testing) is a relatively new security tool category that analyses code dynamically at runtime during its execution within an application server for potential security problems. Thereby it is a combination of both SAST and DAST technologies in one.

It completes the detection of vulnerabilities that may not be found in SAST and DAST analysis and covers application security testing in the continuous security workflow.

IAST technique integrates monitoring agent within the application under validation on the server and collects all security vulnerabilities through monitoring agent on the fly. The collected result on runtime is then passed onto the management server for fast feedback to the developer.

IAST provide complete security coverage as it can easily be driven by any available UI automation tool to run business and technical smoke tests against the Web application and once the application begins to run, the monitoring starts. The IAST agent collects data and return immediate security posture results to Developers/QA and reports the metrics to a unified security dashboard.
According to Gartner, IAST gives good visibility into app code and execution. Additionally, the research firm predicts that by 2019, enterprise IAST adoption will exceed 30% (Arden Rubens, 2017).

As IAST agent is working inside the app, it can apply its analysis to the entire app -- all its code; its runtime control and data flow information; its configuration information; HTTP requests and responses; libraries, frameworks, and other components; and backend connection information. Access to all that information allows the IAST engine to cover more code, produce more accurate results and verify a broader range of security rules than either SAST or DAST (Sherif Koussa, 2018).

### 3.5.2.1 Advantage of using IAST in security testing phase to drive continuous security

Below summarizes table copied from checkmarx tool.

<table>
<thead>
<tr>
<th>Comprehensive Coverage</th>
<th>IAST resides inside the testing application/server and analyses the entire application source code, runtime control, database queries, and third-party libraries and frameworks (Arden Rubens, 2017a).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerabilities</td>
<td>IAST’s advantage comes from the fact that IAST is running and testing during the application’s runtime. This means that IAST can detect anything a DAST tool can; to begin with, furthermore IAST proceeds to cover many of DAST’s weak spots. For example, IAST can find vulnerabilities such as sensitive data stored in the log, which DAST cannot (Arden Rubens, 2017a).</td>
</tr>
<tr>
<td>Immediate Feedback</td>
<td>IAST provides instant feedback for the developer – within seconds of coding, he/she will be able to see the code’s security state allowing the addition of only “clean code” – this ultimately saves time, money and makes secure development as easy as pie. IAST will also have a better look into the application’s code and can provide the developer with more accurate remediation instructions (Arden Rubens, 2017a).</td>
</tr>
<tr>
<td>Zero Configuration</td>
<td>Meaning that if our application is running, IAST will be testing and analyzing, automatically and continuously, feeding back to developer/QA for further action. When DevOps and CI/CD teams build applications with an IAST agent inside from the very start, security is made substantially easy, as scanning for vulnerabilities within the app becomes continuous (Arden Rubens, 2017a).</td>
</tr>
</tbody>
</table>

*Table 14 Advantages of using IAST in security testing phase*
3.5.2.1 Taxonomy of interactive application security testing tools

<table>
<thead>
<tr>
<th>Commercial</th>
<th>Open Source or free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkmarx-CxIAST</td>
<td>Contrast Assess (IAST)-community addition</td>
</tr>
<tr>
<td>Contrast Assess (IAST)</td>
<td></td>
</tr>
<tr>
<td>Veracode</td>
<td></td>
</tr>
</tbody>
</table>

Table 15 Taxonomy of IAST tools

At the time of writing this thesis, we did not find many open source or free tool available for IAST. During the validation, Checkmark-CxIAST best fitted for CI/CD in enabling continuous security practice in an organization. Moreover, it provides deep insight into the application’s code flow, logic flow, data flow and configuration, and monitors the application as it is functionally being tested (automated or manual) and then reports on potential security flaws and vulnerabilities back to the management server, unified security dashboard including bug tracking tool.

3.5.3 Security Test best practices

While working in a DevOps team, we found some of the well-known industry best practices as summarized below that might help project teams to plan security testing to maintain the quality of security testing.

<table>
<thead>
<tr>
<th>Testing Guide (OWASP)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCI Penetration Testing Guidance</td>
<td>Instead of listing complicated testing cases and tools, the PCI penetration-testing guide includes four key agenda of the testing such as Penetration Testing Components, Qualifications of a Penetration Tester, Penetration Testing Methodologies, and Penetration Testing Reporting Guidelines (PCI DSS, 2015, V.1) (Tony Hsu, 2018).</td>
</tr>
<tr>
<td>NIST 800-115 Technical Guide</td>
<td>It provides practical recommendations for planning and conducting penetration-testing activities (Karen et al., 2008).</td>
</tr>
<tr>
<td>Container security</td>
<td>NIST Special Publication 800-190, Explained the potential security concerns associated with the use of containers and provides recommendations for addressing these concerns (Murugiah, et al., 2017).</td>
</tr>
</tbody>
</table>

Table 16 Security test best practices

During thesis writing, I also found kali Linux tools listing very useful in case an organization wants to create their in-house security testing tools chain with a mix of semi-automation or manual. Kali Linux tools listing provides a comprehensive set of security testing tools (Kali Linux tools).

3.6 Deploy

If all the previous stages have been completed successfully, and the application is ready for deployment (approved by QA team to deploy in the live environment), Deploy stage checks involving additional security validation to finish out the continuous security in DevOps.

The validation and checks at the Deploy stage can help find the vulnerabilities that may have slipped through test pre-production testing activities.

Automated Steps involve at this stage for security validation/checks are,

- **Securing Infrastructure configuration**
- **Provisioning the runtime environment**
3.6.1 Securing Infrastructure Configuration

For securing infrastructure configuration, we can use a configuration management tool like chef (Inspec and Hardening framework), Puppet, Ansible fully integrated with CI/CD pipeline to ensure that the configurations of applications, web services, databases, and the OS are Secure. We will define the security configuration (Inspec, Hardening framework) on top of configuration management tool like chef to achieve infrastructure security as a code.

An organization can also use a tool OpenScap to scan infrastructure configuration for security analysis before deploying the application to live environment. OpenScap provides a command line interface also which can easily integrate with CI/CD pipeline.

Secure configured infrastructure means infrastructure configuration and system hardening are compliant with industry security best practices as summarized below,

- CIS Benchmarks
- National Checklist Program (NCP)
- HIPAA
- ISO27001
- PCI DSS

<table>
<thead>
<tr>
<th>Commercial</th>
<th>Open source or free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chef-Inspec</td>
<td>OpenScap</td>
</tr>
<tr>
<td>Puppet</td>
<td>CFEngine</td>
</tr>
<tr>
<td>Ansible</td>
<td>DSC</td>
</tr>
<tr>
<td>SaltStack</td>
<td>Chef</td>
</tr>
</tbody>
</table>

3.6.1.1 Taxonomy of Securing Infrastructure Configuration tools

In this stage, we can also include a tool like Nmap to ensure that only expected ports are open in the production environment. Along with this, we can include Metasploit to ensure that we have adequately hardened our environments against known vulnerabilities, such as scanning with SQL injection attacks. The output of these tools should be put into our repository and compared with the previous version as part of our functional testing process (Gene et al., 2016, p325), or we can include the verification as an automated process within of CI/CD pipeline, for any drift, it will raise an alert.
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

Deploy stage ensure that delivery of application is consistent across the environment with same configuration and delivery model using CD pipeline.

After the secure configuration checks and ensuring the security of the environment, the application will automatically be deployed in the live environment and ready to serve customer request securely.

Now our application will enter into the last stage of our continuous security practice that is continuous monitoring stage.

3.7 Monitoring

Once the application is live, we have to prepare a security-monitoring mechanism to protect and prevent our system from being attacked and exploitation that may be taking place.

In a world of innovative and directed attacks, comprehensive prevention is not possible. The workloads and services must be continuously monitored for indication of unusual behavior that would be indicative of an actual breach (Neil macDonald and Ian Head, 2018)

The automated continuous monitoring solutions will allow the organization to review their security posture and systems function and security feedback loop to DevOps teams in real-time evidence.

For continuous security monitoring procedures, an organization needs to include below-mentioned activities to stay one step ahead of cyber threats,

- Logging,
- Analysis of logs from all available sources
- Security scanning for malicious programs,
- Threat intrusion detection, and prevention
- Monitoring of firewall and network security
- Vulnerability Scanning
- Verifying application code integrity and tampering at run time (Code sign verification)
- Web application firewall and
- Run time application security protection

Continuous security monitoring is essential today because organizations depend on technology and data to complete critical business processes and transactions.

Top of everything, adding threat intelligence will help the organization to visualize and correlate the security events across hosts and network segments (Tony Hsu, 2018).
3.7.1 Web Application Firewalls (WAFs)
Web application firewall (WAF) which monitors user input to an application for suspicious or problematic behavior. For instance, a WAF may examine incoming user input for the tell-tale signs of SQL injection (e.g., any of the variations on the classic pattern ' OR 1=1; --'). If it determines the input to be malicious, it will drop the request thus preventing an attack on the application.

3.7.2 Runtime Application Self-Protection (RASP)
Runtime application self-protection (RASP) technology identifies and blocks application security threats in real time. By adding detection and protection features to the application runtime environment, RASP enables applications to “self-protect” by reconfiguring automatically, without human intervention, in response to certain conditions (CA Veracode Application Security).

In real time, RASP analyses both the application’s behavior and the context of the behavior. Thus, it implements continuous security analysis, with the system responding immediately to any recognized attacks. This context-aware capability also enables RASP to be deployed with minimal up-front tuning or ongoing maintenance; RASP understands how data is used in the application to implement application protection automatically (CA Veracode Runtime).
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

3.7.2.1 The problem RASP addresses
Applications are a top attack vector for cybercriminals. Moreover, as organizations increasingly rely on web, mobile and cloud applications to drive their business, the threat surface has dramatically expanded. There is no one magic solution to protect against all application threats. Enterprises need a comprehensive solution to reduce risk in each phase of the application lifecycle. Just as enterprises deploy multiple layers of physical security (doors, locks, badges, surveillance cameras, etc.), they must apply multiple layers of cybersecurity to protect the applications that run their business throughout their lifecycle – from development to testing, to production (CA Veracode Runtime).

3.7.2.2 How RASP works
As per the CA Veracode explanation, The RASP agent, sitting in the runtime environment, sees application program flow in real-time and uses contextual insight to identify, validate and stop attacks in production applications (CA Veracode Runtime).

This detailed view into the actions of the system – including insight into application logic, configuration and data and event flows – improves accuracy and minimizes false positives. Besides, RASP can easily be applied to the web and non-web applications and does not affect the application design.

An example of a condition that could trigger a RASP response is the execution of instructions that access a database (which might cause a SQL injection exploit). The technology could either be in diagnostic mode and merely sound an alarm regarding an attack, or it could be in self-protection mode and stop a potentially malicious execution (CA Veracode Runtime).

RASP is the only technology that can block accurately and continuously, with minimal human involvement and zero latency (CA Veracode Runtime).

3.7.3 Threat intelligence
Threat intelligence in an organization is the process of acquiring knowledge about threats to an environment and system via multiple sources.

As per Gartner analysts Rob McMillan: “Threat intelligence is evidence-based knowledge, including context, mechanisms, indicators, implications and actionable advice, about an existing or emerging menace or hazard to assets that can be used to inform decisions regarding the subject's response to that menace or hazard” (Rob McMillan, 2013).
CHAPTER 3. CONTINUOUS SECURITY IN DEVOPS ENVIRONMENT

Having this process in place will help an organization to focus from ‘unknown unknowns’ to ‘known unknowns’ by discovering the existence of threats and then shifting ‘known unknowns’ to ‘known knowns,’ where the threat is well understood and mitigated (David Chismon and Martyn Ruks, 2015).

This activity is not a checkbox item.

![Figure 42 Threat Intelligence (Source: mwrinfosecurity)](image)

Threat intelligence process is itself a big topic to discuss and explain. Nevertheless, due to the limitation of this thesis we will leave this topic for future work.

Few of the available tools are summarized below,

### 3.7.4 Taxonomy of Security Monitoring Tools

<table>
<thead>
<tr>
<th>Category</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threat Intelligence</td>
<td>Log collector/sensor: FileBeat, LogStash</td>
</tr>
<tr>
<td></td>
<td>SIEM/visualization: Kibana, ElasticSearch</td>
</tr>
<tr>
<td></td>
<td>Threat intelligence platform: MISP - Open source threat intelligence platform</td>
</tr>
<tr>
<td></td>
<td>Threat intelligence feeds: External Source</td>
</tr>
<tr>
<td>Security Scanning</td>
<td>All-in-one security scanning: Security Onion</td>
</tr>
<tr>
<td></td>
<td>Secure configuration: OpenSCAP</td>
</tr>
<tr>
<td></td>
<td>Vulnerability: Nessus, OpenVAS</td>
</tr>
<tr>
<td>Security Application Protection</td>
<td>Web application firewall (WAF): ModSecurity, Akamai Kona Site Defender</td>
</tr>
<tr>
<td></td>
<td>Runtime Application Self-Protection (RASP): Fortify- Application Defender, CA Veracode Runtime</td>
</tr>
</tbody>
</table>

All the critical events and useful information metrics will be automatically recorded in Unified security dashboard and the bug tracking tool. It will help team members to prioritize their work items, and for an organization to take action based on their internal established Security incident response process.
4 A flow of work in a continuous security value stream

Below (fig.43) is a graphical representation of following of work across our purposed continuous security practice infused in CI/CD pipeline. Primary goal of below graphical representation is to help DevOps team members to evaluate purposed continuous security and provide everyone in the value stream with the fastest possible feedback about the security of what they are creating, enabling them to quickly detect and correct security problems as part of their work, which enables learning and prevents future errors.

The work in the continuous security value stream will move progressively across stages passing through each security checks and validation. As described in the previous chapter each stage will have their automated security validation steps and activities.

In case of any critical security finding in particular stage, it will reject the work moving further into right, and automated feedback loop will be provided to delivery team to work on that security issues finding, this is called as conditional follow of work. The feedback loop will be through a unified security dashboard, bug tracking tool and alert (email, SMS and any available team communication tool).

The stage Develop, Build, test, Deploy and Monitoring are part of automated CI/CD pipeline. However, Requirements and Plan that is team process will act as an enabler for the rest of the stages in the pipeline.

All the work in continuous security value stream has to full fill defined security criteria of an organization; it means while passing through each stage the work has to full fill security condition layout by organization otherwise it will not move to next stage. This is a continuous loop of iterations.

In below values stream, the monitoring stage has been kept close to the deploy stage, because, the monitoring of delivered work will start as soon as work reached to deploy stage. It will be better to call the monitoring stage as continuous monitoring.

![Figure 43 Flow of work in continuous security value stream](image-url)
5 Anatomy of Continuous Security workflow

Continuous security is applicable and relevant to all organizations that must increase the flow of planned work through the technology organization while maintaining quality, reliability, and security for customers. Presented seven stages in Chapter 3 is summaries as below to make this as working secure seven-stage model to achieve continuous security within DevOps culture.

In this workflow, the focus is centred on critical security activities needed for continuous security workflow. However, an organization can integrate a few other security activities to tool along with workflow to tighten the various stages along the DevOps Pipeline. This workflow also assumes that the delivery team has already automated other activities (e.g., unit tests, functional tests, user acceptance tests, integration test, etc.).
6 Taxonomy of continuous security toolchain to secure DevOps environment

To summarise, the toolchain and practices listed in Chapter 3 to achieve continuous security in DevOps environment integrated with CI/CD pipeline.

<table>
<thead>
<tr>
<th>Workflow Stage</th>
<th>Security Checks</th>
<th>Activities Type</th>
<th>Team Process</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUIREMENTS</td>
<td>Security Require</td>
<td>Out-of-band</td>
<td>OWASP Applicat</td>
<td>Free</td>
</tr>
<tr>
<td></td>
<td>Security requir</td>
<td></td>
<td>Application Security Verifi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ements for web</td>
<td></td>
<td>cation Standard</td>
<td></td>
</tr>
<tr>
<td></td>
<td>applications</td>
<td></td>
<td>(ASVS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security require</td>
<td></td>
<td>Common Vulsnebili</td>
<td></td>
</tr>
<tr>
<td></td>
<td>requirements</td>
<td></td>
<td>ty Scoring System (CVSS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>for Release gate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td>Security cont</td>
<td>Out-of-band</td>
<td>OWASP, security by des</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rols</td>
<td></td>
<td>ign principles</td>
<td></td>
</tr>
<tr>
<td></td>
<td>secure architecture and design</td>
<td>Out-of-band</td>
<td>the OECD Privacy Principles</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Secure architec</td>
<td></td>
<td>OWASP top 10 proactive c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ture review</td>
<td></td>
<td>ontrols</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Threat Modeling</td>
<td></td>
<td>OWASP ASVS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security controls assessment for cloud computing</td>
<td>Out-of-band</td>
<td>Microsoft Threat Modeling</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assessment of an organization security posture</td>
<td>Out-of-band</td>
<td>CAIQ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CI/CD Repository</td>
<td>In-line</td>
<td>OWASP SAMM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Microsoft Security D</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>evelopment Lifecycle</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(SDL)</td>
<td></td>
</tr>
<tr>
<td>DEVELOP</td>
<td>Secure coding best practices</td>
<td>Out-of-band</td>
<td>OWASP Secure Coding Practices</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security Knowledge Base</td>
<td>Out-of-band</td>
<td>OWASP Developer Guide Reboot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Open Source Fire</td>
<td>In-line</td>
<td>Jenkins</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wall Prevention</td>
<td></td>
<td>Github</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IDE plugin (Secure Coding Analy/9/9)</td>
<td>Out-of-band</td>
<td>Azure DevOps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Threat modeling analysis</td>
<td>In-line</td>
<td>OWASP Threat Dragon</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code Review</td>
<td></td>
<td>Fortify</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source Code Scanning for Malware detection</td>
<td>In-line</td>
<td>ThreatModeler</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Secure code build</td>
<td></td>
<td>Jenkins</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Secure compiling</td>
<td></td>
<td>Azure DevOps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security Unit testing</td>
<td>In-line</td>
<td>Jenkins</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Static application security testing (SAST)</td>
<td>In-line</td>
<td>Azure DevOps</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SonarQube</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fortify Static Code Analyser</td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 6. Taxonomy of Continuous Security Toolchain to Secure DevOps Environment

<table>
<thead>
<tr>
<th>BUILD</th>
<th>TEST</th>
<th>DEPLOY</th>
<th>MONITOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Unit testing</td>
<td>Dynamic Application Security Testing (DAST)</td>
<td>Securing Infrastructure Configuration</td>
<td>Web Application Firewalls (WAFs)</td>
</tr>
<tr>
<td>Static application security testing (SAST)</td>
<td>Interactive Application Security Testing (IAST)</td>
<td>Hardening Environment checks</td>
<td>Runtime Application Self-Protection (RASP)</td>
</tr>
<tr>
<td>Secret Information Leakage Testing (SILT)</td>
<td>In-line</td>
<td>In-line</td>
<td>Security Scanning</td>
</tr>
<tr>
<td>In-line</td>
<td>In-line</td>
<td>In-line</td>
<td>Vulnerability Scanner</td>
</tr>
<tr>
<td>Jenkins</td>
<td>OWSAP Zed Attack Proxy (ZAP)</td>
<td>OpenScap</td>
<td>Security Onion</td>
</tr>
<tr>
<td>SonarQube</td>
<td>Contrast Assess (IAST)-community addition</td>
<td>Chef-Inspec</td>
<td>OpenVAS</td>
</tr>
<tr>
<td>Fortify Static Code Analyser</td>
<td>Checkman-CoIAST</td>
<td>DSC</td>
<td>FileBeat &amp; LogStash</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Kibana</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MISP - Open source threat intelligence platform</td>
</tr>
</tbody>
</table>

**Figure 46 Taxonomy of Security toolchain in continuous security**
CHAPTER 7. DISCUSSION AND CONCLUSION

7 Discussion and Conclusion

This final chapter of this paper discusses the survey outcome and a conclusion in relation to the implementation of continuous security workflow in DevOps environment.

We employed quantitative methods (survey) to get a better perspective on the outcome of adopting Continuous security workflow in DevOps environment compared to using traditional security program. Our survey focused on projects that had implemented and practiced our continuous security workflow (as explained and shown in chapter 3) within CI/CD implementations to secure their DevOps environment. Continuous security assurance workflow was implemented in 5 different projects in an organization. We shared predefined closed-end questions online with all five projects team members to respondents. The questions and details are available under Appendix D.

7.1 Quantitative Research Method: Survey

Surveys are a commonly used research method worldwide. Survey work is visible not only because we see many examples of it in software engineering research, but also because we are often asked to participate in surveys in our private capacity. At first glance, it might look like an effortless task to create a survey, but there is more that needs to be done than just assembling the instrument (Barbara A. Kitchenham and S.L Pfleeger, 2001). We found that the survey would be the ideal method to use to generalize the finding.

For this paper, a quantitative approach has been employed to conduct our survey as quantitative methods are more structured, with closed-ended questions. The data obtained will be numeric, and is statistically analyzed. Examples of quantitative methods include experiments and surveys (John W. Creswell, 2013). We chose an online survey to compile our results so that respondents from across projects can answer it in a hassle-free manner.

The questionnaire was designed using Google forms, and the collected data were statistically analyzed. It was broadcast to all team members of projects where continuous security workflow had been implemented, and the link to the survey was shared with them. As the survey is prepared using Google forms, the responses are stored directly in a spreadsheet and as well as under responses result tab. The survey questionnaire was shared with 84 projects team members. Questionnaire and survey email are available under Appendix D.

A total of 76 responses received from five different projects, which include both Dev and Ops team members. To preserve the anonymity of the projects’ names, each was assigned an individual ID and referred to as project1, project2, and so on. The team members were promised confidentiality to get them to give genuine answers to the questionnaire forwarded to them.

During the survey, we defined the roles in two categories as Dev and Ops to simplify our data analysis.

Dev: Anyone involved in developing software products and service.
   e.g. (Developer, Tech Lead, Architects, Product manager, Quality assurance, Testers, Scrum Master, and Product Owner).
Ops: Anyone involved in delivering and managing software products and services.
e.g. (Information Security, System Engineers, System administrators, IT operational engineers, Release engineers, Database administrators, Network engineers, and Support professionals).

The roles of the respondents, the size of their projects, and total responses received are presented in the Table below.

<table>
<thead>
<tr>
<th>Projects</th>
<th>Project Team Size</th>
<th>Role</th>
<th>Total Responses (Project wise)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project-1</td>
<td>15-20</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>Project-2</td>
<td>15-20</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>Project-3</td>
<td>10-15</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Project-4</td>
<td>20-30</td>
<td>19</td>
<td>22</td>
</tr>
<tr>
<td>Project-5</td>
<td>15-20</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Team Size</td>
<td></td>
<td>68</td>
<td>76</td>
</tr>
</tbody>
</table>

Table 19: Distribution of respondents

![Figure 47 Distribution of responses (project wise)](image)

![Figure 48 Distribution of responses (Respondents role in project)](image)

![Figure 49 Distribution of responses (project head count)](image)
7.2 Analysis of the Responses: Discussion

We included the same questions (as mentioned in section 2.5) in our questionnaire survey also so that we can compare the latest result (Delivery integrated with continuous security workflow) with previously obtained survey result (using a traditional security program).

**Question:** What are the most significant application security testing challenges inherent in continuous integration/continuous delivery (CI/CD) workflows?

---

**Figure 50** Distribution of responses for Question 4

Summarised below is the results of the survey and comparison with previously obtained results:

<table>
<thead>
<tr>
<th>Observation 1</th>
<th>“Lack of automated, integrated security tools for CI/CD” come down from 61% to 9%.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason</td>
<td>Not anymore! The workflow outlined in this paper has covered all types of security testing tools, and technique at each stage of delivery phase from the security requirements stages onwards seamlessly integrated with CI/CD pipeline. A summary of security toolchain is presented in Chapter 6, which is a summary of all tools used for continuous security workflow to secure DevOps environment (Chapter 3).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Observation 2</th>
<th>Previously, 56% respondent “Inconsistent approach” as the next biggest challenge. However, in the latest survey, it comes down to ~4%.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason</td>
<td>The proposed continuous security workflow in this thesis can be used organically for any projects having the same proposed approach related to tooling and best practices.</td>
</tr>
</tbody>
</table>
Moreover, the designed continuous security workflow is integrated with unified security dashboard to maintain visibility and transparency, and it made it possible to provide a unified view for all stakeholders at one place.

A consistent approach is shown in Chapter 5 of this thesis (which is a summary of Chapter 3). The result is also evidence that the purposed continuous security workflow integrated across all five projects without any issues.

### Observation 3:

<table>
<thead>
<tr>
<th>Reason</th>
<th>~35% respondent that security checks slow down the release process. However, it is still lower than in the previous survey (48%).</th>
</tr>
</thead>
</table>

### Observation 4:

<table>
<thead>
<tr>
<th>Reason</th>
<th>The false positive percentage drastically down from 46% to latest ~9%. It means fewer false positive.</th>
</tr>
</thead>
</table>

### Observation 5:

<table>
<thead>
<tr>
<th>Reason</th>
<th>Another challenge reported in the previous survey (36%) was “Developer resistance.” However, with the latest survey, this is down to ~5%.</th>
</tr>
</thead>
</table>

It is an enormous improvement because the delivery team (Dev + QA+ Security+ Ops) are the principal users of continuous security and main driving force of the delivery pipeline. The continuous security workflow has been baked with all the security tools and technique so that developers do not have to leave their existing CI/CD pipeline. Additional, this thesis has described and suggested best available awareness, training framework, and coding and testing practices.
CHAPTER 7. DISCUSSION AND CONCLUSION

This allows project team members, not to resistance but to adopt the movement of continuous security in their standardized part of a daily operational procedure.

Moreover, all the security checks at each stage are fully automated and integrated with USD and bug tracking tool. This resulted in secure DevOps environment.

<table>
<thead>
<tr>
<th>Observation 6:</th>
<th>Secure code and improvement in application security with each release.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason</td>
<td>With this survey, we had added an option (secure code) to understand the main KPI of continuous security. ~72% of project team members’ respondent that the continuous security practices in their day-to-day activities has brought down the security issues and it has increased the security of the code, and overall more secure application.</td>
</tr>
</tbody>
</table>

The continuous security workflow helped the team to identify security issues early into the development cycle rather than at the experimentation stage (released to the customer).

Our survey also highlighted that by including continuous security activities in DevOps practice the organization achieved not only development and operations goals, but also security goals (high degree security assurance around the confidentiality, integrity, and availability of services and data).

Overall, we achieved the goal and aims of this thesis, “continuous security by designing a secure delivery workflow release gates infused with automated security checks at each gate early in the development cycle to keep the DevOps agility at pace and get the organization out of the checkbox trap and to secure DevOps delivery.”

7.3 Conclusion

The survey results in the above section shows that integrating continuous security into DevOps environment of CI/CD pipeline enables organizations to have secure work stream through every single stage of the project development cycle.

The Continuous security workflow as described and presented in chapter 3, shifts security from reactive to proactive and It brings two seemingly opposing goals “Speed of agility” and “secure code” into one streamlined process of CI/CD pipeline.

The thesis also provided us a valuable takeaway here; Automation is critical for continuous security practice. The workflow made us shift security at the left of the agile development process. Moreover, this supports the quote from Jen Andre; “Machines and tools are fantastic at handing a series of repetitive tasks, while humans are great at the deriving context from data. Why not offload this repetitive task of security validation to machines and allow humans to focus on innovation and improvement?” (Jen Andre, 2018).

This is what we did in the continuous security workflow; we infused the security tools at each stage of workflow within CI/CD to allow the security check to happen automatically with work stream.

This paper has also shown best practices for using security controls and toolchain of free and commercial to perform security validation of software delivery.

According to A. Martin and J. Eloff, A certain level of information security culture is already present in every organization using IT, but this culture could be a threat if it is not on an acceptable level (A.Martins
and J. Eloff, 2002, p12). However, by establishing continuous security practices and automation of security controls validation, an organization can foster a healthy and acceptable security culture (we show in evidence in our survey, observation 5).

The chapter 3 also presented that by automating manual security validation activities, we can generate evidence on demand to demonstrate that our controls are operating effectively, whether to auditors, assessors, or anyone else working in our value stream.

We found that continuous security workflow would improve not only overall security but also create processes that are easier to audit and that attest to the effectiveness of controls, in support of compliance with regulatory and contractual obligations. We do this by:

1. **Security as code.**
   According to Francois Raynaud (founder and managing director of DevSecCon): "Security as code is really about making security more transparent, to work with developers and speak the same language. We really need to give them the tools and the security policies so they know what to do and can do it themselves (Robert Lemos, 2017)."

   This is also evident in this paper (chapter 3); the continuous security workflow speaks the same language of developers so that they do not have to leave their existing automated delivery pipeline. We made the whole workflow transparent by integrating with CI/CD pipeline and with USD, bug-tracking tool.

2. **Securing DevOps by designing a secure workflow delivery model.**
   This was the goal and aims of this thesis, and it has been achieved by designing a secure delivery workflow release gates infused with automated security checks at each gate early in the development cycle to keep the DevOps agility at pace and to secure DevOps delivery. The single pane view is documented in chapter 5 of paper for reference.

3. **Embrace automation to integrate Security Tools and Practices within the CI/CD pipeline.**
   This paper of continuous security has used existing DevOps values of automation to infused security automation in CI/CD pipeline rather than added as a layer on top of the agile delivery process.

   Automation allows security controls to be fast, scalable and effective thus making it possible to keep a high pace for detecting errors, alerting about the errors, fixing the errors, finding countermeasures for future errors and forensics to identify why an error occurred (Amazon Web Services, 2016). The automation allows continuous security processes to be consistent and repeatable, with predictable outcomes for similar tests; it allows logging and documentation to be automatic (Anders Wallgren, 2015). The secure workflow derived in this paper has embraced automation to achieve continuous security in DevOps environment; chapter 3 has covered this in details.

4. **Security Tools and Practices integrated with CI/CD pipeline.**
   The workflow outlined in this paper has covered all types of security testing tools, and technique at each stage of delivery phase from the security requirements stages onwards seamlessly integrated with CI/CD pipeline. A summary of security toolchain is presented in Chapter 6, which is a summary of all tools used for continuous security workflow to secure DevOps environment (Chapter 3).
5. **Shifting security to the left of CI/CD assembly line.**

By involving security, experts from the start of the development process it is easier to plan and execute integration of security controls throughout the development process without causing delays or creating issues by implementing security controls after systems are running (Dave Shackleford, 2017). This paper covered this by involving security expert during requirement (section 3.1) and planning (section 3.2) stage as well as throughout the workflow to guide developer and work as a coach for the DevOps team. It is evident in section 3.1.3.

Having continuous security practice in place, an organization can spend more time adding value to products and less time and dollar in fixings costly vulnerabilities and unintentional mistake late in the agile delivery process (or in production).

In sum, Agile is a methodology, DevOps is a culture, and Continuous security practice is the process if infusing security end-to-end throughout that culture and methodology.

The benefits of continuous security bring to an organization that embraces it is copious such as:

- Break the silos and friction between DevOps and Security. Besides, it allows them to work as one towards the common business goals.
- It will allow developers to deliver high quality and secure code.
- Empower the DevOps team to take personal responsibility for security.
- Having security checks early in the DevOps practice to help early identify the vulnerabilities.
- Greater speed and agility for security teams.
- It eliminates silos, Promotes collaboration and teamwork between Development, Operations, and Security.
- Allows an agile team to treat security as a code.
- Provide visibility and transparency regarding security among all stakeholder of an application — short feedback loop to the development team regarding vulnerabilities and risk.
- Having automated security checks allows security team members to focus on the high-hanging fruit instead of struggling with false positive cases.
- Conveyed security awareness and practiced among Development team for secure coding.
- Continuous security workflow has security checks at every stage of the development process, from the requirements stages to development to onwards and this makes everyone responsible for security.
- Moreover, it will help the delivery team to increase the overall application security and prevent data breaches.
- Continuous security practice provided a security check in the form of seven-stage workflow so that security checks can be carried out in a continuous and iterative loop until the desired product is achieved.

The Continuous security practices in DevOps environment eliminate barriers between idea and Experimentation to "Go faster and Innovate and always stay secure".
8 Future Work

The use of artificial intelligence for detecting intrusion on the computer system is now widely considered as the only way to build an efficient and adaptive intrusion detection system. An organization can use an artificial neural network for efficient Intrusion detection.

Another area to use Artificial intelligence is to understanding monitoring information for an application to know when it is behaving maliciously. It can add significant value to the business outcome.

The purposed workflow in this thesis can also extend to include OPERATE and ADAPT stages. Moreover, the DevOps team can use Deep learning methods along with CI/CD workflow pipeline.

It would also be interesting to conduct surveys on more number of organizations with different sets of a project not limiting to the web-based project but to include a variety of technologies based project to possibly expand this study’s coverage on continuous security implementation using workflow designed in this paper.
Appendix A – PowerShell Script to hook Metadefender API within CI/CD Pipeline

function OPSWATScanFile
{
    param(
        [string]$site,
        [string]$FolderPathForScan
    )

    Try
    {
        if (Test-path $FolderPathForScan)
        {

            $FilesPathInFolder = Get-ChildItem -Recurse "$FolderPathForScan" | Where { !$_.PSIsContainer }
            foreach($FileInfo in $FilesPathInFolder)
            {

                Write-Host $("\" "* 50)
                write-host "Uploading File: $FileInfo"
                $FilePath = $FileInfo.FullName
                $FileName = $FileInfo.Name
                $uri = "$site/file"
                $ProgressPreference = 'SilentlyContinue'
                $response = Invoke-RestMethod -Uri $uri -Method Post -InFile $FilePath -UseDefaultCredentials
                $response = $response -Replace "@{data_id=","" -Replace ",""
                $resultURI = $uri/$response
                $resultResponse = Invoke-RestMethod -Uri $resultURI -Method Get -UseDefaultCredentials | ConvertTo-Json | Format-Json
                if ($resultResponse -clike "*Processing*")
                {
                    write-host "Processing..."
                    DO
                    {
                        Start-Sleep -s 3
                        $resultResponse = Invoke-RestMethod -Uri $resultURI -Method Get -UseDefaultCredentials | ConvertTo-Json | Format-Json
                    } while ($resultResponse -clike "*Processing*")

                }
            }

            if($FullDetails)
            {

        }
APPENDIX A

```powershell
write-host $resultResponse
}
Else {
    $resultResponse = $resultResponse | ConvertFrom-Json
    if (!($resultResponse.process_info.blocked_reason)) {
        write-host " File:" $FileName 
        write-host " Data ID:" $resultResponse.data_id 
        write-host "Profile:" $resultResponse.process_info.profile 
        write-host " Result:" $resultResponse.process_info.result 
        write-host "Process time:" $resultResponse.process_info.processing_time -ForegroundColor Green 
        write-host 
        continue 
    } Else {
        write-host " File:" $FileName 
        write-host " Data ID:" $resultResponse.data_id 
        write-host "Profile:" $resultResponse.process_info.profile 
        write-host " Result:" $resultResponse.process_info.result 
        write-host "Process time:" $resultResponse.process_info.processing_time 
        write-host "Blocked Reason:" $resultResponse.process_info.blocked_reason -ForegroundColor Red 
    }
}
Else {
    Write-Host "Provide Folder Path DoesNot Exist: $FolderPathForScan" -ForegroundColor Red 
}
catch {
    $ErrorMessage = $_.Exception.Message
    Write-Host "ERROR: $ErrorMessage" -ForegroundColor Red 
    Break 
}
Appendix B – Script to use signtool.exe for digital signing of Code

```powershell
#requires -version 3.0
[CmdletBinding()]
param()
Set-StrictMode -Version 3.0
Trace-VstsEnteringInvocation $MyInvocation

function assert($condition, [string] $message) {
    if (!$condition) { throw $message }
}

function assertDir([string] $path, [string] $message) {
    assert $path $message
    $message = $message -f $path
    assert (Test-Path $path -PathType Container) $message $path
}

function assertFile([string] $path, [string] $message) {
    assert $path $message
    $message = $message -f $path
    assert (Test-Path $path -PathType Leaf) $message $path
}

try {
    [string]$rootFolder = Get-VstsInput -Name rootFolder -Require
    [string[]]$filter = (Get-VstsInput -Name fileFilter -Default '') -split "\r?\n"
    [string]$subjectName = Get-VstsInput -Name subjectName -Require
    [string]$description = Get-VstsInput -Name description -Require
    [bool]$verbose = Get-VstsInput -Name verbose -AsBool

    Assert-VstsPath $rootFolder -PathType Container

    $signTool = [Environment]::GetEnvironmentVariable('SIGNTOOL')
    assertFile $signTool "Environment variable 'SIGNTOOL' not found"

    [string]$tfsUrl = [Environment]::GetEnvironmentVariable('SYSTEM_TEAMFOUNDATIONSERVERURI')
    assert $tfsUrl "Environment variable 'SYSTEM_TEAMFOUNDATIONSERVERURI' not found"
    [string]$tfsProject = [Environment]::GetEnvironmentVariable('SYSTEM_TEAMPROJECT')
    assert $tfsProject "Environment variable 'SYSTEM_TEAMPROJECT' not found"
    [string]$buildID = [Environment]::GetEnvironmentVariable('BUILD_BUILDID')
    assert $buildID "Environment variable 'BUILD_BUILDID' not found"
    [string]$sourceVersion = [Environment]::GetEnvironmentVariable('BUILD_SOURCEVERSION')
    assert $sourceVersion "Environment variable 'BUILD_SOURCEVERSION' not found"

    $files = @(Find-VstsMatch -DefaultRoot $rootFolder -Pattern $filter)
    $files = $files | Where-Object { -not (Test-Path -LiteralPath $_ -PathType Container) }

    assert $files "No files found under ${rootFolder} matching: ${filter}"

    # https://docs.microsoft.com/en-us/dotnet/framework/tools/signtool-exe
    $args = @(
        '/sm', ''"RootCA"'',
        '/r', ''$description @${sourceVersion}"'',
        '/d', ''"${tfsUrl}${tfsProject}/_build?buildId=${buildID}"'',
        '/fd', 'SHA256',
    )
```
if ($verbose) { $args += '/v' }

Set-Location $rootFolder
$files = @( $files | ForEach-Object { $f = (Resolve-Path $_ -Relative).TrimStart('./') $f = """${f}""
}$f )

for ($i = 0; $i -le 3; $i++) {
    # sign and apply digital timestamp
    $Global:LASTEXITCODE = 0
    Invoke-VstsTool -FileName $signTool -Arguments "sign $args $files" -WorkingDirectory $rootFolder
    if ($LASTEXITCODE -ne 0) {
        if ($i -lt 3) {
            Write-Host "Signing failed with exit code $LASTEXITCODE, retrying..."
        } else {
            throw "Signing failed with exit code $LASTEXITCODE after 3 retries"
        }
    } else {
        break
    }
}

finally {
    Trace-VstsLeavingInvocation $MyInvocation
}
### Appendix C – Owasp Application Security Verification Standard 3.0

Below Listed table is compiled from OWASP Application Security Verification Standard version 3.0 (OWASP ASVS, 2015).

<table>
<thead>
<tr>
<th>Security Requirement verifications</th>
<th>Control Objective</th>
</tr>
</thead>
</table>
| V1. Architecture, design and threat modeling | 1. [level 1] components of the application are identified and have a reason for being in the app  
2. [level 2] the architecture has been defined, and the code adheres to the architecture  
3. [level 3] the architecture and design is in place, in use, and effective |
| V2. Authentication | Authentication is the act of establishing or confirming, something (or someone) as authentic, that is, that claims made by or about the thing are true.  
Ensure that a verified application satisfies the following high-level requirements:  
- Verifies the digital identity of the sender of a communication.  
- Ensures that only those authorized can authenticate and credentials are transported securely. |
| V3. Session management | One of the core components of any web-based application is the mechanism by which it controls and maintains the state for a user interacting with it. It defined as the set of all controls governing state-full interaction between a user and the web-based application.  
Ensure that a verified application satisfies the following high-level session management requirements:  
- Sessions are unique to each and cannot be guessed or shared  
- Sessions are invalidated when no longer required and timed out during periods of inactivity. |
| V4. Access control | Authorization is the concept of allowing access to resources only to those permitted to use them.  
Ensure that a verified application satisfies the following high-level requirements:  
- Persons accessing resources holds valid credentials to do so.  
- Users are associated with a well-defined set of roles and privileges.  
- Role and permission metadata is protected from replay or tampering. |
<p>| V5. Malicious input handling | The most common web application security weakness is the failure to properly validate input coming from the client or the environment before using it. This weakness leads to almost all of the significant vulnerabilities in web applications, such as cross- |
| V6: Output encoding / escaping | |</p>
<table>
<thead>
<tr>
<th>(APPENDIX C)</th>
<th>site scripting, SQL injection, interpreter injection, locale/Unicode attacks, file system attacks, and buffer overflows.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ensure that a verified application satisfies the following high-level requirements:</td>
</tr>
<tr>
<td></td>
<td>• All input is validated to be correct and fit for the intended purpose.</td>
</tr>
<tr>
<td></td>
<td>• Data from an external entity or client should never be trusted and should be handled accordingly.</td>
</tr>
<tr>
<td>V7. Cryptography at rest</td>
<td>Ensure that a verified application satisfies the following high-level requirements:</td>
</tr>
<tr>
<td></td>
<td>• That all cryptographic modules fail in a secure manner and that errors are handled correctly.</td>
</tr>
<tr>
<td></td>
<td>• That a suitable random number generator is used when randomness is required.</td>
</tr>
<tr>
<td></td>
<td>• That access to keys is managed securely.</td>
</tr>
<tr>
<td>V8. Error handling and logging</td>
<td>The primary objective of error handling and logging is to provide a useful reaction by the user, administrators, and incident response teams. The objective is not to create massive amounts of logs, but high-quality logs, with more signal than discarded noise. High-quality logs will often contain sensitive data and must be protected as per local data privacy laws or directives.</td>
</tr>
<tr>
<td></td>
<td>This should include:</td>
</tr>
<tr>
<td></td>
<td>• Not collecting or logging sensitive information if not explicitly required.</td>
</tr>
<tr>
<td></td>
<td>• Ensuring all logged information is handled securely and protected as per its data classification.</td>
</tr>
<tr>
<td></td>
<td>• Ensuring that logs are not forever, but have an absolute lifetime that is as short as possible.</td>
</tr>
<tr>
<td></td>
<td>If logs contain private or sensitive data, the definition of which varies from country to country, the logs become some of the most sensitive information held by the application and thus very attractive to attackers in their own right.</td>
</tr>
<tr>
<td>V9. Data protection verification</td>
<td>There are three key elements to sound data protection: Confidentiality, Integrity, and Availability (CIA). This standard assumes that data protection is enforced on a trusted system, such as a server, which has been hardened and has sufficient protections. Applications have to assume that all user devices are compromised in some way. Where an application transmits or stores sensitive information on insecure devices, such as shared computers, phones, and tablets, the application is responsible for ensuring data</td>
</tr>
</tbody>
</table>
stored on these devices is encrypted and cannot be easily illicitly obtained, altered or disclosed.

Ensure that a verified application satisfies the following high-level data protection requirements:
- **Confidentiality**: Data should be protected from unauthorized observation or disclosure both in transit and when stored.
- **Integrity**: Data should be protected being maliciously created, altered or deleted by unauthorized attackers.
- **Availability**: Data should be available to authorized users as required.

<table>
<thead>
<tr>
<th>V10. Communications Security Verification</th>
<th>Ensure that a verified application satisfies the following high-level requirements:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• That TLS is used where sensitive data is transmitted</td>
</tr>
<tr>
<td></td>
<td>• That strong algorithms and ciphers are used at all times</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V11. HTTP security configuration</th>
<th>Ensure that a verified application satisfies the following high-level requirements:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• The application server is suitably hardened from a default configuration</td>
</tr>
<tr>
<td></td>
<td>• HTTP responses contain a safe character set in the content type header.</td>
</tr>
<tr>
<td></td>
<td>Adding Content-Disposition to API responses helps prevent many attacks based on</td>
</tr>
<tr>
<td></td>
<td>a misunderstanding on the MIME type between client and server, and the “filename”</td>
</tr>
<tr>
<td></td>
<td>option specifically helps prevent Reflected File Download attacks.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V12: Security configuration verification requirements</th>
<th>Ensure that a verified application satisfies the following high-level requirements:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Malicious activity is handled securely and adequately as not to affect the</td>
</tr>
<tr>
<td></td>
<td>rest of the application.</td>
</tr>
<tr>
<td></td>
<td>• Do not have time bombs or other time-based attacks built into them. Do not</td>
</tr>
<tr>
<td></td>
<td>“phone home” to malicious or unauthorized destinations.</td>
</tr>
<tr>
<td></td>
<td>• Applications do not have back doors, Easter eggs, salami attacks, or logic</td>
</tr>
<tr>
<td></td>
<td>flaws that can be controlled by an attacker.</td>
</tr>
</tbody>
</table>

Malicious code is extremely rare and is difficult to detect. Manual line by line code review can assist looking for logic bombs, but even the most experienced code reviewer will struggle to find malicious code even if they know it exists. This section is not possible to complete without access to source code, including as many third-party libraries as possible.

<table>
<thead>
<tr>
<th>V13. Malicious controls</th>
<th>Ensure that a verified application satisfies the following high-level requirements:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• The business logic flow is sequential and in order</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V14: Internal security verification requirements</th>
<th>Ensure that a verified application satisfies the following high-level requirements:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• The business logic flow is sequential and in order</td>
</tr>
<tr>
<td>V16. File and resources</td>
<td>Ensure that a verified application satisfies the following high-level requirements:</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>• Untrusted file data should be handled accordingly and in a secure manner</td>
</tr>
<tr>
<td></td>
<td>• Obtained from untrusted sources are stored outside the web root and limited permissions.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V17. Mobile</th>
<th>Mobile applications should:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Should have the same level of security controls within the mobile client as found in the server, by enforcing security controls in a trusted environment</td>
</tr>
<tr>
<td></td>
<td>• Sensitive information assets stored on the device should be done so in a secure manner</td>
</tr>
<tr>
<td></td>
<td>• All sensitive data transmitted from the device should be done so with transport layer security in mind.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V18. Web services</th>
<th>Ensure that a verified application that uses RESTful or SOAP-based web services has:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Adequate authentication, session management and authorization of all web services</td>
</tr>
<tr>
<td></td>
<td>• Input validation of all parameters that transit from a lower to the higher trust level</td>
</tr>
<tr>
<td></td>
<td>• Essential interoperability of SOAP web services layer to promote API use</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V19. Configuration</th>
<th>Ensure that a verified application has:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Up to date libraries and platform(s).</td>
</tr>
<tr>
<td></td>
<td>• A secure by the default configuration.</td>
</tr>
<tr>
<td></td>
<td>• Sufficient hardening that user-initiated changes to default configuration does not unnecessarily expose or create security weaknesses or flaws to underlying systems.</td>
</tr>
</tbody>
</table>
Appendix D – Survey Questionnaire

Dear Continuous Security Practitioners,

I am a Master’s student at Technische Universität Wien, TU Wien (Austria). As part of Master’s Thesis with title “Continuous security in DevOps Environment: Integrating automated Security checks at each stage of the continuous deployment pipeline.” I am surveying to collect the improvement and benefits provided by the adoption of continuous security workflow in your project(s).

This survey will not take more than 2-3 minutes. I would like to know the improvement brought by adopting continuous security infused in DevOps delivery line compare to previously used traditional security programs in your project.

All your responses are strictly confidential, and will only be accessed by myself, MOHAMMED, Jawed. Your anonymity will be preserved, and no details about a person or organization or project will be included in the thesis.

Thank you in advance.

Best Regards,
Mohammed Jawed, DevOps Leader
Adoption of Continuous security in DevOps Environment: Integrating automated Security checks at each stage of the continuous deployment pipeline.

This survey questionnaire is designed to collect the statistics of improvement and benefits provided by adopting continuous security in DevOps environment using secure workflow integrated within your project delivery pipeline compared to previously using traditional security programs.

**Figure 51 Survey Questionnaire**

https://docs.google.com/forms/d/e/1FAIpQLsdMGWLCA4AB5YQuMr1f2KHEO-EuMznW75d6bqMfEmSSt4KFmQ/viewform?usp=sf_link
Bibliographic


A. Martins and J. Elof (2006): Assessing Information Security Culture, Department of Computer Science, p2-3, Rand Afrikaans University, South Africa


A. Martina and J. Eloff (2002): Information Security Culture, Department of Computer Science, p2-3, Rand Afrikaans University, South Africa


Amazon Web Services (2016): Introduction to devsecops on AWS (2016), retrieved on 19th March 2019, slideshare.net/AmazonWebServices/introduction-to-devsecops-on-aws-68522874


79
BOB DIACHENKO (2019): 800+ Million Emails Leaked Online by Email Verification Service, retrieved on 10th Jan 2019, securitydiscovery.com/800-million-emails-leaked-online-by-email-verification-service


Christof Ebert, Gorka Gallardo, Josune Hernantes, Nicolas Serrano (2016): DevOps, IEE Software 33.3, DOI: 10.1109/MS.2016.68, Publisher:IEEE, p94-100


CVSS Calculator, V3.0: first.org/cvss/calculator/3.0


Forrester: forrester, retrieved on 3rd March 2019, go.forrester.com/


Github - Microsoft/devskim: *DevSkim is a set of IDE plugins and rules that provide security "linting" capabilities.*, retrieved on 4th Jan 2019, github.com/Microsoft/DevSkim.


Kali Linux tools:tools.kali.org/tools-listing


Margaret Rouse (2018): Rev up IT with continuous delivery and DevOps (March 2018), retrieved on 20 March 2019, searchitoperations.techtarget.com/definition/continuous-deployment


Microsoft/credscan: Microsoft/credscan: A Static Analysis Tool To Scan For ..., retrieved on 25 Feb 2019, secdvtools.azurewebsites.net/helpcredscan.html.


Neil macDONald, Ian Head (2018): DevSecOps: How to seamlessly integrate security into DevOps(30 Sep 2018), Gartner, ID: G00315283


BIBLIOGRAPHIC


Stephanie Herr (2018): CALMS and how it relates to Database DevOps (1 October 2018), retrieved on 25 Feb 2019, red-gate.com/blog/calms-relates-database-devops


