INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




DevSecOps for web applications: a case study

DIOGO HENRIQUE ARAÚJO GONÇALVES
Junho de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

DevSecOps for web applications: a case study

Diogo Henrique Araújo Gonçalves

Dissertation to obtain a master's degree in Informatics Engineering, Specialisation in Computer Systems

Supervisor: Isabel Azevedo

Porto, June 2022

Dedication

To my wife, Ana Jesus, the never-ending motivator of my life.

Resumo

O paradigma *DevOps* permite agilizar o processo de entrega de *software*. Visa reduzir as barreiras existentes entre as equipas responsáveis pelo desenvolvimento e as equipas de operação. Com recurso a estruturas de *pipelines* o processo de desenvolvimento de software é conduzido através de diversas etapas até à sua entrega. Estas estruturas permitem automatizar várias tarefas de forma a evitar erros humanos, liberta os intervenientes de tarefas morosas e repetitivas. Mais previsível e com maior exatidão o tempo necessário para as entregas de software é encurtado e mais frequente. Dadas estas vantagens o paradigma tem muita adoção por parte da indústria de desenvolvimento, no entanto, o aumento do volume das entregas acarreta desafios, nomeadamente no que diz respeito à segurança das soluções desenvolvidas.

Negligenciar os fatores de segurança pode levar a organização a acarretar com custos financeiros e denegrir a sua reputação. A integração entre o paradigma *DevOps* e segurança originou o paradigma designado por *DevSecOps*. Este visa a adoção pelo processo de desenvolvimento de ações de segurança, que após inseridas nas diversas fases de entrega, permitirão analisar e validar a solução, de forma a assegurar a sua consistência.

A arquitetura das aplicações web é por sua natureza acessível, o que resulta à sua maior exposição. Este projeto apresenta uma lista de problemas de segurança encontrados durante a pesquisa efetuada no domínio das aplicações web, analisa quais as ferramentas para a deteção e resolução destes problemas, quais as suas implicações no tempo de entrega de software e a sua eficiência na deteção de falhas.

Concluí com uma implementação de um fluxo de execução utilizando o paradigma DevSecOps, para compreender a sua contribuição no melhoramento da qualidade do software.

Palavras-chave: DevSecOps, DevOps, Serviços de rede, Segurança

Abstract

The DevOps paradigm streamlines the software delivery process, reducing the barriers between the teams involved in development and operations. It relies on pipelines to structure the development process until delivered. These structures enable the automation of many tasks, avoiding human error and freeing the team elements from doing slow and repeated tasks. More predictable and accurate development allows teams to reduce the time required for software deliveries and make them more frequent. Despite the wide adoption of the paradigm, the increase in deliveries cannot compromise the security aspects of the developed solutions. Companies may incur financial costs and tarnish their reputations by neglecting security factors.

Joining security and DevOps originate a new paradigm, DevSecOps. It aims to bring more quality compliance and avoid risk by adding security considerations to discover all potential security defects before delivery.

Web applications architecture, by their accessibility intent, has a vast exposed area. This project presents a list of common security issues found during the research performed in the web application security domain analyses, what tools are used to detect and solve these problems, which time implications they cause in the overall software delivery and their effectiveness in defect detection.

It concludes with implementing a pipeline using the DevSecOps paradigm to establish its viability in improving software quality.

Keywords: DevSecOps, DevOps, Web Services, Security

Acknowledgements

This document represents the fulfilment of a promise made to my thesis advisor Isabel Azevedo of ISEP. It was a privilege to work under her supervision a second time. I'm very grateful for the counselling and support that made this accomplishment possible.

Also, I would like to thank all my teachers at ISEP that went beyond their teachings and fed my passion for computer science.

Table of Contents

Table of Figures

Table of Tables

Acronyms

AHP	Analytic Function Deployment
API	Application Programming Interface
ASD	Agile Software Development
CD	Continuous Delivery
CI	Continuous Integration
CPE	*Common Platform Enumeration*
CR	Consistency Ratio
CVE	Common Vulnerabilities and Exposures
CWE	Common weakness Enumeration
DAST	Dynamic application security testing
DevOps	Development Operations
DevSecOps	Development Security Operations
IAST	Interactive application security testing
NCD	New Concept Development
QFD	Quality Function Deployment
SAST	Static application security testing
SCA	Software Composition Analysis
TAR	Technical Action Research

1 Introduction

The initial chapter gives an overview of the project presented here. It describes the context and objectives regarding the underlying problem. It ends with a summary of the remaining chapters of this document.

1.1 Context

Organisations adopting continuous integration and delivery (CI/CD) workflows bring many benefits (451 Research, 2018). So, development and operations individuals have joined DevOps (Development and Operations) teams. This paradigm allows the software to be delivered faster and with more iterations. However, software delivery in these dynamic, fast-paced environments is not driven by compliance and security risk avoidance. A paradigm DevSecOps wants to integrate security principles through the DevOps delivery model (Rakesh Kumar, Rinkaj Goyal, 2020). This project focuses on the DevSecOps paradigm to determine its potential.

1.2 Problem Statement

Technics and methodologies adopted by the DevOps paradigm shift aim to remove traditional boundaries between development and operation. A collaborative development approach reduces the time between “commit” and deployment into a production environment (451 Research, 2018). Fast-paced environments were focused almost entirely on delivery speed and iteration volume. With widespread adoption by larger enterprise organisations, the paradigm evolved into DevSecOps, where security is considered throughout the software delivery process.

The security component in DevSecOps aims to discover, catalogue, and fix security defects before any software release. However, the adoption of security measures throughout the development process is typically perceived to slow or stop software releases because of limitations of the security tools and lack of innovation in these technologies (Rajapakse et al., 2021 (3)).

This project addresses the challenges of integrating security into the DevOps paradigm by seeking proper approaches and tools to support security automation to increase testing coverage while efficiently reducing repeated manual testing.

1.3 Objectives

This project highlights the benefits of a DevSecOps centred software delivery implementation that prioritises quality and compliance requirements while avoiding security risks. The addition of security measures in an automated manner to a DevOps pipeline is a challenge. Still, after gathering an overview of the practices and experiences, the components from the automated portion of the DevSecOps solution will be submitted.

Therefore, the two primary objectives for this project:

(1) Assess what characterises a DevSecOps pipeline compared to a DevOps pipeline.
(2) Implement the automated portion from the DevSecOps pipeline for a specific case, retrieve knowledge about how effective they are at identifying vulnerabilities, and their impact on the overall software delivery.

1.4 Methodology

Because of its experimental nature, the technical action research (TAR) methodology was practised to resolve the objectives proposed in this work. The creation of an experimental artefact to solve a problem and figure is effects in practice before a real case scenario application (Wieringa, 2014).

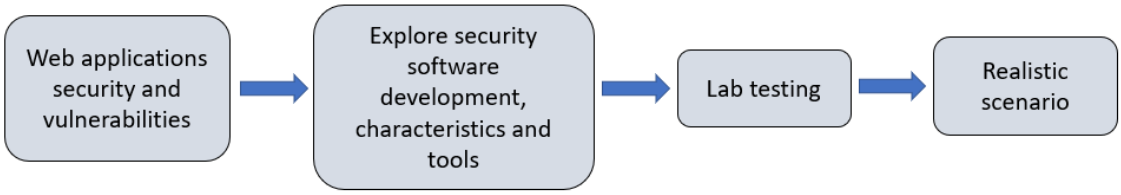


Figure 1 - Application of Technical Action Research

Figure 1 follows the TAR researcher and cycle structure. The design cycle began with an investigation of multivocal literature conducted to source the characteristics of DevOps (Section 2) and cyber security in the software development context (Section 3). Empirical Cycle conducted research for a solution (DevSecOps). One of the objectives of this project is to discern the differences between DevOps and DevSecOps. Analysis of the problems derived from transitioning from DevOps to DevSecOps, and which considerations should be taken to mitigate them concluded with an artefact (Section 3.3, Section 4). In the engineering cycle, a solution was developed to validate and improve the artefact (Section 5). The focus of this project was

not to create a software solution. Instead, existing projects were selected to evaluate the artefact implementation.

1.5 Document Structure

This chapter includes the context, the inherent problem, and the project's objectives.

The document has five more chapters, bibliographic references, and several attachments. The remained chapters are:

- **Background:** This chapter summarises the challenges encountered in the software development industry and what implications on the company culture and methodology were taken to solve them. The main subjects are waterfall vs agile software development, continuous integration, continuous deployment, continuous delivery, and DevOps.

- **State of the Art**: The subject of cyber security is presented, and its presence in software development is analysed. The DevSecOps paradigm is explored.

- **Design:** This chapter presents the proposed solution's requirements, tools, and implementation.

- **Results Analysis:** This chapter presents the obtained results from all the scans performed at the target project and how they've performed at OWASP top 10 vulnerabilities detection.

- **Conclusion:** Final chapter sums up the objectives set in Section 1.3, their conclusion, future improvements, and project contributions.

2 Background

This chapter serves as a subject introduction to the main scope of this dissertation. What changes in the software development process were introduced, and what challenges caused them. It explains the transition from waterfall to agile software development and how it evolved to a DevOps methodology.

2.1 Agile

Waterfall development is a methodology used to build new products. The name, waterfall, comes from how the process moves through one step at a time. This model makes the product through different phases (Figure 2). Teams are separated and intervene in the product development lifecycle at different times.

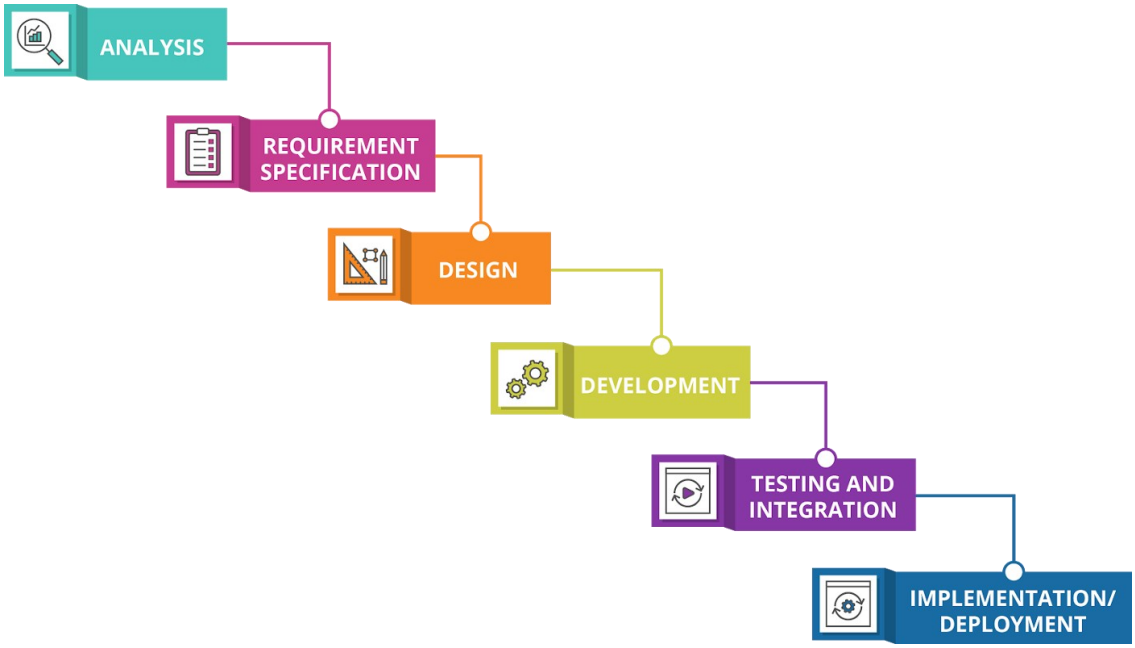


Figure 2 – Waterfall product development
(Justin Jones, Scott Waddell, 2019)

It assumes that requirements are all considered after the design phase and won't change during the development process. Therefore adding or changing a requirement after the development start can be lengthy and expensive (Justin Jones, Scott Waddell, 2019)(Steve blank, 2014).

In 2001 the Agile Software Development (ASD) method was introduced (Figure 3). It follows an incremental development approach. Sprints are partial increments aimed at the continuous improvement of the product features. Teams interact through the development process, taking shorter feedback cycles to better adapt to market and customer needs.

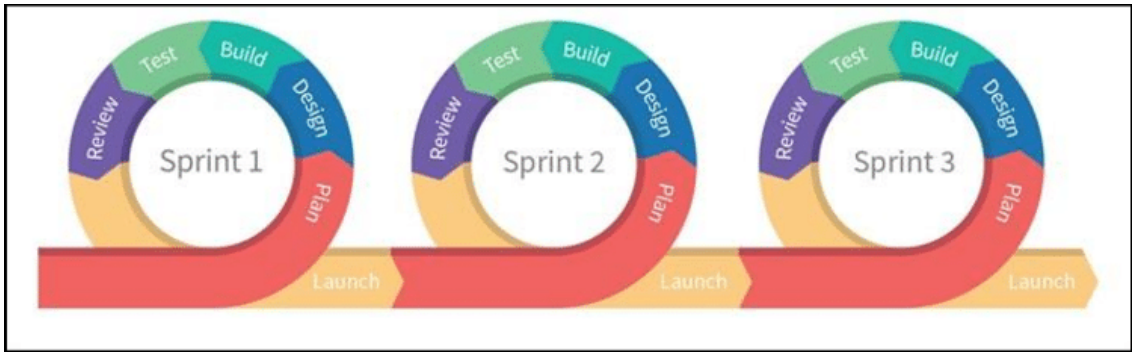


Figure 3 – The Agile Methodology
(Caluza, Las Johansen, 2020)

Rapid software adaptation imposes frequent and struggles less information exchange between development and operations teams. This is considered the handicap of the agile development method that gave rise to a new paradigm, DevOps.

2.2 Continuous Integration & Continuous Deployment – CI/CD

As ASD practices mature, enterprises seek to deliver more rapidly and reliably value to customers with minimal overhead by automating their integration and verification system.

Continuous integration (CI) offers a mechanism to conduct testing and rapid experimentation that drives innovation, frequent release cycles, and improved software quality and productivity.

The concept of continuous delivery (CDE) distances from traditional software delivery models by maximising how enhancements and bug fixes are delivered to the user or production environment. The frequency in which software is released increases from the ability to ensure that the product is ready. This is achieved by testing the solution, and if it is accepted, a manual deployment is set to deliver it to production. With faster and frequent user feedback, organisations can better manage the risk and cost of deployments (Jez Humble, Joanne Molesky 2011) (S.Mojtaba et al., 2017).

Continuous Deployment (CD) has most of the traits of CDE. It distinguishes itself by deploying every change into the actual production environment where customers and users use it. Because of this, CD requires more certainties and might not be suited to all organisations, while CDE can be applied to all types of systems and organisations (S.Mojtaba et al., 2017).

A report by 451 Research and Synopsys from 2018 revealed that the adoption of CI/CD in enterprises is growing, and this trend was expected to increase in the years after. Questioned how many servers or nodes the organisations had today and their estimate in the two years that followed (Figure 4).

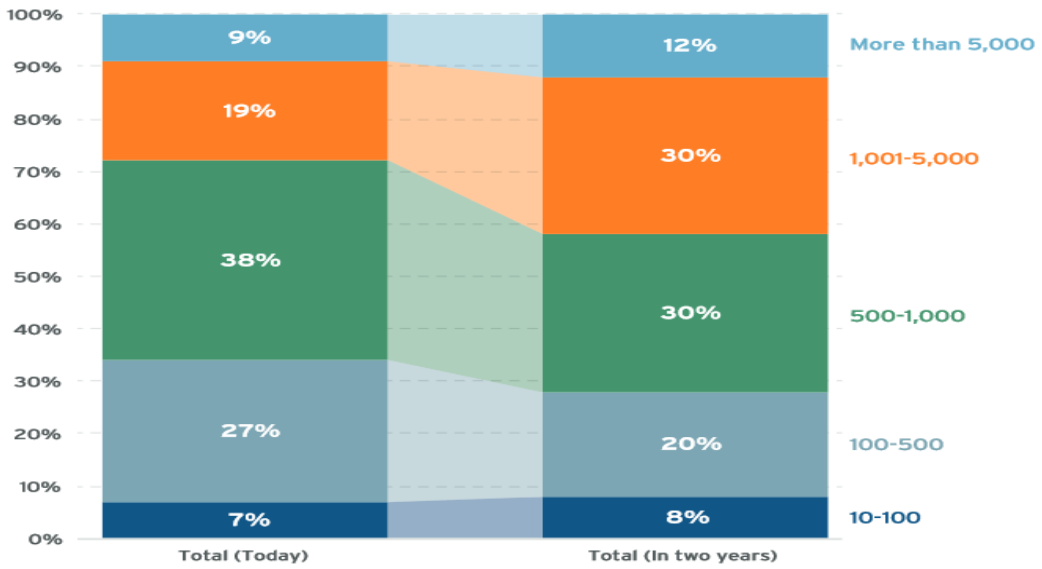


Figure 4 – Number of servers or nodes in CI/CD - today vs expected in two years
(451 Research, 2018)

The same report found that from their sample of organisations, 49% did new developments in a matter of days, 22% in weeks, and the same percentage within hours. A lower rate of 5% reported new deployments in minutes and 2% in seconds. Regarding the changes committed, 67% of the organisations indicated a significant code deployment by CI/CD workflows, 17% delivered substantial and complex changes, and 16% mentioned small and simple changes.

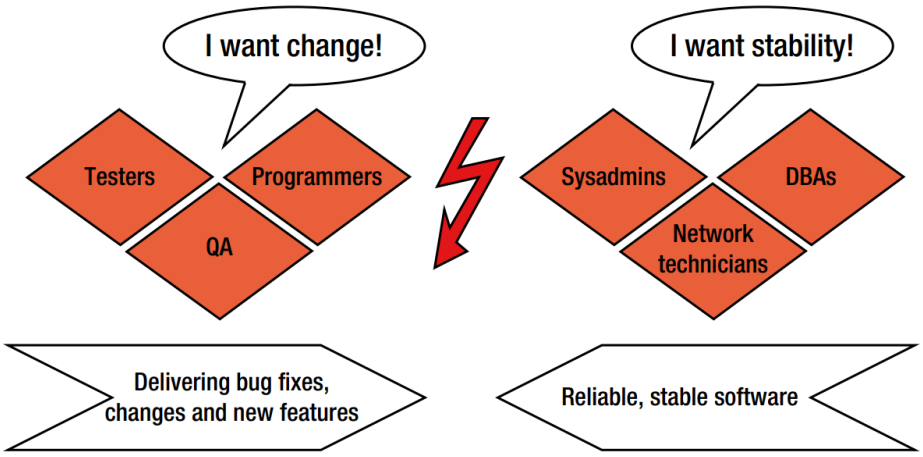


Figure 5 – Agile software development teams' conflict of interest
(Michael Hüttermann, 2012)

The increased volume brings more complexity to the CI/CD implementations. Contradictions between developing and operation teams might arise from this process and compromise the agility and quality of software delivered (Figure 5).

2.3 Development and Operations – DevOps

The agile software development delivery method combined different teams, programmers, testers, and quality assurance (QA) into a cross-functional development team. Derived from the success and adoption of agile software delivery methods, DevOps methodology combines development and operations teams in collaboration with each other to develop informed and sharp strategies for software integration and delivery workflows. Unlike agile software development, which drives the process from inception through construction but ends at transition, DevOps begins with the elaboration stage and follows the process until operations. This includes delivering the software to production and making it available to end-users (Figure 6).

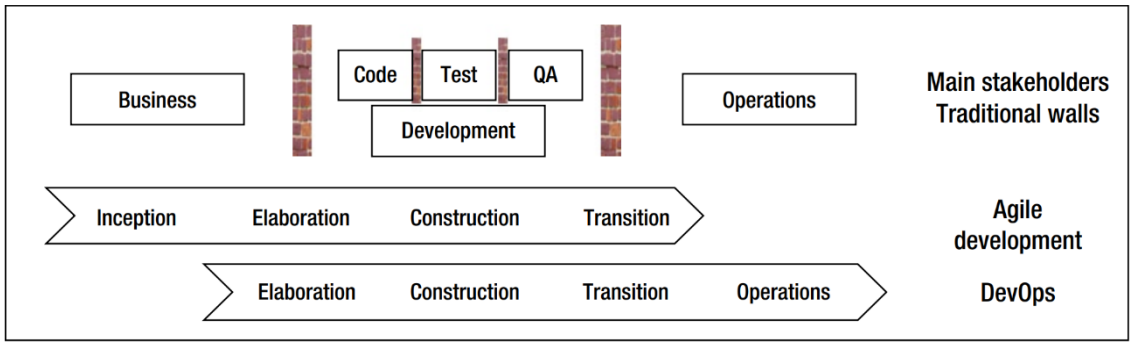


Figure 6 – Agile operates from inception to transition, DevOps begin with elaboration to operations
(Michael Hüttermann, 2012)

2.3.1 Challenges in adopting DevOps

Metrics and methodologies necessary for a DevOps methodology imply, in many circumstances, changes to the organisation's established culture. Despite proven success (N. Forsgren et al.,2017), scepticism and suspicion may arise from people's discomfort with the paradigm shift.

Efforts should not be spared in encouraging collaboration between teams, spreading knowledge related to tools and process adoption, reducing otherwise increased training costs, and hiring specialists.

2.3.2 DevOps model propositions

Companies must adopt a new culture and follow new metrics to adopt a DevOps approach. Collaborators no longer perform apart and serially - developer codes, followed by AQ tests, and ends with operations monitoring. How practitioners should successfully adopt DevOps motivates the build of models.

The DevOps proposition in (Lucy E. Lwakatare et al., 2017) is based on the following four principles:

1) Culture, the name DevOps, implies collaboration between developers and operators combining efforts to plan, design, and implement an enhanced application development. Tightened relationships increase the feedback loops, extending skills and responsibilities to find ways to improve constantly.

2) Automation is required to scale. Applying various tools and services strategically to reduce repeatable tasks and complexity allows the software development to remain compliant with Agile software development by establishing consistency and lowering the failure rates of new deliveries. Automation is implemented throughout the development, deployment, delivery, and support stages.

3) Measurement, finding indicators of faults by allowing more significant testing, implementing continuous deployment, and pursuing system stability. Also, the insights gained from measurements of deployment frequency, time to restore, and the failure rate can pinpoint which elements are beneficial to improve the overall solution.

4) Monitoring the volume of data from logs, system status, and infrastructure after deployments and obtaining relevant information. This information must be consolidated effectively and visually accessible so users can retrieve feedback quickly.

Also considered in (Håvard Myrbakken, Ricardo Colomo-Palacios, 2017), another principle for DevOps is sharing experiences and knowledge about the tools or techniques that play a vital role in DevOps. It promotes the relationship between the elements of the team and drives their interconnection for problem-solving.

Another proposition (Luz et al., 2019) is for a model based on the observations made in analysing successful applications of DevOps. It relates the categories required for a DevOps adoption (automation, transparency and sharing, continuous measurement, quality assurance) with the expected outcomes (agility, resilience, continuous measurement, quality assurance) (Figure 7).

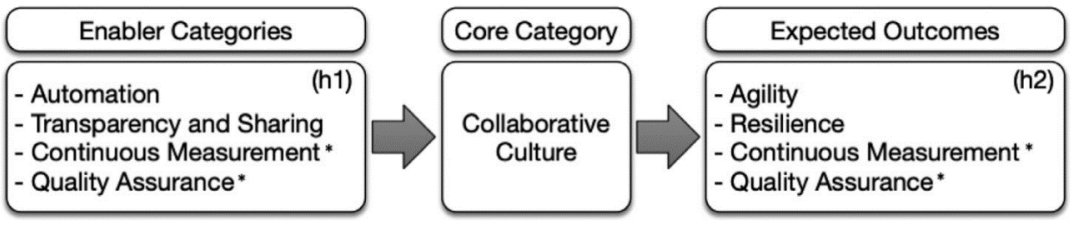


Figure 7 – Categories and Relationships
(Luz et al., 2019)

It is consensus that DevOps bridges the gap between development and IT operations. Deliveries happen in a faster/lean manner, rework is limited, and higher quality software is produced. The global objective of continuously improving the overall value towards the clients is maximised.

As part of continuous deployment (CD), DevOps relies on pipelines to automatically conduct and deliver software from the code submission to production deployment. Also, for the continuous delivery (CDE) pipeline, software changes are performed automatically until there are ready for deployment after manual authorisation (Figure 8).

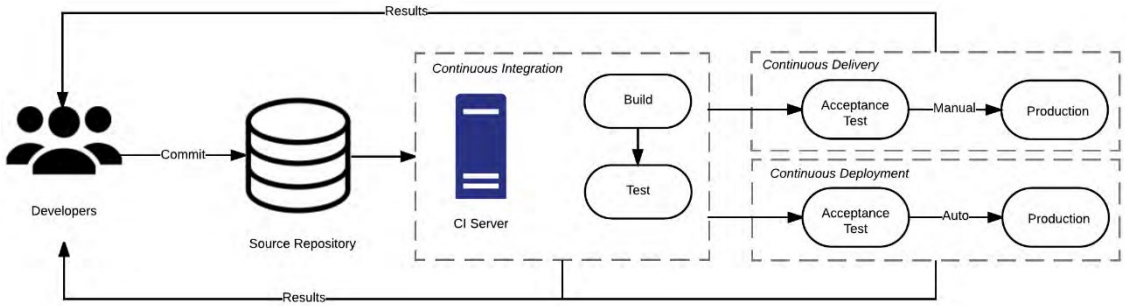


Figure 8 – Stages in a typical software deployment
(S. Mojtaba et al., 2017)

A typical software deployment pipeline structure (Figure 8) reflects four stages of development, design, code, test-quality assurance (QA), and build. Usually handled at a later stage of the software development come the security tests and activities. Considered non-functional requirements, security aspects are perceived to slow down and complicate the dynamic and fast-paced software delivery environments. The next chapter explores the negative impacts of oversight security on software delivery and how the realisation of this made, once again, the software delivery paradigm evolve.

3 State of the Art

This chapter explores the cyber security subject, its evolving, the impacts caused by disregarding it, and common security issues organisations practice. Regarding software development, it presents the primary defects causing security defects. The chapter explores the DevSecOps concept, why it was introduced, and its characteristics, challenges, and benefits.

3.1 Cyber security

A vulnerability can be exploited to compromise a system, either by software code flaw or system misconfiguration. Attackers can take advantage to gain unauthorised access to systems or networks. Perpetrators use vulnerabilities to control applications, damage files, or access sensitive information. Data breaches are attacks performed with the intent to retrieve an individual's record information. In 2021, this attack's top five average costs were higher in healthcare, financial, pharmaceutical, technology, and energy.

In 2020 global data breach cost on average $3.85 million, 2021 saw this value reach new heights with a worldwide average cost of $4.24 million and represents an increase of 11.9% since 2015 (IBM 2021).

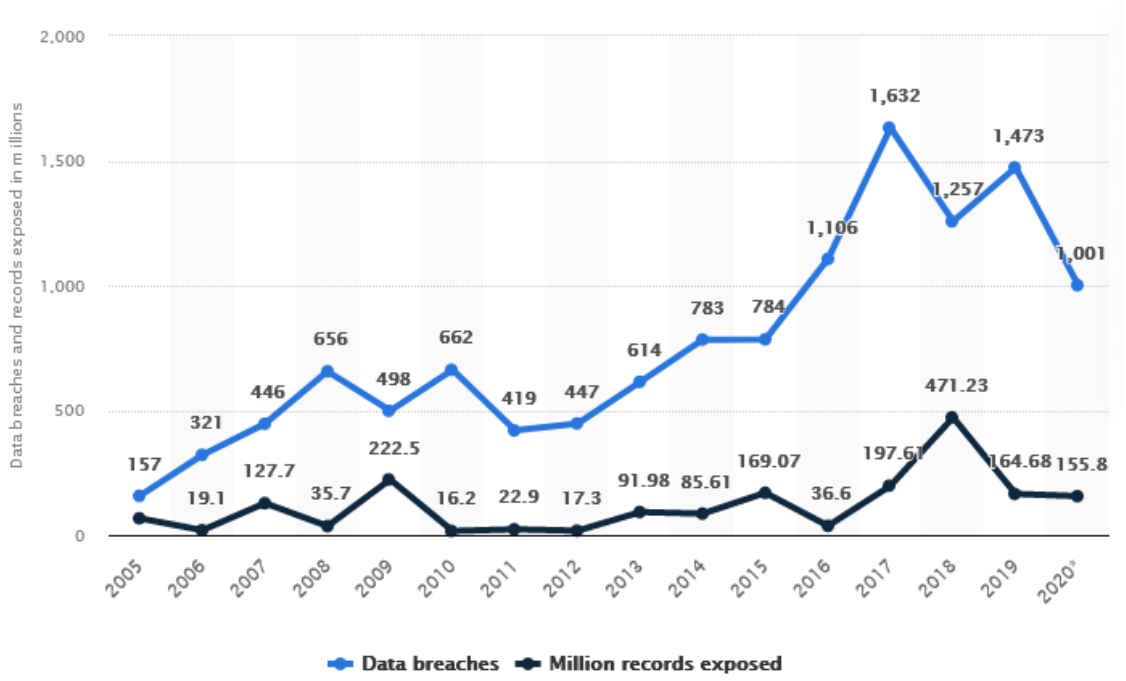


Figure 9 - 2005 to 2020 Data breaches and sensitive records exposed in the United States, excluding non-sensitive records exposed
(Statista 2022)

As shown in the (Identity Theft Resource Center, 2022) data breach report, in the United States, there were 1962 data breaches in 2021, a 68 per cent increase compared to the 2020 total of 1108 (Figure 9).

On average, in 2021, the number of days to identify a data breach was 212, and after being identified, it took on average 75 days to be contained (IBM 2021). At this stage, the organisations encompass monetary costs to solve the problem, and the public perception deteriorates. Eventually, additional costs on liability damages and litigation expenses have clients see their information exposed or used in a prejudicial way.

3.1.1 Common security issues

The open web application security project (OWASP) is led by a non-profit foundation called OWASP Foundation, the organisation started in 2001, and in 2004 the foundation was established in the US. To improve software security worldwide, in 2011, the organisation expanded to Europe with the name OWASP Europe VZW. First published in 2003, the top 10 web Application Security Risks list common web application security risks, such as injection, authentication, XML external entity (XXE) attacks, and misconfiguration. The OWASP also suggests related testing tools and prevention controls for each security issue (OWASP, 2022).

Common weakness enumeration (CWE) is a category system for software weaknesses and vulnerabilities. It is managed by the MITRE Corporation, a non-profit founded in 1958 by public and private partnerships to support various U.S. government agencies. Like OWASP, it releases an updated list every year. CWE Top 25 Software Errors lists the most common software coding errors, such as SQL injection, Cross-site request forgery (CSRF), and buffer overflow (CWE, 2022).

On both lists, broken access control is the top vulnerability, with 94% of applications containing this vulnerability. Access control ensures that users are restricted to their intended permissions. Failure to comply with this measure can lead to unauthorised information disclosure, modification, or even data destruction.

In 2021 Cognyte, a cybersecurity analytics firm left more than 5 billion records exposed online without an access control required. In the same year, the social network website LinkedIn had the personal data from 93% of its users on sale. This includes full name, phone numbers, physical and email addresses, geolocation records, and more. The list goes on with another social network website Facebook, where researcher Alon Gal discovered a leaked database with 553 million accounts (Maria Henriquez, 2021).

3.2 Fault detection in software development

The elements involved in a software development environment can include defects in the software. Has pointed out by (Jones Capers, 1996), after extensive state-of-industry surveys, defects are divided into five categories:

- Requirements defects
- Design defects
- Code defects
- Documentation defects
- Poor implementation of fixes or additional defects introduced accidentally while repairing prior defects

These may cause failures detected in compilation or tests. Still, other times, defects might not fail or are not considered by the tests, and only after deployed in a production environment the conditions to fail are met.

Most defects are introduced during the coding phase, either by mistake or misunderstood requirements. Despite this, more flaws are found further in the development process with testing being made and deeper integration.

As the defect gets more integrated into the system, the time and effort it takes to identify the problem increases, and the test case becomes more complex. A fault detected in production

might be difficult or impossible to replicate in a developer machine. Because software can be reusable, even a simple defect, if present in many places, can cause a big problem. If scalability isn't addressed in the code, the solution might not be enough for the expected load in a production environment.

Addressing defects in software development comes with a cost, and the later in the process, the more its value escalates.

Considering as an example, that an Application Programming Interface (API) is considered in the design phase for sharing user data, and only during the pentest (system test phase) is detected that it doesn't require authentication, the problem implies a rollback to the design phase for this requirement be added.

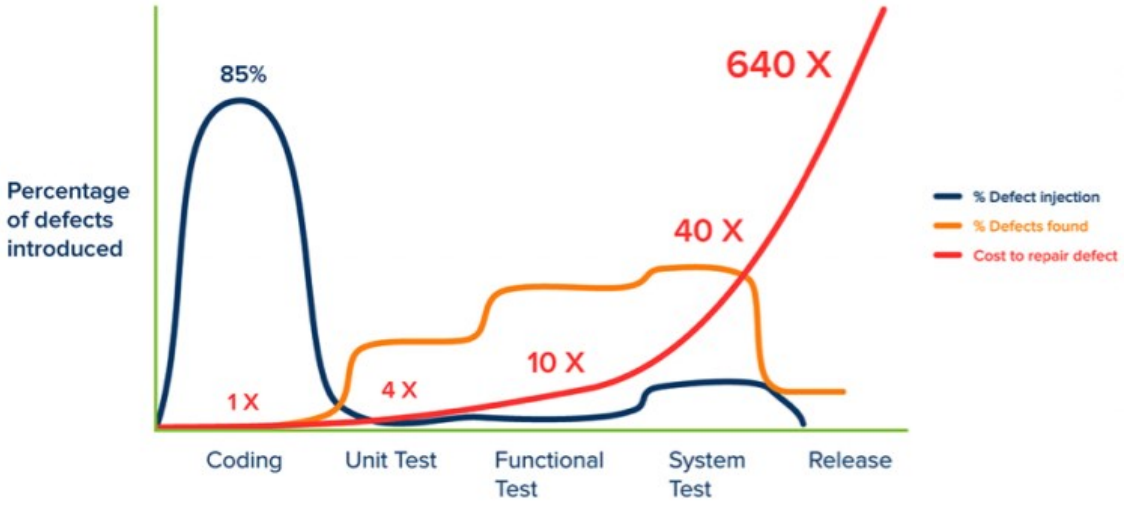


Figure 10 – Defect injection, defects found, cost to repair the defect
(Arthur Hicken,2018)

Figure 10 contrasts three essential parameters in the software development process:

- Percentage of defects introduced according to the phase of the project
- Percentage of defects discovered according to the phase of the project
- The cost of bugs according to the phase of the project

As previously mentioned, the first line shows that most defects are introduced in the code development phase (with a threshold of 85%), and another, but less drastic, rise in the system testing phase.

The second curve concerning the discovery of defects becomes more expressive in the testing phases, with the number of defects detected rising each passing stage. Remarkably the number of defects found during system tests is almost ten times the number detected during unit tests.

The third line represents the cost related to the defects and fixes. The increase is almost exponential thought out the development process. This highlights the need to address flaws as soon as possible.

3.3 DevSecOps

By acknowledging the importance of early testing as a fundamental part of the overall software development quality, continuous testing aims to build quality from the start, “shifting left” the product evaluation early and more often. As each or most increments are tested, defects are caught immediately and are addressed as soon as they occur. This makes their impact smaller, cheaper, and easier to fix.

Also shown before, when maliciously taken advantage of, security vulnerabilities in software cause damage to organisations and individuals. As the software development industry became aware of these problems, security shifted from being addressed after development to becoming a priority throughout the DevOps cycle.

In 2012, Neil MacDonald (Rakesh Kumar, Rinkaj Goyal, 2020) described the process of applying DevOps with shift-security-left, security-by-design, and continuous security testing, naming it DevOpsSec. Since then, the paradigm adopted the designation of DevSecOps (Developers, Security, Operations), placing the security (SEC) in the middle.

DevSecOps aims to secure coding and security tests to discover all potential security defects within the DevOps workflow before any release.

Since DevSecOps extends from the DevOps paradigm, it naturally carries the same challenges for its implementation. DevOps is a more mature paradigm with widespread adoption challenges thoroughly analysed and propositions to solve them.

In cultural mindset change, it becomes vital that communication and high trust must be established between all teams to address security as a primary concern. Knowledge and learning propagation are fundamental (Rodríguez et al., 2017)

DevOps implementation results in fast and agile software delivery. Maintaining the right balance between security compliances and quick delivery can be challenging to implement in these dynamic and fast-paced environments. Security scans can take a substantial amount of time to perform. Their analysis can length the process further because of false-positive results and manual or time-consuming interpretation of outputs. These factors promote a harmful perception of security integration, an obstruction too fast and frequent releases, causing organisations to become reluctant to a paradigm change.

Adding security tools into a DevOps structure is a challenging endeavour. Security tools lack the innovation for automated and integrated DevOps environments, and this inadequacy constitutes another technical issue for DevSecOps adoption (Rajavi Desai, T N Nisha, 2021)(451 Research, 2018)(Rajapakse et al., 2021) (Rajapakse et al., 2021(2)).

As challenging as it could be, adopting DevSecOps parading is very beneficial to the overall quality of the delivered software, risk avoidance, delay and rework reduction caused by late defects detection.

3.3.1 DevSecOps Implementation

DevSecOps practice is characterised by proactively engaging different teams or individuals to improve safe software development continuously. Accomplishing this implies that involved parties embrace building security and compliance into the software. They rely on tools and methodologies driven by the knowledge obtained and shared experiences.

This means that teams will be formed by individuals with different technical backgrounds working towards the same objectives. Developers, IT Operators, and Security reunite the skills required for a DevSecOps implementation.

The foremost vital components for the DevSecOps solution are:

- Culture
  A characteristic inherited from DevOps, the organisation must cultivate collaboration between individuals and communication to facilitate knowledge propagation and enhance competence. In conjunction, all the participants should define metrics to be accepted and acknowledged.
  Security practice alone is not enough. Customer relationship provides feedback and security strategies to ensure security maintenance.

- Automation to enable continuous testing
  Functional testing, at its core, may require ensuring that some functionality is available. By comparison, security testing has a broader area to cover, multiple testing scenarios, authentication, authorisation, injection and so on, can be repetitive and tedious. To resemble the rapid deployment of DevOps, DevSecOps must promote the automation of security controls. As the repeated manual testing is reduced, more security coverage can be implemented, and individuals can focus on tasks not suited for automation, like logic flaws and security requirements.

- Measurement, Monitoring and logging
  To achieve a state of continuous improvement, organisations must perform measurements on key performance indicators. Historically productivity has been measured by metrics like “lines of code written”, “time spent on tasks”, and “time

estimates" these may serve as capacity planning tools. Still, they commonly contribute to false insights about the performance of teams and individuals.

Instead, organisations must focus on quality and stability, therefore measuring the user acceptance of the new deployment and the system's stability before that deployment, so it can quickly react to that feedback. In (Nicole Forsgren et al., 2018), four key metrics are presented:

- Lead time measures how long it takes from client request to delivery. For further precision, the intermediate stages can be measured, for example, lead time from committed code to running in production.

- Deployment frequency, how periodic software is deployed to production.

- Time to restore service, in a fast-changing environment, failure can occur. Therefore, it's important to quantify how expeditious the process to restore that service is.

- Change fail rate is an essential metric for measuring the volume of changes that cause the system to fail.

Monitoring and logging are fundamental for corroborating the results obtained from the automation security controls.

The automation security goal is to identify all potential security defects before software delivery while reducing the amount of repeated manual testing. However, it doesn't mean that manual intervention can be replaced entirely to implement DevSecOps efficiently (Håvard Myrbakken, Ricardo Colomo-Palacios, 2017) (Rajapakse et al., 2021 (2)) (Pedra et al., 2021).

3.4 DevOps Vs DevSecOps pipeline

As mentioned earlier, a DevOps pipeline is conceived to automate and direct a continuous and expeditious software delivery. It combines practices such as CI, CD and CDE to achieve these goals without compromising the quality of the software delivered. The DevOps workflow consists of a CI/CD pipeline to carry the submitted code through the build, integrate, package, and release acceptance stages. The DevOps pipeline relies on tools and scripts for automating the development and integration of the various steps that constitute it (Gokarna et al., 2021). Tools are required for:

- Building the solution

- Unit tests
- Acceptance test
- Deploying the application

Adding security controls in DevOps is satisfied by DevSecOps. Extending from the DevOps collaboration promotion, DevSecOps involves security experts from the start of the workflow. A “shift left” approach to enhance the quality of the overall workflow carefully plans, designs, develops and maintains security through all the stages. Concerning the pipeline, DevSecOps adds tools to the existing DevOps tools repertoire to fulfil the following security controls (Rakesh Kumar, Rinkaj Goyal, 2020):

- Code Review & Security Guidelines Testing
- Software composition analysis
- Unit & Integration Security Testing
- Static Application Security Testing
- Dynamic Application Security Testing
- Fuzz Testing
- Interactive Application Security Testing
- Artefact repository Security management
- User Acceptance & Security Testing
- Security Smoke Testing
- Application and System Logging
- Continuous Monitoring and Alerting

3.4.1 DevSecOps Tools

The security controls must be implemented into one or more phases of the DevSecOps pipeline. Table 1 considers the pipeline phases, each phase’s target activities, and available open-source tools.

Table 1 – Open-source security tools suite

(GitHub, 2022(1)) (GitHub, 2022(2)) (SANS, 2022) (XebiaLabs, 2022)

Phase	**Target Activities**	**DevSecOps Open-source tools by licences category**		
		Apache	**GNU GPL**	**MIT**
Build	SCA (Software Composition Analysis)	Dependency Check, Synk	Bundler-Audit (GPL-3.0)	Blunder
Integrate	Unit & Integration Testing	TestNG		
	SAST (Static Application Security Testing)	Retire.js, Anchore, Docker, Bench, Clair	Find Security Bugs (GPL-2.0), SonarQube (GPL-2.0), SpotBugs (GPL-2.0)	DevSkim
	DAST (Vulnerability Scan, PenTest, Exploit Test)	OWASP Zed Attack Proxy	Wfuzz(GPL-2.0), Wapiti(GPL-2.0),Nmap(Modified GPL-2.p), SQLMap(Modified (Gpl2.0), Vuls (GPL-3.0), Sslyse (AGPL-3.0), BDD-Security (agPL-3.0)	
	IAST (Interactive Application Security Testing)	Hdiv		
Package	Artifact Repository	Archiva		
Accept	Acceptance, Smoke, Load & Performance Testing	Jmeter, Selenium, Robot		Cucumber
Deploy	Security Smoke Testing	Jmeter, Selenium, Robot		Cucumber
Operate	Logging, Analysis, Visualization & Notification	OpenSearch, Elasticsearch, Logstash, Kibana, ElastAlert, Grafana, Graphite, Seyren	Graylog2 (GPL-3.0), Rsyslog (GPL-3.0)	
	Continuous Monitoring	Sysdig Falco, Prometheus	Nagios Core (GPL-2.0), Zabbix (GPL-2.0), Munin (GPL-2.0), Cacti (GPL-2.0), Icinga (GPL-2.0)	StastDd
	Quality & Performance Measurements, Analytics, Trending & Alerting	OpenSearch, Elasticsearch, Logstash, Kibana, ElastAlert, Grafana, Graphite, Seyren, Prometheus, InspectIT	Graylog2 (GPL-3.0), Zabbix (GPL-2.0), Cacti (GPL-2.0), Icinga (GPL-2.0), LimeSurvey (GPL-2.0)	

3.4.2 Marker projects

To test and evaluate the DevSecOps pipeline's effectiveness, previously existing projects will be analysed, and one will be selected for this project. These projects have previously known defects, including training material and testing tools.

OWASP NodeGoat

Developed by the OWASP foundation, the OWASP NodeGoat is a Node.js web application made vulnerable by containing the most common web application defects. It's part of a significant project Broken Web Applications (BWA), also from OWASP, that includes a more comprehensive collection of vulnerable applications but can be obtained and executed singularly. The latest update at the time of this writing was released on 5 September 2021 (OWASP NodeGoat, 2021).

Hackzon

It was a project that started development by Rapid7 to demonstrate the effectiveness of their product, the Metasploit web vulnerability scanner. Nowadays, Hackzon is an open-source project that contains a web application that simulates an online store. It uses AJAX and web services and has several defects for testing purposes. The latest update at this writing was released on 11 May 2021 (Hackzon, 2021).

Goof

It's an open-source project developed by Snyk Limited with some common vulnerabilities to demonstrate Snyk software's abilities to detect and fix them. It is based on another tutorial project from Dreamers Lab, built with JavaScript, Node.js and MongoDB. The latest update at this writing was released on 22 August 2021. (Goof, 2021)

Web Security Dojo

Developed by Maven Security Consulting, a company that specialises in information and IT security, Web Security Dojo is a virtual machine that offers vulnerable web applications, training material and testing tools. The latest update at this writing was released on 4 May 2020 (Web Security Dojo, 2020).

Additional projects are available but weren't considered since the source code of the web applications isn't accessible, and other than the website, no further insights are attainable.

3.4.3 Summary

This chapter provided an insight into security in the software development environment. It explored the DevOps methodology and the goals set by DevSecOps to solve the security concerns. DevSecOps builds on top of the DevOps methodology as a “shift left” approach to the security matter. In essence, DevSecOps emphasises the importance of security education for all members involved in software development. Regarding the DevOps CI/CD pipeline, it proposes integrating security controls through all its stages to improve the overall quality and compliance of software.

4 Design

This chapter presents the environment setup for this project's planning and implementation phase. The selected target project and the security testing tools chosen to integrate and build the security pipeline are analysed. The solution focused on a portion of the security pipeline relying solely on automated tools. The deep code analysis tools attempt to identify defects in the software composition. Web scanning tools, include penetration and exploit testing on a running application to explore how it behaves.

Recalling the security controls from the Section 3.4 and after analysing the available tools, the solution can cover or partially cover all but the last two security controls:

- Code Review & Security Guidelines Testing
- Software composition analysis
- Unit & Integration Security Testing
- Static Application Security Testing
- Dynamic Application Security Testing
- Fuzz Testing
- Interactive Application Security Testing
- Artefact repository Security management
- User Acceptance & Security Testing
- Security Smoke Testing
- Application and System Logging
- Continuous Monitoring and Alerting

Logging and monitoring are significant in a production environment or in a running solution, either way, both depend on knowledgeable user interpretation.

4.1 Requirements

The selected target project should be open-source, as major security controls involve source code analysis. The tools must be integrated with a CI/CD pipeline tool to drive the process through all the security scans. Like the targeted project, security tools must also be open source these were previously identified in Section 3.4.1. Tools should individually or in conjunction with other tools, enable deep code scanning and analysis to identify known vulnerabilities, and outdated libraries. It also must perform web scanning to identify vulnerabilities and determine if they can be exploited.

4.2 Target Project

A project with previously known defects had to be found to evaluate the effectiveness of the multiple security tools. Source code access was necessary because of the “shift left” approach from the DevSecOps paradigm that begins at code development.

The chosen project was the OWASP NodeGoat. As mentioned in the previous chapter, NodeGoat is developed in Node.js by the open web application security project (OWASP). This open-source application is purposely designed to include the OWASP top 10 vulnerabilities. OWASP shares comprehensive documentation on how those vulnerabilities can be exploited on the application.

The type of vulnerabilities included in the project extends from the injection, broken authentication, Cross-Site Scripting (XSS), Insecure Direct Object References (DOR), Misconfiguration, Sensitive Data, Access Controls, Cross-Site Request Forgery (CSRF), Insecure Components, Redirects, Regular Expressions DoS (ReDoS) and Server-Side Request Forgery (SSRF).

The NodeGoat application (Figure 11) simulates a retirement saving management service. It’s a relatively simple application that includes user authentication and profile. There’s also an administrator role and private access. It uses the NoSQL database MongoDB for its data storage.

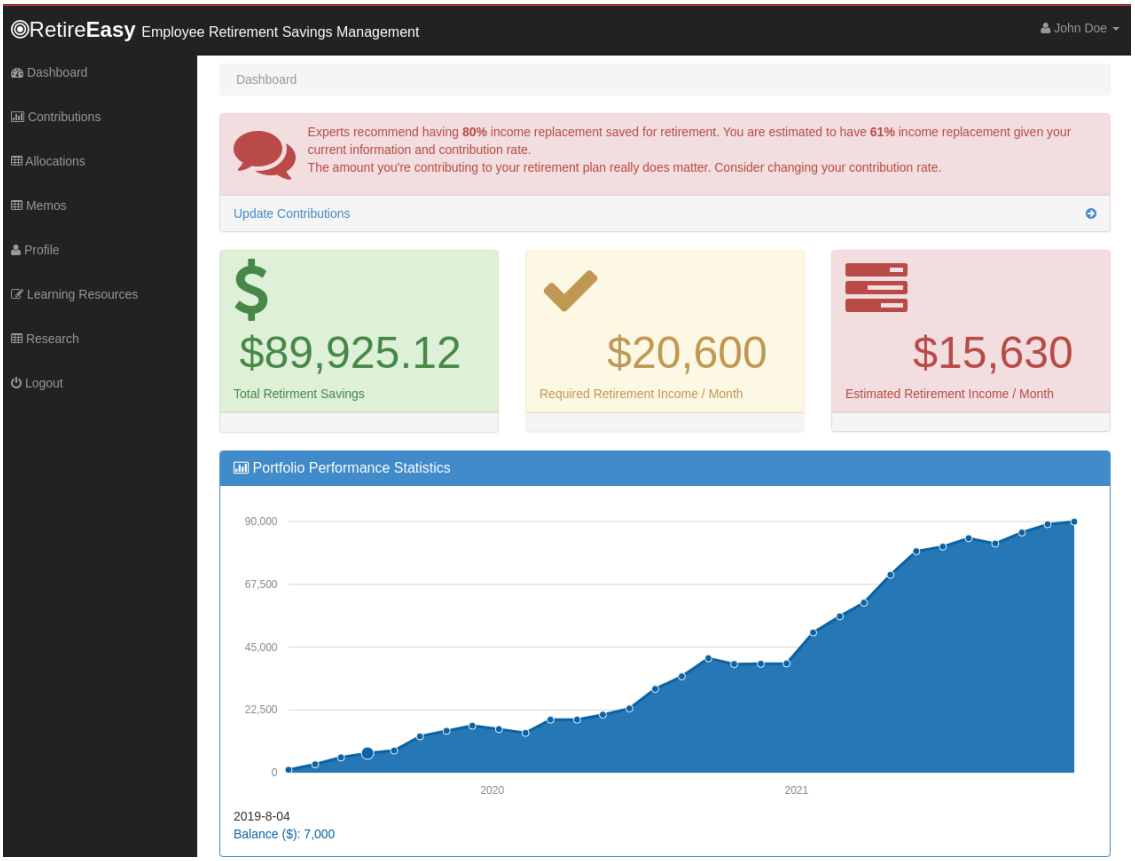


Figure 11 – OWASP NodeGoat user page

This project uses the containerised version of the NodeGoat application, consisting of two docker containers, one for MongoDB and the other for running the application.

4.3 Security Control Tools

After deciding which targeted project to use, security control automation tools were selected with several considerations. One of the primary considerations regarding tool characteristics was their flexibility and customisation, so open-source tools were prioritised. Part of their flexibility includes the integration with the CI tool Jenkins. Graphical user-interfaces (GUI) tools might be easy to use but be a barrier to automation. As a result, the tools should support command-line interfaces (CLIs). Following are the tools that meet these criteria.

4.4 Code review and software composition analysis (SAST)

Code review and software composition is achieved by performing extensible and automated source code analysis in a static setting, meaning that a running application is not required.

4.4.1 Code Review Audit Script Scanner (CRASS)

The implementation began tackling the project by setting aside the fact that the vulnerabilities of the NodeGoat project were previously known.

In a "White-box environment" where the source code is available to be audited, an automatic code inspection tool was used to familiarise the project. The most direct tool found was the Code Review Audit Script Scanner (CRASS) (CRASS, 2022).

This solution includes a one-shell script for secure code scanning (grep-it.sh), having scanning patterns defined for major programming languages such as Java, JSP, .NET, PHP, HTML, Android, iOS, Python, Ruby, C, etc. The grep-it tool uses standard command-line tools, particularly the grep, to find and highlight strings that might be intriguing for analysis from a security perspective. It doesn't require the installation of other dependencies and is contained in a shell script, no building of the source code is required.

Results obtained from the script execution and gathered in a new directory (grep-output) are split into various files according to their priority (1-9). Lower values assure more certainty that defects were identified.

The execution of the grep-it on the NodeGoat project confirms our expectations and has several strings found highlighted defects presented in the source code. Prompting the next steps in the pipeline for code review and software composition analysis.

4.4.2 OWASP Dependency-Check

OWASP Dependency-Check (OWAP Dep. Check, 2022), as implied by the name, was developed by the OWASP foundation as a free and open-source project. It's a Software Composition Analysis (SCA) tool that scans the files contained in the project to analyse hits dependencies in libraries, file names, manifest, packages, vendors, and versions. Them compares them with the Common Vulnerabilities and Exposures (CVE) and National Vulnerability Database (NVD) databases. It generates a report summarising the vulnerabilities and associated CVE vulnerabilities. The highest confidence ranking assures how certainly the defect was identified (Figure 12). Discovered evidence is displayed according to their file location in the project, publicly disclosed problems information is added, and a rank attributed by is severity rating according to the Common Vulnerability Scoring System (CVSSv3) on a scale of 0-10 (Table 2).

Table 2 - Common Vulnerability Scoring System (CVSS) (CVE Vulnerability, 2022)

Severity	**Base Score**
Informal	0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

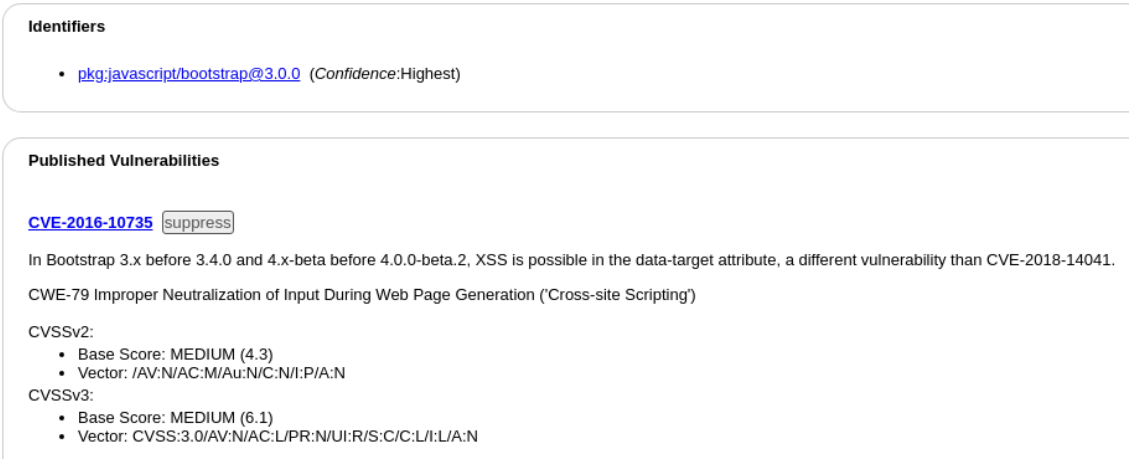


Figure 12 – Portion of Dependency Check Report

At this early stage in the deployment process, this tool pinpoints vulnerabilities in source code dependencies, directing for upgrading those dependencies or enabling decisions for alternatives to be made.

4.4.3 SonarQube

SonarQube started in 2007, named initially Sonar was developed by SonarSource (SonarSource S.A. 2022 (1)). It is free to use under the GNU Lesser General Public License but has additional features accessible only through paid license plans or plugins. The tool inspects the code for security defects like bugs, vulnerabilities, and hardcoded sensitive information.

The SonarQube solution adopted is composed of two elements the docker container, which runs the server and database, and the sonar scanner.

While the SonarQube Server runs, the sonar scanner is executed against the project source code. When finished, an analysis report is generated with the obtained findings (Figure 13). This report can be accessed through the web dashboard interface or extracted into YAML files through the command line.

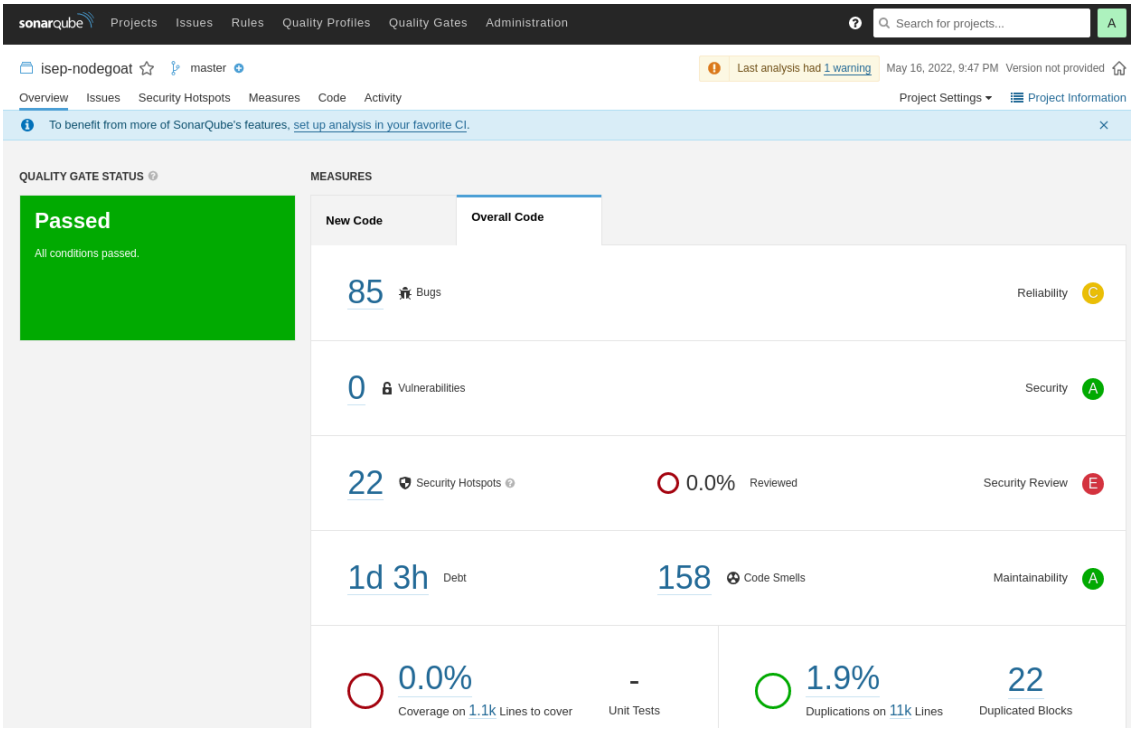


Figure 13 – SonarQube dashboard report

This tool's ability to track vulnerabilities and its comprehensible interface enhance the quality and sustainability of code development while increasing software development productivity.

4.5 Web application user interface (DAST)

After analysing the source code, the application is initiated in an isolated environment. The focus now shifts to potential disclosures of sensitive information, mainly data injection. Dynamic application security testing (DAST) flows through the user interface (UI) layout while doing security analysis, this encompasses one of the main challenges for web applications automated testing. Surpassing this challenge requires not only security scanning but also web UI automation.

Testing scenarios:

- Logging in with different users or wrong accounts
- Logging users out for session management testing
- Creating a new user account
- Brute-forcing a user account login
- Fuzz data input
- Script Injection

The following tools were used to achieve the security UI testing coverage of the application:

- OWASP ZAP
- Selenium

4.5.1 OWASP ZAP

Developed by the OWASP foundation, ZAP, also known as Zed Attack Proxy, is an open-source web application security scanner (OWASP ZAP, 2022). This application performs two distinct scanners, namely the spider and the active scan.

Purposely, the spider scan attempts to discover all the URLs on a particular website. Starting from a list of previously known URLs (seeds), it then follows all the hyperlinks extensively until no new resources are found. The active scan looks for security vulnerabilities by sending malicious requests, such as XSS, SQL injection, CSRF token, etc. The collected information is then presented in an analysis report (Figure 14).

Summary of Alerts

Risk Level	Number of Alerts
High	3
Medium	4
Low	6
Informational	1
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
Cross Site Scripting (DOM Based)	High	2
Cross Site Scripting (Reflected)	High	3
SQL Injection	High	2
CSP: Wildcard Directive	Medium	23
Directory Browsing	Medium	2
Vulnerable JS Library	Medium	2
X-Frame-Options Header Not Set	Medium	22
Application Error Disclosure	Low	1

Figure 14 – OWASP ZAP report example

ZAP spider scan explores all potential URLs and web resources. Nevertheless, a web resource might require manual intervention, such as authenticated resources or user registration. Another component of ZAP is the proxy tool that in conjunction with a web automation UI framework guides ZAP through web pages that require manual input, allowing their inspection.

4.5.2 SeleniumBase

Selenium is an open-source tool for UI automated testing framework created in 2004 by Jason Huggins. SeleniumBase is a project derived from the Selenium WebDriver APIs. Also, an open-source tool was developed in python to be a CLI application to conduct selenium testing scripts (Michael Mintz, 2022). SeleniumBase allows testing to be simulated in memory, meaning that the actual browser launch is facultative. This improves the testing stability and shortens the duration of his execution. It is used to replicate human behaviours, like text input, and in conjunction with ZAP proxy, it allows the identification of security issues in web resources that would require manual inspection.

4.5.3 Jenkins

An automation tool that combines all the tools and parts in a continuous flow is required to conduct the testing process. Jenkins, a Java-based open-source CI/CD tool, was chosen to automate the software analysis.

In 2004 Kohsuke Kawaguchi began the development of Hudson at Sun Microsystems. In 2011 after Oracle acquired Sun Microsystems, Hudson was forked to is now Jenkins. Jenkins remains maintained by the open-source community to this day, while the Hudson project has been terminated since 2016.

4.6 Pipeline

As mentioned earlier, the CI/CD software development pipeline drives the process from software commitment through various stages until the deployment in the target environment.

Focussing on the security aspects introduced by the DevSecOps pipeline and by realising that a fair amount of security automation tests are unbiased to the business subject of the software developed, it was chosen to deliver a pipeline that would enable an impartial analysis of the software from the main CI/CD pipeline. This division allows more versatility, teams can run the pipeline at the beginning of their CI/CD Pipeline, individual security controls can be selected, and developers can use the pipeline often during the development before committing the code to the main repository. Another reason for splitting the solution into an individual pipeline was to conceal it from the remaining environments. Allowing a possible defected solution to be examined in another environment contributes to maintaining the resilience of the main environments since the defects are treated in a prior isolated one. Taking these characteristics into account, the following DevSecOps pipeline was designed:

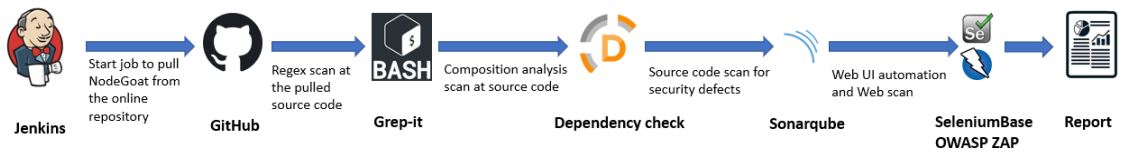


Figure 15 - DevSecOps Pipeline

The pipeline implementation (Figure 15) in Jenkins starts by retrieving the source code from a Git repository. This must be triggered in Jenkins, but it can be changed for beginning at every new commit or at a specific time. The first stage also includes creating a directory inside the project to store the artefacts generated by the security tests.

After downloading the source code, the first security control is grep-it (25). Since grep-it has customisable options that can be defined in the script, it will also be downloaded from a Git repository to consider these options. Like all the remaining security control stages, this stage can be skipped, but as was acknowledged with the target model project, this simple tool is capable of quickly identifying defects in the source code.

The OWASP Dependency Check (23) is the next stage to compare the source code elements with the elements from the CVE and NVD databases. This stage generates a report with the scan information, possible defects identified and information about them.

Still, for source code analysis, the next stage begins by validating the Sonar server availability. The Sonar server runs in a docker container that should already be available since it contains all the previous generated reports and configurations for the project or scans made in it. Jenkins initialises the sonar scanner for the source code, and results are reported through the sonar server.

After the source code analysis controls, the pipeline initialises the application. The NodeGoat application from the target project is available through a docker container. After initialised, it uses port 4000 to communicate.

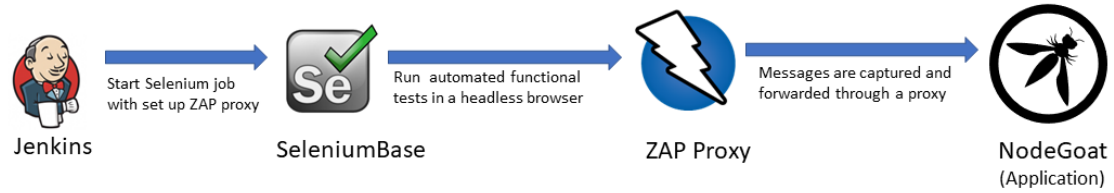


Figure 16 – Selenium – ZAP proxy functional tests

The next stage begins with the OWASP ZAP spider scan (28) to retrieve potential URLs and web resources. The NodeGoat application has a login page that prevents resources from being accessible without login. The SeleniumBase tool (Figure 16) included in this stage allows the scan to imitate the manual input on the website. Login is performed in conjunction with ZAP to proxy the Selenium execution, so previously inaccessible resources can be covered. These

resources are passively collected and joined to the previous resources retrieved from the spider scan.

To improve the application coverage, the Selenium script includes the following tasks:

- Open NodeGoat application.
- Insert login credentials and press submit button.
- Access the contributions page and press submit button.
- Access allocations, memos and profile pages.

Functional tests can be integrated into this stage to cover the application resources further or specify the coverage of a particular application area. Collected resources are used in the following scan. ZAP active scan picks up from previously collected resources and uses known attacks against them to compromise the application. Results can be exported in three formats JSON, XML or HTML. The last format was chosen for reporting in this implementation.

Aiming to simplify the accessibility of this pipeline as a resource for any member of the software development team, scan reports and stage results are shared through an e-mail to a previously defined address. The goal was to encapsulate all the pipeline processes into a framework that, in its most straightforward approach, requires only the source code to be committed to the Git repository, while at the same time enabling the selection of which security controls to execute for better suit the software development needs.

5 Result Analysis

This chapter presents the obtained results from all the scans performed at the target project, and how they've performed at OWASP top 10 vulnerabilities detection.

5.1 Grep-it

Grep-it (4.4.1) contains a configuration area to enable or disable languages or components to search for. These correspond to specific functions and regular expressions suited to the desired configuration.

As suggested by the script author, the first runs were left with all the options checked ('true'), but the number of results, 267 files, was an overwhelming task to analyse. Since the NoadGoat project uses JavaScript, Node.js and HTML, the configuration used was the following:

- DO_HTML="true"
- DO_JAVASCRIPT="true"
- DO_MODSECURITY="true"
- DO_MALWARE_DETECTION="true"
- DO_CRYPTO_AND_CREDENTIALS="true"

The first two options (DO_HTML and DO_JAVASCRIPT) are language related, the following tree (DO_MODSECURITY, DO_MALWARE_DETECTION, DO_CRYPTO_AND_CREDENTIALS) are security related. The scan generated 85 files in approximately one minute and forty seconds.

In the security spectrum, there were only two files with critical information. They were collected in the generated file 4_cryptocred_password.txt (Figure 17). The scan was able to identify

database credentials in those files. The profile-test.js is a tutorial test example, and db-reset.js is used to set up the mongo database and load “dummy data” into it.

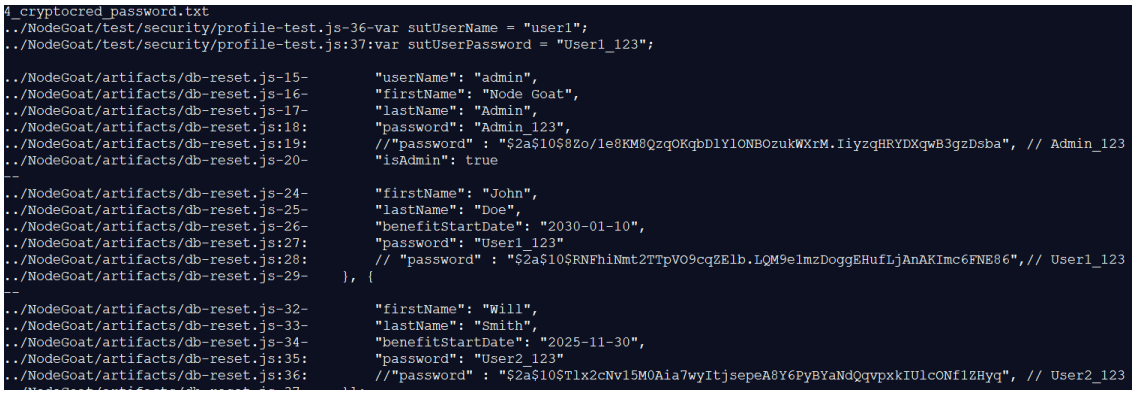


Figure 17 – grep-it scan, extract from 4_cryptocred_password.txt

Reminded that this is a regular expression searching tool, many ‘False positives’ were expected. Despite this, it proved the scan’s effectiveness in quickly finding the words that matched the regular expressions. As previously mentioned, this tool was introduced to roughly explore the project contents, and to find intriguing elements that could guide the exploration of further vulnerabilities. For example, the cypher algorithms identified in the generated files below be targeted for eventual exploitation:

- 5_cryptocred_ciphers_md2.txt - MD2. Security
- 5_cryptocred_ciphers_md5.txt - MD5. Security
- 5_cryptocred_ciphers_rc2.txt - RC2 cypher
- 5_cryptocred_ciphers_rc4.txt - RC4 cypher

5.2 Dependency-check

Dependency check (4.4.2) scan was executed with three options the project name, the output directory, and the scan targeted directory, the NodeGoat source code project. It ran in 87 seconds, but the time might differ if the local database requires an update, this is evaluated at the beginning of the scan.

Results from the scan:

- Dependencies scanned: 16279 (11653 unique)
- Vulnerable Dependencies: 109
- Vulnerabilities Found: 143

Vulnerabilities classified according to the CVSS base score (Table 2):

Table 3 - Common Vulnerability Scoring System

CVSSv3_BaseScore	Occurrences
9.8	21
9.1	4
8.8	2
8.7	1
8.6	3
8.1	4
7.8	5
7.5	33
7.4	3
7.3	3
7.2	6
7.1	2
7	1
6.5	6
6.3	1
6.1	27
5.9	2
5.6	5
5.5	2
5.3	10
4.4	1
2.5	1
TOTAL	143

The most prevalent vulnerabilities in Table 3 were:

- Regular expression denial of service attacks (ReDoS) - Regular Expression implementations may reach extreme situations that cause them to work very slowly
- Denial-of-service condition (DoS) - use 100% of its processor time and unable to process any other incoming requests until the process is restarted
- Symlink, also known as 'Zip-Slip'., could allow a local attacker access to unintended directories
- Prototype pollution protection and alter the Object prototype - This allows attackers to override properties that will exist in all objects, which may lead to DoS or Remote Code Execution
- Cross-Site Scripting (XSS)
- Man-in-the-middle attack

- Modification of Assumed-Immutable Data (MAID), which allows a malicious user to modify the prototype
- Command Injection via the template function

Regarding the OWASP top 10 vulnerabilities, the dependency check scan detected eight of them:

- Injection [Detected]
- Broken Auth [Undetected]
- XSS [Detected]
- Insecure Direct Object References [Detected]
- Misconfiguration [Undetected]
- Sensitive Data [Detected]
- Access Control [Detected]
- CSRF [Undetected]
- Insecure Components [Detected]
- Redirects [Detected]

All of these were solved by updating the dependencies to a recent version.

5.3 SONARQUBE

SonarQube (4.4.3) scan took less than one minute to execute, overall results were:

- 85 Bugs
    - 8 Major
        - 3 FALSE POSITIVE - These are part of the tutorial resources as examples.
        - 4 CSS related
        - 1 Noncompliant Code
    - 77 Minor – HTML related

- 0 Vulnerabilities

- 128 Code Smells, which constitute “a maintainability issue that makes your code confusing and difficult to maintain.” (SonarSource S.A. 2022 (2))
    - 141 Major
    - 6 Critical
    - 5 Minor
    - 6 Info

- 22 Security Hotspots, meaning that “a security-sensitive piece of code is highlighted, but the overall application security may not be impacted.” (SonarSource S.A. 2022 (3)).

The identified security hotspots were composed of both false and true positives.

False positives include tree authentication hotspots. These identify the same hard-coded credentials from db-reset.js that the grep-it scan found. This file is used to set up the project database, including the dummy data. Another false positive identified one command injection security hotspot from an exec call in a grunt file. Grunt files are used to automate tasks, so despite being in the source code, they are not part of the NodeGoat solution. Seven false positives regarding week cryptography pointed to the Math.random's pseudo-random generation, except that none of the pseudo-random data is used for encryption. There were four false positive flagged links with "target=_blank" and two hardcoded IP addresses on the 'Others' category.

True positives include two ReDOS associated with regular expressions that can cause a denial of service, and three Code Injection (RCE) issues are highlighted.

Overall, the expectations regarding SonarQube were shattered because the scan just detected one potential type of vulnerability from the OWASP top 10. This came as a surprise because SonarQube is one of the recommended SAST tools from OWASP.

- Injection [Potentially Detected]
- Broken Auth [Undetected]
- XSS [Undetected]
- Insecure Direct Object References [Undetected]
- Misconfiguration [Undetected]
- Sensitive Data [Undetected]
- Access Control [Undetected]
- CSRF [Undetected]
- Insecure Components [Undetected]
- Redirects [Undetected]

On the other hand, it highlights the difficulty imposed on the tools to passively find security problems regarded with authentications, access control, insecure cryptography, etc. The current state of SAST tools predominantly only allows dependencies flaws to be identified.

5.4 Selenium and ZAP

From the beginning, the spider ZAP scan (4.5.1), detected 95 URLs in about three minutes. This number improved to 98 URLs after the Selenium (4.5.2) functional test. The active scan (4.5.1)

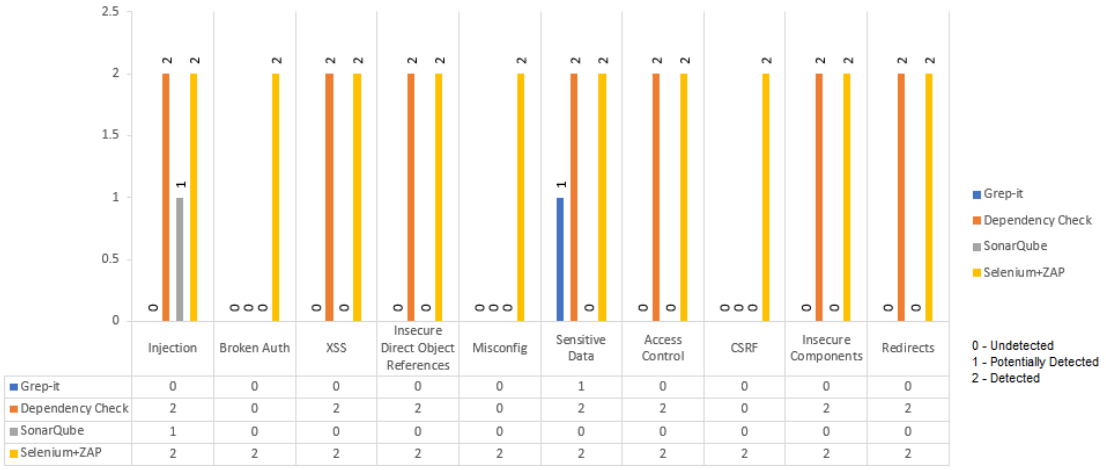
that followed took almost 15 minutes to finish, with surprising results (Table 4) when compared to the results obtained from the previous scans.

Table 4 - ZAP active scan results

Vulnerability	**Risk Level**	**Nº of Instances**
Cross-Site Scripting (DOM Based)	High	2
Cross-Site Scripting (Reflected)	High	3
SQL Injection	High	3
CSP: Wildcard Directive	Medium	3
Directory Browsing	Medium	2
Vulnerable JS Library	Medium	2
X-Frame-Options Header Not Set	Medium	26
Application Error Disclosure	Low	1
Cookie without SameSite Attribute	Low	4
Cross-Domain JavaScript Source File Inclusion	Low	26
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	53
Timestamp Disclosure - Unix	Low	6
X-Content-Type-Options Header Missing	Low	40
Information Disclosure - Suspicious Comments	Informational	28

It should be taken into consideration that these were the first DAST scan performed, and so require a running NodeGoat solution. The results can be consulted on the generated ZAP report, which contains a description of all vulnerabilities alongside relevant information like performed attack description, possible solutions and CWE id.

Selenium and ZAP scan results check all the OWASP top 10 vulnerabilities (Figure 18).



	Injection	Broken Auth	XSS	Insecure Direct Object References	Misconfig	Sensitive Data	Access Control	CSRF	Insecure Components	Redirects
Grep-it	0	0	0	0	0	1	0	0	0	0
Dependency Check	2	0	2	2	0	2	2	0	2	2
SonarQube	1	0	0	0	0	0	0	0	0	0
Selenium+ZAP	2	2	2	2	2	2	2	2	2	2

Figure 18 – Global representation of the obtained results from different scans

Equipped with this information through the e-mail reception, teams and companies might follow the recommendations to solve the vulnerabilities or take decisions about which alternatives are necessary.

5.5 Summary

Overall opinion regarding the obtained results its very satisfactory. The grep-it.sh results (5.1) despite being false positives, enriched the knowledge about the targeted project, and proved its effectiveness in detecting the specified regular expressions. Since it is open source, users can improve the script to better suit their needs. Dependency Check scan results accentuate the importance of the included dependencies as a security concern. The obtained results showed that defects contained in the imported dependencies were capable to compromise the overall solution. SonarQube scan results returned a limited number of security defects, nevertheless it identified other elements suited for improvement. Important factors for the overall code quality and maintainability of the solution. The combination of OWASP ZAP scans with Selenium function tests recognised a vast number of vulnerabilities. A targeted running application is required to be able to perform the scans, but Selenium tests are not obligatory. If desired Selenium tests can be integrated later with OWASP ZAP during the functional tests section of the DevSecOps pipeline. The more tests are conducted with selenium and OWASP ZAP proxy combined, the more coverage OWASP ZAP will conduct in the active scan.

As showed by the Figure 18 the following resolutions regarding the security subject can be ascertained:

- In two of the scans the results were not conclusive, SonarQube – Injection, and Grep-it – Sensitive Data.

- Broken Auth, Miss Configuration, CSRF are only detected by Selenium+Zap Scans.
- The remained vulnerabilities were detected in Dependency Check and Selenium+Zap. Since they’ve detects all or most of the OWASP top 10 vulnerabilities, they were the most effective security controls.

6 Conclusion

In this final chapter, the objectives set in Section 1.3 are revisited, with an overview of the accomplishments and what form the project perspective can still be added or improved in future work.

6.1 Outcomes

The TAR methodology identifies three distinct and independent roles (Wieringa, 2014):

- **Technical Researcher:** Conducted the research and design for the solution, by structuring the guidelines for is implementation.
- **Empirical Researcher:** Concluded the research by applying the guidelines into an artefact. The artefact was practised, and its results were evaluated. TAR studies are single-case studies, but the application of the artefact in similar projects can optimistically attain similar results.
- **Helper Client:** This project focused on Web Applications and was tested in a Node.js solution. But it can be embraced by other web projects, providing that the language for their development is compatible, granting the engineering community an improvement for the overall quality of the developed software.

Following the TAR methodology, these project objectives were resolved:

- **Assess what characterises a DevSecOps pipeline compared to a DevOps pipeline.**
  To accomplish this object, it was first acknowledged that DevSecOps, when implemented correctly, can increase overall software quality. The “shift left” approach helps reduce the cost of software security defects by detecting them earlier in software development. State

of the art in Section 3 resulted from the research and analysis of DevOps and DevSecOps, which is crucial to identify what they share in common and what DevSecOps added to the DevOps.

- **Implement the automated portion from the DevSecOps pipeline for a specific case, retrieve knowledge about how effective they are at identifying vulnerabilities and their impact on the overall software delivery.**
  After getting to know the requirements for the DevSecOps pipeline, the security controls' tools were analysed and afterwards selected, Section 4.3. The target project that served to build and evaluate the pipeline effectiveness was also identified in Section 4.2. The solution focused on the automation components of the DevSecOps pipeline, and the required user intervention was kept at a minimum. Using all open-source tools allows them to be customised to best suit the project's needs. This objective was achieved with a pipeline that contained four scan tools to cover the OWASP top 10 vulnerabilities detection. These vulnerabilities were indeed found in the target project, successfully concluding the first part of this objective, Section 5.
  Regarding the impacts on the software development cycle, the average time spent by these scans is inferior to 30 minutes. By the targeted project contents, it can be extrapolated that the more complex the project is, the longer these scans will take to run.
  But for instance, the grep-it scan isn't a particularly efficient scan to be executed at every deployment since it returns many false negative results. Instead, new developers or penetration testers should use it for project exploration to become familiar with the project and its contents. Dependency-check is also relevant if new dependencies are added to the project or the previous scan was taken some time ago. The scan that took longer to execute was the active ZAP scan, it used the data collected by the spider ZAP scan and the Selenium with ZAP proxy. The duration of this scan can be reduced by running the Selenium with ZAP proxy to perform functional or unit tests and then running the active ZAP scan just with the data collected by those tests. In doing so, tests can be gradually performed at the newly developed components instead of a broader data collection. This concludes the second objective of this project.

6.2 Limitations and Future work

DevSecOps is an iterative and cyclic process, as every iteration adds new data and experience, decisions are made to solve problems or improve the delivery plan upon previous results. The solution proposed in this project focused on the automated activities that contemplate the security controls. As mentioned in the Section 3.4, the DevSecOps methodology begins with the education of all the members involved, passes through the software analysis, and extends into operations and infrastructure realm, a full DevSecOps solution should take into consideration those factors. The conditions imposed on the tools used in the solution developed may not

represent the best solution for all software development environments. Some of these could be subject to change, for instance, if an open-source nature isn't a restriction.

6.3 Contributions of the work

Software exploitation threats have become more prevalent, and penalties to an organisation's reputation and monetary implications derived from these attacks highlight the need for further software development control. Of their exposed nature, web applications are more accessible to those with bad intentions. So, efforts must be made to minimise and solve defects in the development and delivery of software. DevSecOps prioritised software security, and the solution proposed by this project attempts to solve the problem of integrating automatic security tools into the DevOps pipeline. This flexible approach allows the company to decide what strategy to follow up regarding the frequency of security check-ups and which scans should be done at any given time during the software development process.

References

451 Research, 2018	Devsecops realities and opportunities. https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/devsecops-realities-opportunities-451.pdf
Arthur Hicken,2018	The Shift-Left Approach to Software Testing. https://www.stickyminds.com/article/shift-left-approach-software-testing. Accessed 30 January 2022. Adapted from Jones Capers, Applied Software Measurement: Global Analysis of Produtivity and Quality, (1996)
Caluza, Las Johansen, 2020	Development of J48 Algorithm-Based Application in Predicting Teacher's Techno-Pedagogical Competence. 18. 293-310.
CRASS, 2022	CRASS. https://github.com/floyd-fuh/crass. Accessed 5 June 2022
CVE Vulnerability, 2022	2021 CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html. Accessed 30 January 2022
CWE, 2022	Awesome devops. https://github.com/awesome-soft/awesome-devops. Accessed 30 January 2022
GitHub, 2022(1)	Awesome devsecops. https://github.com/devsecops/awesome-devsecops. Accessed 30 January 2022
GitHub, 2022(2)	Gokarna, Mayank, and Raju Singh. “DevOps: A Historical Review and Future Works.” In 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), 366–71. Greater Noida, India: IEEE, 2021. https://doi.org/10.1109/ICCCIS51004.2021.9397235.
Gokarna, et al., 2021	Snyck-labs. Super vulnerable todo list application. https://github.com/snyk-labs/nodejs-goof. Accessed 2 December 2021
Goof, 2021	Hacjzin. A modern vulnerable web app. https://github.com/rapid7/hackazon. Accessed 15 January 2022
Hackzon, 2021	Myrbakken, Håvard, and Ricardo Colomo-Palacios. “DevSecOps: A Multivocal Literature Review.” In Software Process Improvement and Capability Determination, edited by Antonia Mas, Antoni Mesquida, Rory V. O’Connor, Terry Rout, and Alec Dorling, 770:17–29. Communications in Computer and Information Science. Cham: Springer International Publishing, 2017. https://doi.org/10.1007/978-3-319-67383-7_2.
Håvard Myrbakken, Ricardo Colomo-Palacios, 2017	IBM Security and the Ponemon Institute: Cost of a Data Breach Report 2021. Retrieved from: https://www.ibm.com/security/data-breach
IBM Security,Ponemon Institute, 2022	Identity Theft Resource Center’s 2021 Annual Data Breach Report Sets New Record for Number of Compromises, 2022. Retrieved from : https://www.idtheftcenter.org/post/identity-theft-resource-center-2021-annual-data-breach-report-sets-new-record-for-number-of-compromises/ Accessed 23 January 2022
Identity Theft Resource Center, 2022	Why enterprises must adopt devops to enable continuous delivery. The Journal of Information Technology Management
Jez Humble, Joanne Molesky 2011	Value Proposition, Problem Statement e Elevator Pitch
João R Andrade, 2019	The Cascading Costs of Waterfall, accessed 4 January 2022. https://medium.com/@joneswaddell/the-cascading-costs-of-waterfall-5c3b1b8beaec
Justin Jones, Scott Waddell, 2019	Koen, P., Ajamian, G., Boyce, S., Clamen, A., Fischer, E., Fountoulakis, S., ... Seibert, R. (2002). “Fuzzy-Front End: Effective Methods, Tools and Techniques”. Em P. Belliveau, A. Griffin, & S. Soremrmeyer (Eds.), PDMA Toolbook 1 for New ProductDevelopment(pp. 2–35). New York: John Willey & Sons.
Koen. Ajamian, et al., 2002	Lwakatare, Lucy Ellen, Pasi Kuvaja, and Markku Oivo. "Dimensions of DevOps." In International conference on agile software development, pp. 212-217. Springer, Cham, 2015
Lucy E. Lwakatare, et al., 2017	Luz, Welder Pinheiro, Gustavo Pinto, and Rodrigo Bonifácio. "Adopting DevOps in the real world: A theory, a model, and a case study." Journal of Systems and Software 157 (2019): 110384

Luz et al., 2019	The top data breaches of 2021. https://www.securitymagazine.com/articles/96667-the-top-data-breaches-of-2021. Accessed 30 January 2022
Maria Henriquez, 2021	Hüttermann, Michael. "Beginning devops for developers." Apress, Berkeley, CA, 2012.
Michael Hüttermann, 2012	SeleniumBase. https://seleniumbase.io/. Acessed 16 April 2022
Michael Mintz, 2022	N. Forsgren, M. C. Tremblay, D. VanderMeer, and J. Humble, ‘‘DORA platform: DevOps assessment and benchmarking,’’ May 2017
N. Forsgren et al.,2017	Nicola, Susana, Eduarda Pinto Ferreira, e JJ Pinto Ferreira (2012). “A novel framework for modeling value for the customer, an essay on negotiation”. In: International Journal of Information Technology & Decision Making 11.03, pp. 661–703.
Nicola, Ferreira, et al., 2012	Nicole Forsgren PhD, Jez Humble, Gene Kim. "Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations". Published by: IT Revolution, 2018. ISBN:1942788355, 9781942788355. 2018
Nicole Forsgren et al., 2018	OWASP Dependency Check. https://owasp.org/www-project-dependency-check/. Accessed April 2022
OWAP Dep.Check, 2022	OWASP NodeGoat. https://github.com/OWASP/NodeGoat. Accessed 14 January 2022
OWASP NodeGoat, 2021	OWASP ZAP. https://owasp.org/www-project-zap/. Accessed April 2022
OWASP ZAP, 2022	OWASP Top Ten - https://owasp.org/www-project-top-ten/. Accessed 30 January 2022
OWASP, 2022 (1)	Patrick Van Der PIJL (2012). “How to really understand your customer with the value proposition canvas”. https://designabetterbusiness.com/2017/10/12/how-to-really-understand-your-customer-with-the-value-proposition-canvas/. Accessed 14 November 2021
Patrick Van Der PIJL, 2012	Pedra, Mauro Lourenço, Mônica Ferreira da Silva, and Leonardo Guerreiro Azevedo. “DevOps Adoption: Eight Emergent Perspectives.” ArXiv:2109.09601 [Cs], September 20, 2021. http://arxiv.org/abs/2109.09601.
Pedra et al., 2021	Rajapakse, Roshan Namal, Mansooreh Zahedi, and Muhammad Ali Babar. “An Empirical Analysis of Practitioners’ Perspectives on Security Tool Integration into DevOps.” In Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 1–12. Bari Italy: ACM, 2021. https://doi.org/10.1145/3475716.3475776.
Rajapakse et al., 2021	Roshan N. Rajapakse, Mansooreh Zahedi, M. Ali Babar, Haifeng Shen. "Challenges and solutions when adopting DevSecOps: A systematic review, Information and Software Technology, Volume 141, 2022, 106700, ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2021.106700.
Rajapakse et al., 2021 (2)	Rajapakse, Roshan Namal, Mansooreh Zahedi, and Muhammad Ali Babar. “An Empirical Analysis of Practitioners’ Perspectives on Security Tool Integration into DevOps.” In *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–12. Bari Italy: ACM, 2021. https://doi.org/10.1145/3475716.3475776.
Rajapakse et al., 2021 (3)	Rajavi Desai and T N Nisha. “Best Practices for Ensuring Security in DevOps: A Case Study Approach.” Journal of Physics: Conference Series 1964, no. 4 (July 1, 2021): 042045. https://doi.org/10.1088/1742-6596/1964/4/042045.
Rajavi Desai, T N Nisha, 2021	Kumar, Rakesh, and Rinkaj Goyal. “Modeling Continuous Security: A Conceptual Model for Automated DevSecOps Using Open-Source Software over Cloud (ADOC).” Computers & Security 97 (October 2020): 101967. https://doi.org/10.1016/j.cose.2020.101967.
Rakesh Kumar, Rinkaj Goyal, 2020	Rodríguez, Pilar, Alireza Haghighatkhah, Lucy Ellen Lwakatare, Susanna Teppola, Tanja Suomalainen, Juho Eskeli, Teemu Karvonen, Pasi Kuvaja, June M. Verner, and Markku Oivo. “Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study.” Journal of Systems and Software 123 (January 2017): 263–91. https://doi.org/10.1016/j.jss.2015.12.015.
Rodríguez et al., 2017	Shahin, Mojtaba, Muhammad Ali Babar, and Liming Zhu. “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices.” IEEE Access 5 (2017): 3909–43. https://doi.org/10.1109/ACCESS.2017.2685629.
S.Mojtaba, et al., 2017	Securing Web Application Technologies SWAT Checklist. https://www.sans.org/cloud-security/securing-web-application-technologies/. Accessed 30 January 2022

SANS, 2022	SonarQube. https://www.sonarqube.org. Accessed 17 April 2022
SonarSource S.A 2022 (1)	Issues. https://docs.sonarqube.org/latest/user-guide/issues/. Accessed 10 June 2022
SonarSource S.A 2022 (2)	Security Hotspots. https://docs.sonarqube.org/latest/user-guide/security-hotspots/. Accessed 10 June 2022
SonarSource S.A 2022 (3)	Annual number of data breaches and exposed records in the United States from 2005 to 2020. Source: Identity Theft Resource Center, Retrieved from: https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/ Accessed 23 January 2022
Statista, 2022	Tesla and Adobe: Why Continuous Deployment May Mean Continuous Customer Disappointment, accessed 4 January 2022. https://steveblank.com/2014/01/06/15756/
Steve blank, 2014	Maven Security. Web Security Dojo. https://www.mavensecurity.com/resources/web-security-dojo. Accessed 15 January 2022
Web Security Dojo, 2020	The ultimate list of open source devops tools. https://xebialabs.com/the-ultimate-devops-tool-chest/open-source/. Accessed 30 January 2022
Wieringa, 2014	Technical Action Research. In R. Wieringa (Ed.), Design Science Methodology for Information Systems and Software Engineering (pp. 269–293). Springer Berlin Heidelberg. https://link.springer.com/chapter/10.1007/978-3-662-43839-8_19
XebiaLabs, 2022	CVE Vulnerability. https://www.imperva.com/learn/application-security/cve-cvss-vulnerability/. Accessed 1 May 2022

1 Attachments

1.1 Value Analysis

This chapter explains the steps taken from the project idea to the opportunity definition and strategy. It presents the concept models for the innovation process and opportunity analysis. Afterwards, the concepts of the different types of value, the gains, and pains associated with the chosen solution are presented and followed by the presentation of the value proposition, with an illustration of the canvas business model. Near the end, the chapter analyses and selects the requirements with support from the Quality Function Deployment (QFD) system. This chapter uses the Analytic Hierarchy Process (AHP) tool to identify the most critical criterion for a DevSecOps approach.

1.1.1 New Concept Development Model

The innovation process of a business or a product consists of three parts:

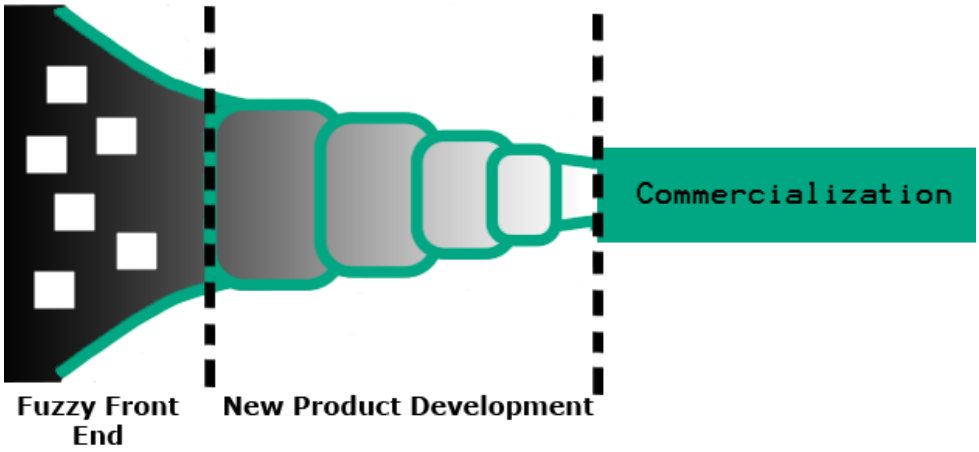


Figure 19 – Innovation Process (Adapted from Koen. Ajamian et al., 2002)

- Fuzzy Front End (FFE) - it's the least formal area, it aims to generate ideas that eventually will consist in opportunities.
- New Product Development (NPD) - After opportunities are identified in FFE, this area engages in developing a rigorous plan for the proposed objectives.
- Commercialization - the final area where the developed product is sold.

The Fuzzy Front End approach can be chaotic and experimental. Therefore a new methodology was developed, the New Concept Development Model (NCD), intended to define a common terminology for all representations of o this process.

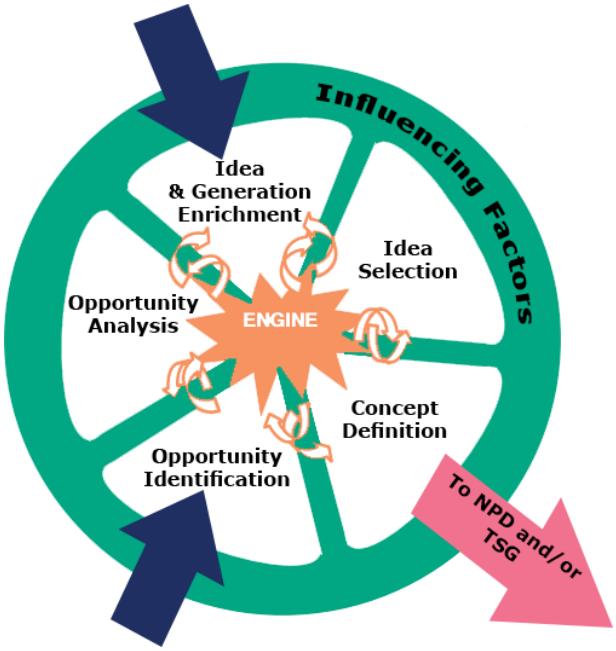


Figure 20 – New Concept Development (NCD) (Adapted from Koen. Ajamian et al., 2002)

The model contains the model engine at its centre, where the factors related to the company reside, leadership, culture, and business strategy. These factors affect the transitions between the other five elements around the engine. The circular shape of the model and the internal arrows point to the interaction between activities in each element.

Represented by the surrounding circumference are the external factors that influence the company through all the processes until the final one, commercialisation.

External arrows point to the starting elements, opportunity identification, idea generation, and enrichment. The arrow pointing to the outside of the circumference represents the ending of the NCD/FFE process and the transition to NPD or technology stage-gate (TGS).

In the inner circle reside five elements, each representing a controllable activity. Regarding this project, they consist of the following descriptions:

- Opportunity Identification – By adding a security-by-design approach to the DevOps paradigm, the overall quality of the software will be improved.
- Opportunity analysis – When adopting a dynamic continuous delivery/deployment on a DevOps approach, it is understood that software released is fast and frequent. But despite being a concern, security is usually perceived as an inhibitor for DevOps agility. Significant benefits can be obtained if DevSecOps is done correctly.

- Idea generation – After identifying an opportunity, different ideas were considered to solve the problem. A study was conducted to understand what DevSecOps tools and techniques suited Oracle SOA.
- Idea selection – After identifying the tools and techniques available, the most promising were chosen to resolve the problem.
- Concept definition – Development of a Proof of Concept to assert the feasibility of DevSecOps.

1.1.2 Value

"Value has been defined in different theoretical contexts as need, desire, interest, standard/criteria, beliefs, attitudes, and preferences." (Nicola, Ferreira, et al., 2012).

The authors suggest that the value definition is related to its context, so the supplier must set a strategy to enable the customer to realise the product's benefits. A business strategy must contemplate multiple factors, particularly market segment, and identify the differentiating factors from the competitors.

Costumer value relates the inherent costs from the product acquisition with his benefits. Therefore, the solution presented by this project will increase the collaboration between development, operations, and security teams to dynamically rectify the security issues inherent in Web Application solutions, enhancing the software quality, compliance, and avoiding risk.

Between all the solutions offered, the perceived value for the customer might not match its intrinsic value. Factors like the current need and market changes influence the perception of a specific product at a particular moment. For example, if a recent attack or zero-day vulnerability has been detected, security concerns will be at the top priorities for the time being.

1.1.3 Pains and Gains

A study conducted in 2018 concluded that a DevSecOps enhanced the following gains in software development (451 Research, 2018):

- Software quality and security will reduce the required rework and fixes.
- Compliance and regulatory requirements.
- Risk avoidance by security testing enables software deployments to be made quickly and frequently.

The same study registered some setbacks in the DevSecOps approach. The following pains were identified:

- Lack of automated tools
- No standard approach
- Lack of integrated security tools
- Noise of false positives and inconsistency
- Security Testing slows down the process
- Developer Resistance
- Compliance

1.1.4 Value Proposition

"… a value proposition it's an affirmation that attempts to demonstrate the connection between consumer necessities and the benefits that advent from using our product." (Author translation from João R Andrade, 2019)

The following value proposition aims to enhance software development by improving the quality of the overall solution. Security controls are integrated to make the solution compliant. Finally, the answer will allow early defect detection and consequent fixes, increasing the client's response capacity.

1.1.5 Value proposition – Canvas

A canvas business model (Figure 21) is a conceptual tool that relates elements to express the business logic. Focussed on the customer needs to identify the required products and services. In conjunction with the previously identified pains and gains are essential to address the job-to-be-done.

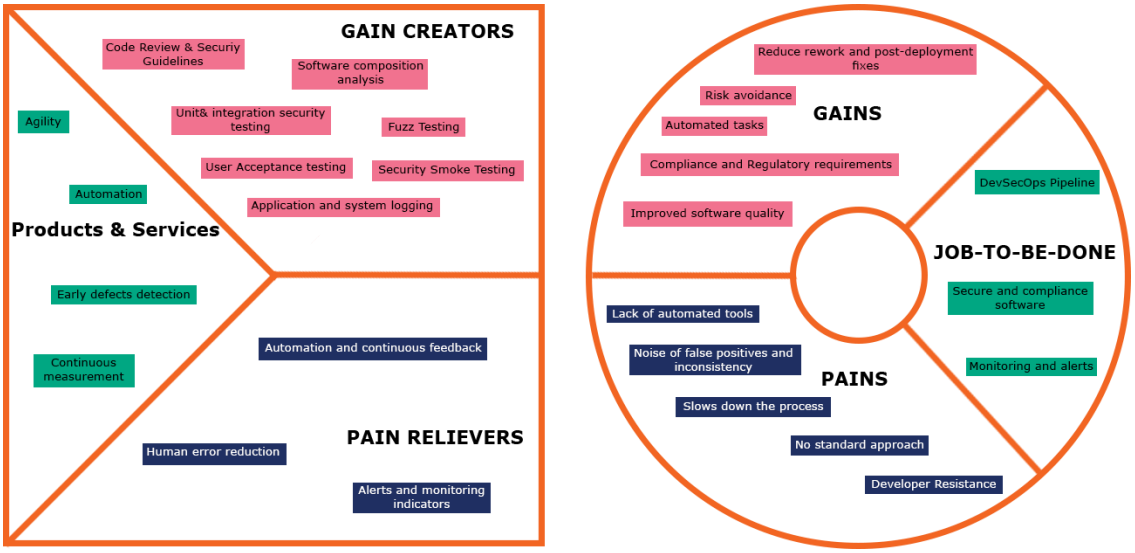


Figure 21 – Canvas Business Model
(Patrick Van Der PIJL, 2012)

1.1.6 Idea Generation

Following the critical thinking from the NCD model, a set of ideas were gathered to solve the problem subject of this work:

- **DevSecOps implementation:** Product evaluation starts early and more often. Each or most increments are tested to discover all potential security defects before any release.

- **Security tests before deployment:** Testing the solution is ensured before the deployment, and manual acceptance is required for production release.

- **Security tests at production release:** Costumers and users receive the final product before security has been tested.

1.1.7 Analytic Hierarchy Process (AHP)

With the focus on solving the identified problem, choosing between the previous ideas was taken regarding the Analytic Hierarchy Process (AHP). Thomas L. Saaty developed this tool to solve multi-criteria decision problems, particularly when subject to the opinions of various individuals (Saaty, Thomas L, 2008).

The tool indicates that to decide the following steps should be considered:

1. Define the problem and determine what kind of knowledge is needed.

2. Develop a graphical representation of the problem's objectives, criteria, and alternatives.

3. Identify the importance of each criterion in terms of its contribution to achieving the result and create an array where different hierarchical levels arrange the elements about the criteria they represent.

4. Alternatives are prioritised through a mathematical process that analyses their importance and preference.

The following factors were established as fundamental criteria for technological decision:

- Solution to discover all or the majority of potential security defects
- Automation
- Agility
- Fast delivery

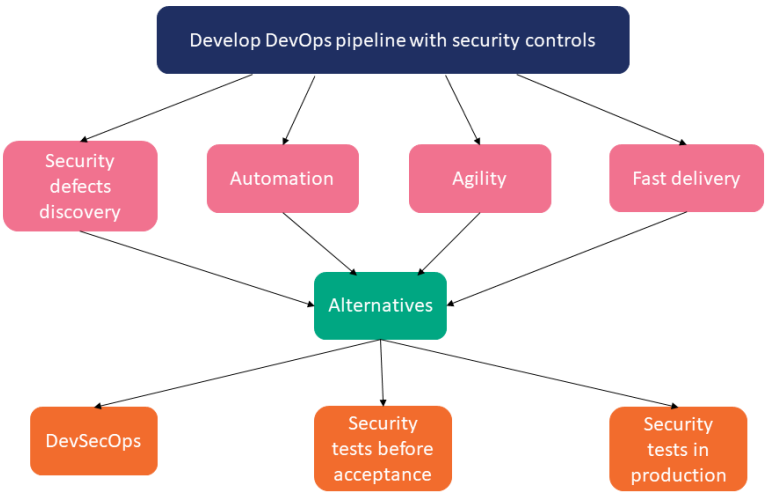


Figure 22 – AHP Diagram

The importance or dominance of a given criterion is given by the fundamental scale of absolute numbers (Figure 23).

Intensity of importance on an absolute scale	Definition	Explanation
1	Equal importance	Two activities contribute equally to the objective
3	Moderate importance of one over another	Experience and judgment strongly favor one activity over another
5	Essential or strong importance	Experience and judgment strongly favor one activity over another
7	Very strong importance	An activity is strongly favored and its dominance demonstrated in practice
9	Extreme importance	The evidence favoring one activity over another is of the highest possible order of affirmation
2,4,6,8	Intermediate values between the two adjacent judgments	When compromise is needed

Figure 23 – AHP fundamental scale of absolute numbers

(Saaty, Thomas L, 2008)

Given the scale defined in Figure 23, the criteria for the solution were classified according to the following table (Table 5). These values reflect the knowledge obtained in the elaboration of state of the art (3) combined with the expected difficulties in developing the proposed tasks.

Table 5 - AHP Criteria Pairwise Comparison

Criteria	Sec. defects discovery	Automation	Agility	Fast delivery
Sec. defects discovery	1	3	2	4
Automation	0,33	1	3	2
Agility	0,33	0,33	1	2
Fast delivery	0,25	1	0,33	1
Total	**1,91**	**5,33**	**6,33**	**9**

Criteria Comparison Normalization

From Table 5, the following matrix is constructed to be normalised. Normalisation is obtained by dividing each entry by the sum of the values belonging to the same column, with the following formula:

$A' = [a'_{ij}] = \frac{a_{ij}}{\sum_{k=1}^{n} aik}$ for $1 \leq i \leq n$ and $1 \leq j \leq n$

Table 6 – AHP criteria normalised matrix - level two

Criteria	Sec. defects discovery	Automation	Agility	Fast delivery
Sec. defects discovery	0,52	0,56	0,32	0,44
Automation	0,17	0,19	0,47	0,22
Agility	0,17	0,06	0,16	0,22
Fast delivery	0,13	0,19	0,05	0,11

With the criteria normalised matrix, the relative priority of each criterion is calculated with the arithmetic average of the values obtained in the matrix Table 7.

Table 7 – AHP criteria normalised matrix – Relative Priority

Criteria	Sec. defects discovery	Automation	Agility	Fast delivery	***Relative Priority***
Sec. defects discovery	0,52	0,56	0,32	0,44	0,46
Automation	0,17	0,19	0,47	0,22	0,26
Agility	0,17	0,06	0,16	0,22	0,15
Fast delivery	0,13	0,19	0,05	0,11	0,12

The correspondent percentage was calculated accordantly in Table 8.

Table 8 – AHP Criteria Relative Priorities

Criteria	Result (%)	Priority
Sec. defects discovery	46 %	1st
Automation	26 %	2nd
Agility	15 %	3rd
Fast delivery	12 %	4th

The consistency of the calculated relative priorities is determined with the Consistency Ratio (CR). The initial criteria pairwise comparison matrix is multiplied by the relative priority (Priorities vector).

$$\begin{bmatrix} 1 & 3 & 2 & 4 \\ 0{,}33 & 1 & 3 & 2 \\ 0{,}33 & 0{,}33 & 1 & 2 \\ 0{,}25 & 1 & 0{,}33 & 1 \end{bmatrix} X \begin{bmatrix} 0{,}46 \\ 0{,}26 \\ 0{,}15 \\ 0{,}12 \end{bmatrix} = \begin{bmatrix} 2{,}02 \\ 1{,}10 \\ 0{,}63 \\ 0{,}54 \end{bmatrix}$$

The average results give the average consistency ratio for the consistency ratio calculated previously.

$$\boldsymbol{\lambda max} = \mathbf{Avg}\left(\frac{2,02}{0,46}, \frac{1,10}{0,26}, \frac{0,63}{0,15}, \frac{0,54}{0,12}\right) = 4,33$$

Now, the following formula gives the average of the consistency index:

$$\boldsymbol{CI} = \frac{\lambda max - n}{n-1} = \frac{4,33-4}{4-1} = 0,11$$

The consistency ratio is obtained by dividing the previously calculated consistency index by the value four (0,9) in Figure 24, where the upper row is the order of the random matrix, and the lower is the corresponding index of consistency for random judgements (Saaty, Thomas L, 2008).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

Figure 24 – Consistency Ratio aid table

(Saaty, Thomas L, 2008)

$$\boldsymbol{CR} = \frac{CI}{0,90} = \frac{0,11}{0,90} = 0,12 > 0,1$$

Since the Consistency Ratio (CR) value is higher than 0,1, it is possible to conclude that the relative priorities are not consistent. And so, it is not possible to figure out that the highest priority criteria are the security defects discovery.

Chosen criteria all correlate to the Agile paradigm that proclaims quickness and agility. It is possible that one or more priorities could be accentuated, and despite having the highest priority, Security defects discovery wasn't an incontestable result.

Alternatives Comparison by Criteria

Alt.1 – DevSecOps

Alt.2 – Security tests before production deployment

Alt.3 - Security tests after production deployment

Table 9 - AHP Security defects discovery Alternatives Comparison

Security defects discovery	Alt.1	Alt.2	Alt.3
Alt.1	1	6	9
Alt.2	0.33	1	2
Alt.3	0.11	0.33	1
Total	1,44	7,33	12

Table 10 - AHP Security defects discovery Normalized Matrix

Security defects discovery	Alt.1	Alt.2	Alt.3	***Priority Vector***
Alt.1	0,69	0,82	0,75	0,75
Alt.2	0,22	0,14	0,17	0,18
Alt.3	0.08	0.05	0,08	0,07

The highest priority alternative for the security defects discovery criteria is DevSecOps (Alt.1).

Table 11 - AHP Automation Alternatives Comparison

Automation	Alt.1	Alt.2	Alt.3
Alt.1	1	3	1
Alt.2	0,33	1	2
Alt.3	0.5	3	1
Total	1,83	7	4

Table 12 - AHP Automation Normalized Matrix

Automation	Alt.1	Alt.2	Alt.3	***Priority Vector***
Alt.1	0,55	0,43	0,25	0,41
Alt.2	0,18	0,14	0,50	0,27
Alt.3	0,27	0,43	0,25	0,32

For the automation criteria, the highest priority alternative remains DevSecOps (Alt.1).

Table 13 - AHP Agility Alternatives Comparison

Agility	Alt.1	Alt.2	Alt.3
Alt.1	1	5	1
Alt.2	0,25	1	4
Alt.3	0.5	4	1
Total	1,75	10	6

Table 14 - AHP Agility Normalized Matrix

Agility	Alt.1	Alt.2	Alt.3	***Priority Vector***
Alt.1	0,57	0,5	0,17	0,41
Alt.2	0,14	0,01	0,67	0,27
Alt.3	0,29	0,4	0,17	0,29

For the agility criteria, the highest priority alternative remains DevSecOps (Alt.1).

Table 15 - AHP Fast delivery Alternatives Comparison

Fast delivery	Alt.1	Alt.2	Alt.3
Alt.1	1	3	0,25
Alt.2	6	1	0,5
Alt.3	9	7	1
Total	16	11	1,75

Table 16 - AHP Fast delivery Normalized Matrix

Fast delivery	Alt.1	Alt.2	Alt.3	***Priority Vector***
Alt.1	0,06	0,27	1,4	0,58
Alt.2	0,38	0,09	0,29	0,25
Alt.3	0,56	0,64	0,57	0,59

For the fast delivery criteria, the highest priority alternative is security tests after production deployment (Alt.3). This is justified because the product is delivered after engaging in security tests and therefore available to the clients and users.

Table 17 - Alternatives Priority by Criteria Matrix

	Sec. defects discovery	Automation	Agility	Fast delivery
Relative Priority (Table 8)	0,46	0,26	0,15	0,12
Alt.1	0,75	0,41	0,41	0,58
Alt.2	0,18	0,27	0,27	0,25
Alt.3	0,07	0,32	0,29	0,59

Obtained the priority vectors values for each criterion, they are related to the criteria relative priorities calculated before on Table 8. Each column of a criteria's alternative priority vector is multiplied by the values on relative priorities.

$$\begin{bmatrix} 0,75 & 0,41 & 0,41 & 0,58 \\ 0,18 & 0,27 & 0,27 & 0,25 \\ 0,07 & 0,32 & 0,29 & 0,59 \end{bmatrix} X \begin{bmatrix} 0,46 \\ 0,26 \\ 0,15 \\ 0,12 \end{bmatrix} = \begin{bmatrix} 0.58 \\ 0,22 \\ 0,23 \end{bmatrix}$$

Table 18 - Overall Alternative Priority Results

Criteria	Result (%)	Priority
DevSecOps	58 %	1st
Security tests before production deployment	22 %	3rd
Security tests after production deployment	23 %	2nd

In conclusion, the results obtained in Table 18 distances the DevSecOps idea from the other alternatives when security is a concern. The other two alternatives got a near calculated result. The main difference resides primarily in the nature of the second alternative (Security tests before production deployment), which requires manual inputs through the development delivery cycle.