



Shift-Left Security

Critical TechWorks

Author

Lourenço Santos - 1181586

CTW Supervisor

Joel Martins

ISEP Advisor

Prof. Luís Nogueira - LMN

CTW Technical Supervisors

José Maia

Diogo Mota Almeida

David Barreiro

2021/2022

Acknowledgements

I would like to thank Critical TechWorks for the opportunity and for the welcoming environment they offered me. Particularly I'd like to thank the ITOps team: Joel Martins, José Maia, Diogo Almeida, David Barreiro for their guidance. The members of the ITOps team were my "go-to" guys and were my mentors throughout this whole process.

Huge thanks to the Tetris team as well, with whom I've been working for a part of this internship and who taught me a lot.

Thanks to Doaa Assaf for her suggestions during this internship and availability, even though she's from a completely different team inside CTW (IT Security team), which I also had the pleasure to get to know and interact with, something that shows the great spirit inside this company.

I am grateful as well to Carlos Cunha, who was part of the ITOps team too, unfortunately now he's no longer working at the company, but he's one of the reasons I got this internship.

I would also like to thank professor Luís Nogueira for always providing quick support, guidance, and helpful suggestions.

Once ITOps, ITOps forever.

Summary

DevSecOps, a relatively new term in the application security (AppSec) space, is about introducing security earlier in the software development life cycle (SDLC) by expanding the close collaboration between development and operations teams in the DevOps movement to include security teams as well.

Essentially, DevSecOps means that security is a shared responsibility, and everyone involved in the SDLC has a role to play in building security into the DevOps CI/CD workflow.

The earlier we introduce security into the workflow, the sooner we can identify and remedy security weaknesses and vulnerabilities. This concept is part of “shifting left,” which moves security testing toward developers, enabling them to fix security issues in their code in near real time rather than waiting until the end of the SDLC, where security was bolted on in traditional development environments.

DevSecOps spans the entire SDLC from planning and design to coding, building, testing, and release, with real-time continuous feedback loops and insights.

To implement DevSecOps, organizations should consider a variety of application security testing (AST) tools to integrate into their CI/CD process, will be expanded later in this report. Some commonly used AST tools follow:

- Static Application Security Testing (SAST)
- Software Composition Analysis (SCA)
- Interactive Application Security Testing (IAST)
- Dynamic Application Security Testing (DAST)

Key words (Theme): DevSecOps, Security, Pipeline, AST, Maturity Model

Key words (Technology): Docker, ZAP, DSOMM, Dependency-Check

Table of Contents

1	<i>Introduction</i>	<i>1</i>
1.1	Motivation.....	1
1.2	The Organization.....	1
1.3	The Project	2
2	<i>Context</i>	<i>6</i>
2.1	Problem.....	6
2.2	State of The Art.....	7
2.2.1	DevSecOps Globalization.....	7
2.2.2	Existing Technologies	10
2.2.3	SAST.....	15
2.2.4	DAST	21
2.2.5	Aquasec Trivy	22
2.2.6	Threat Modeling.....	23
3	<i>Work Environment</i>	<i>27</i>
3.1	Planning.....	27
3.1.1	Scrum or Waterfall?	29
3.1.2	User Stories	30
3.2	Communication	30
3.3	Tools.....	31
4	<i>Analysis and Design.....</i>	<i>33</i>
4.1	Dependency-Check Integration	33
4.2	Dependency-Track Integration	34
4.3	Aquasec Trivy Integration	35
4.4	OWASP ZAP Integration	36
4.5	Full Pipeline Integration	1
4.6	Threat Model by Threat Dragon	1
5	<i>Solution Implementation</i>	<i>3</i>
5.1	Injecting Security into CI/CD Pipelines	3

5.1.1	OWASP Dependency-Check	3
5.1.2	OWASP Dependency-Track	7
5.1.3	Aquasec Trivy	10
5.1.4	OWASP ZAP	15
5.2	Threat Model	18
5.2.1	Installation.....	18
5.2.2	Threat Model for the Internal Tools	19
6	Conclusions	22
6.1	The Process.....	22
6.2	Results.....	21
6.3	Suggestions.....	Erro! Marcador não definido.
	References	25

Table of Figures

Figure 1- DSOMM Implementation Levels.....	3
Figure 2- DSOMM Structure	4
Figure 3- Workshop "Nice to have" from DSOMM.....	4
Figure 4- Container vs VM	12
Figure 5- Aquasec Trivy	22
Figure 6- Batman's Threat Model	23
Figure 7- Threat Dragon diagram.....	26
Figure 8- The Scrum cycle	28
Figure 9- Reasons against Waterfall	29
Figure 10- Jira active sprint	30
Figure 11- Dependency-Check Integration	33
Figure 12- Dependency-Track Integration	34
Figure 13- Aquasec Trivy Integration.....	35
Figure 14- OWASP ZAP Integration.....	36
Figure 15- Full Pipeline Integration.....	1
Figure 16- Threat Model for Internal Tools	1
Figure 17- Dependency-Check Report	4
Figure 18- Dependency-Check Maven plugin	5
Figure 19- Supression of a false positive	5
Figure 20- Dependency Check on JenkinsFile	6
Figure 21- Build fail due to log4j vulnerability	6
Figure 22- CycloneDX Pom Configuration.....	8

Figure 23- Dependency-Track Pom Configuration.....	8
Figure 24- Build package and image to use with trivy	10
Figure 25- .trivignore file.....	11
Figure 26- Trivy stage on JenkinsFile	11
Figure 27- Ignored CVE by Trivy	11
Figure 28- Warning in Jenkins	12
Figure 29- Aquasec Trivy Output	13
Figure 30- Pipeline fail due to vulnerability	14
Figure 31- CVE-2021-44228 caught by Trivy.....	14
Figure 32- ZAP config file	16
Figure 33- ZAP output	17
Figure 34- Internal Tools Threat Model	19
Figure 35- Part of Threat Dragon Report	20
Figure 36- Folder provided with all the research.....	23
Figure 37- DSOMM Points Comparison	21

Table of Tables

Table 1 - Chosen technologies	10
Table 2- Dependency-Check vs Dependency-Track (OWASP, 2021)	21

Notation and Glossary

AWS	Amazon Web Services
BOM	Bill-Of-Material
CI/CD	CI/CD combines practices of continuous integration and either continuous delivery or continuous deployment. CI/CD bridges the gaps between development and operation activities and teams by enforcing automation in building, testing and deployment of applications (CI/CD, n.d.).
Container	OS-level virtualization that shares the services of a single operating system kernel, using fewer resources than virtual machines (Docker (software), n.d.).
Containerization	The action of setting up a container for deployment.
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DAST	Dynamic Application Security Testing
ESOFT	Software Engineering subject from the 2 nd semester of Computer Engineering in ISEP
Git	Version Control System
IAST	Interactive Application Security Testing
ISEP	Instituto Superior de Engenharia do Porto
ITOps	DevOps team that guided me through this whole internship
OWASP	Open Web Application Security Project® (OWASP) is a nonprofit foundation that works to improve the security of software

PESTI	ISEP subject in which the internship is considered
RDBMS	Relational Database Management System - database management system used by relational databases such as Oracle databases, MySQL and PostgreSQL.
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SDLC	Software Development Lifecycle
SSDLC	Secure SDLC
Tetris	Team I'm working with, and which is responsible for the development of the internal tools for the company
US/ User Story	Describes the type of user, what they want and why
Unit	CTW Corporate is divided into numerous Units. These Units are divided in teams, and each team works on its own projects.
VM	Virtual Machine
ZAP	Zed Attack Proxy



1 Introduction

This document reports the development of the internship project executed in the context of the PESTI subject at Critical TechWorks.

The first chapter describes the project, the student's motivation, the organization, as well as the report structure.

1.1 Motivation

As part of the Informatics Engineering curriculum at ISEP, students are challenged to study and develop a solution for an informatics engineering problem. This represents a great opportunity to expand and improve the skillset acquired throughout the course.

With this internship, both soft and technical skills could be further developed. This project also presented the opportunity to work with high-end emerging and growing technologies, which was tempting.

This project was a welcomed first contact with local tech companies, providing a jumpstart into the job market.

1.2 The Organization

Established in 2018, Critical TechWorks is a company formed as joint venture between BMW Group and Critical Software to lead the future of motion. Critical TechWorks was put together exclusively to support BMW in building software for its future driving machines.

The BMW Group provides its technology-centric challenges, loyalty, and industry expertise on how to make hi-tech and ultra-reliable cars, while Critical

Software brings in its culture, values, software development talent and proven agile methodologies.

Autonomous driving, car info-entertainment systems on the car, and electrification are just some of the technologies in development by joint venture, on top of the increasing efficacy of production on the group factories.

The BMW Group owns 51% of the Critical TechWorks partnership and Critical Software the remaining 49% of the company. (Critical Techworks, 2021)

1.3 The Project

The implementation of a DevSecOps culture is becoming more useful and necessary nowadays. When I was first interviewed for this internship, the team I'm currently working with, was looking for someone that could help them improve security in their projects, integrating it in the DevOps process to facilitate the life of the developers, and work side by side with the operations team.

This team is working on the internal tools of the company, so my first tasks were to get to know the technologies they are using and study the state of art of DevSecOps so I could start from zero and know what I was doing.

While I was studying the state of art of DevSecOps I came across a term called "Maturity Models".

A maturity model is a tool that helps people assess the current effectiveness of a person or group and supports figuring out what capabilities they need to acquire next to improve their performance.

Here's a simple example of a Maturity Model. We might define levels like this:

- Level 1.** Knows how to make a dozen basic drinks (eg "make me a Gin and Tonic")
- Level 2.** Knows at least 100 recipes, can substitute ingredients (eg "make me a Vieux Carre in a bar that lacks Peychaud's")

Level 3. Able to come up with cocktails (either invented or recalled) with a few simple constraints on ingredients and styles (eg "make me something with sherry and tequila that's moderately sweet"). (Fowler, 2014)

Then I found a Maturity Model for DevSecOps implementation called the DSOMM (DevSecOps Maturity Model), illustrated in figure 1 and 2.

The DevSecOps Maturity Model shows security measures which are applied when using DevOps strategies and how these can be prioritized.

With the help of DevOps strategies security can also be enhanced. For example, each component such as application libraries and operating system libraries in docker images can be tested for known vulnerabilities.

Attackers are intelligent and creative, equipped with new technologies and purpose. Under the guidance of the DevSecOps Maturity Model, appropriate principles and measures are at hand implemented which counteract the attacks. (OWASP, 2021)

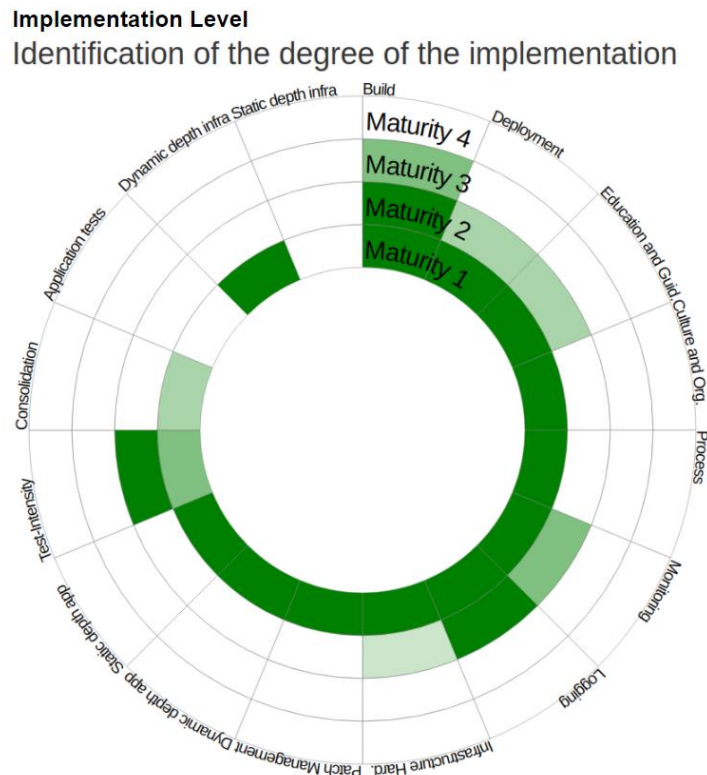


Figure 1- DSOMM Implementation Levels (OWASP, 2021)

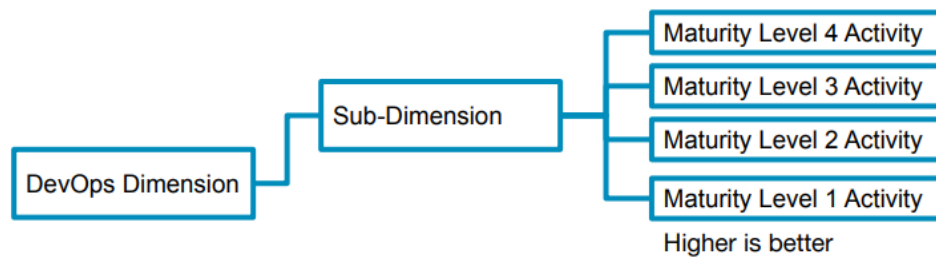


Figure 2- DSOMM Structure

After some research about this matrix, I decided to schedule a workshop about the first 2 levels of the DSOMM with the unit I'm working with, so we could all decide the measures to implement.

Sub-Dimension	Process	Have	Nice to have	Don't have	Usefulness	Time
	A patch policy is defined		YES		High	Very Low
	Automated PRs for patches		YES		Very High	Low
Design	Conduction of simple threat modeling		YES		Medium	Medium
	Ad-Hoc Security trainings for software		YES		Medium	Very Low
Guidance	Security code review		YES		Medium	Low
	Definition of simple BCDR practices for		YES		High	Medium
App Hardening	Application Hardening Level 1		YES		High	High
	Definition of quality gates		YES		High	Very Low
Consolidation	Simple false positive treatment		YES		High	Very Low
Dynamic depth for	Simple Scan		YES		Low	Medium
Static depth for	Test of server side components with known vuln		YES		Very High	Low
Static depth for	Stored Secrets		YES		Low	Very Low
	Usage of trusted images		YES		Medium	Very Low
Patch Management	Reduction of the attack surface		YES		Medium	Medium
	Usage of a max lifetime for images		YES		Medium	High
	Each team has a security champion		YES		High	Low
	Regular security training for all		YES		High	High
Education and	Regular security training of security		YES		Very High	Low
	Reward of good communication		YES		Medium	Low
	Simple mob hacking		YES		Medium	Medium
App Hardening	Application Hardening Level 2		YES		High	High
	Checking the sources of used libraries		YES		Medium	Medium
Consolidation	Simple visualization of defects		YES		Medium	Low
	Coverage of client side dynamic		YES		High	Medium
	Usage of different roles		YES		Low	Medium
Static depth for	Static analysis for important server		YES		High	Low
	Check for image lifetime		YES		Low	Very Low
Static depth for	Test of virtualized environments		YES		Medium	Very Low
	Test the definition of virtualized		YES		Medium	Very Low
Application Tests	Security unit tests for important		YES		Medium	High

Figure 3- Workshop "Nice to have" from DSOMM

In the figure 3, that represents a table that I made on Microsoft Excel I filtered the points that we, as a team, considered important to have in our unit to have better security standards.

With all this information, my team and I were able to create some User Stories related to the implementation of some of these practices, for example:

- SPIKE: Test semgrep for a few tetris frontends
- SPIKE: Test find-sec-bugs in a few Tetris backends
- SPIKE: Test OWASP SKF
- SPIKE: Test OWASP Dependency-Check

(Spike is an article that we publish on our platform to share our knowledge related to specific topics)

These are just an example of some US that were created, others weren't Spikes, but since I had no experience with these tools, I had to do a lot of research and then try to implement them on our projects before integrating these tools in the pipeline.

As we saw on the Microsoft Excel sheet, there were many things to implement, so there were no reasons for me to evaluate the 3rd and 4th levels.

Summing all the points we had implemented, considering level 1 – 1 point and level 2 – 2 points we got a total of 38 points out of 103 points, where we considered 86 would be nice to have, which will be taken into consideration when I finish my internship to see how many more points, we were able to get since I've started working here.

2 Context

In this chapter, the project is contextualized with the problem that the company had and what was the solution implemented to overcome this problem.

2.1 Problem

When I first arrived at the team I'm working with, there were no security concerns around this development process. They adopted a DevOps culture, which consists in a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams. It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.

The DevOps movement began around 2007 when the software development and IT operations communities raised concerns about the traditional software development model, where developers who wrote code worked apart from operations who deployed and supported the code.

A DevOps team includes developers and IT operations working collaboratively throughout the product lifecycle to increase the speed and quality of software deployment.

DevOps teams use tools to automate and accelerate processes, which helps to increase reliability. A DevOps toolchain helps teams tackle important DevOps fundamentals including continuous integration, continuous delivery, automation, and collaboration. (DevOps, 2021)

DevOps values are sometimes applied to teams other than development. When security teams adopt a DevOps approach, security is an active and integrated part of the development process. This is called DevSecOps.

As I mentioned before, DevSecOps consists, basically, in integrating the security to this continuous development process, and when I first got at the company, I had to start this whole process from scratch. That's when I found the DSOMM to guide me through this phase.

2.2 State of The Art

The state of the art provides an overview of the solution and compares it with similar ones, highlighting why this project is unique and relevant.

2.2.1 DevSecOps Globalization

The relationship between development and security teams is often contentious. Security might see developers as a liability when it comes to protecting data and systems, and developers often view security as a disruption to their workflow.

Both parties are right if the organization in which they work fails to create an environment of collaboration and shared goals between development and security. Without that kind of culture, the two groups will inevitably be at odds with one another. (Nadeau, 2019)

The article cited portrays real histories from the following companies—Microsoft, Verizon, and the Pokemon Company—these 3 examples have very different business models and security needs. However, they all benefited from taking a DevSecOps approach to their internal development process.

The success stories include:

- A **Verizon** developer dashboard providing vulnerability visibility
- **Pokemon Company** embraces security by design to protect children's privacy
- Sharing information, best practices bring development and security together at **Microsoft**

These are just small examples from big companies who have been benefitted from a DevSecOps approach.

Nowadays, most “big shark” companies have adopted this methodology, so it’s only logical for us to follow their example.

2.2.2 Why is security so important?

Over the past years, there have been an increase of attacks and data breaches all over the world. These attacks and data breaches can lead a company to bankruptcy, besides many more risks associated with it.

Nowadays, security tools are so expensive, and engineers specialized in this field are so well paid since everyone is becoming aware to the relevance of security.

For example, in a successful online store, a DOS attack can cost thousands or millions of euros to the company, a data breach of a company can expose millions of users or employees and risk the end of that same company.

Between 2020 and 2021, the personal data of **700 million** LinkedIn users, nearly 93% of the company’s members, was on sale online. Though the data did not include login credentials or financial information, it included a lot of personal information, such as:

- Full names
- Phone numbers
- Physical addresses
- Email addresses
- Geolocation records

In January 2021, the largest personal data breach in Brazilian history was discovered (223 million records). The data sets were discovered by PSafe and then reported by Tecnoblog. The databases included names, unique tax identifiers, facial images, addresses, phone numbers, email, credit score, salary and more. The data also

contains the personal data of several million deceased individuals. In addition, 104 million vehicle records were available. (Henriquez, 2021)

Security researcher Alon Gal discovered a leaked database belonging to Facebook, containing 533 million accounts.

According to Gal, “A database of that size containing the private information such as phone numbers of a lot of Facebook's users would certainly lead to bad actors taking advantage of the data to perform social-engineering attacks or hacking attempts.” (Holmes, 2021)

2.2.3 Advantages of a DevSecOps approach

First, it's an improvement of DevOps, where we integrate security practices and tools before the deployment of the code. Then, we have a faster security without the risks, developers wanted to create code as quickly as possible, despite the number of checks required before the results could go live. This way, DevSecOps Engineers can check the code for possible vulnerabilities during the development process.

Another advantage is that the company is able to lower the prices. Automation is hardly a new concept, and anyone familiar with it knows that it offers three primary benefits: speed, reliability, and cost reduction. DevSecOps automation enables organizations to satisfy security targets with less of a human factor (that is to say, less manpower). (Gallagher, 2021)

Another advantage is more reliable security practices. Automation is applied to processes designed to find and highlight security vulnerabilities in code. The clarity offered by DevOps and DevSecOps pipelines also makes it clear who is best suited to solve specific issues. With this setup, issues are more likely to be found and repaired earlier on, so much so that more problems can be solved within the timeframe before release, resulting in higher-quality end products.

2.2.4 Existing Technologies

This section briefly describes the technologies used to develop the solution. The table below summarizes the technologies chosen.

Table 1 - Chosen technologies

Database type	PostgreSQL
SCA Tool	OWASP Dependency-Check
Container Scanner	Aquasec Trivy
Containerization and deployment	Docker
	Kubernetes
Programming Language	Java, TypeScript, JavaScript
Programming Frameworks	Angular, React.js, Quarkus, Spring
Operating System	Ubuntu 20.04 (not essential, used only for testing)
Cloud Virtual Machine Host	Azure
	AWS

2.2.4.1 Database

At Critical Techworks, the Tetris team uses PostgreSQL.

2.2.4.1.1 PostgreSQL

PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. (PostgreSQL, n.d.)

Unlike other RDMBS, it supports both non-relational and relational data types, making it one of the most compliant, stable, and mature relational databases available.

Benefits from using this database:

- Performance and scalability
- Concurrency support
- Deep language support
- Business continuity
- 100% open source

2.2.4.2 Deployment

2.2.4.2.1 Containers – Docker

Just as shipping industries use containers to isolate different cargos – for example, to transport in ships and trains – software development technologies increasingly use an approach called containerization.

A standard package of software – known as a container – bundles an application's code together with the related configuration files and libraries, and with the dependencies required for the app to run. This allows developers and IT pros to deploy applications seamlessly across environments. (Microsoft Azure, 2021)

What's the difference between containers and VMs?

VMs were born, designed by running software on top of physical servers to emulate a particular hardware system.

Each VM runs a guest OS. VMs can run different Oss on the same server. For example, a Unix VM can sit alongside a Linux VM, and so on. Each VM has its own libraries, binaries, and apps that it services, and the VM may be many Gb in size.

On the other side, containers sit on top of a physical server and its host OS (Windows or Linux for example). Each container shares the host OS kernel and, usually, the binaries and libraries as well. Shared components are read-only, they are light in size in comparison to the Gb that a VM uses.

Figure 4 represents in a visual way the difference between both this containers.

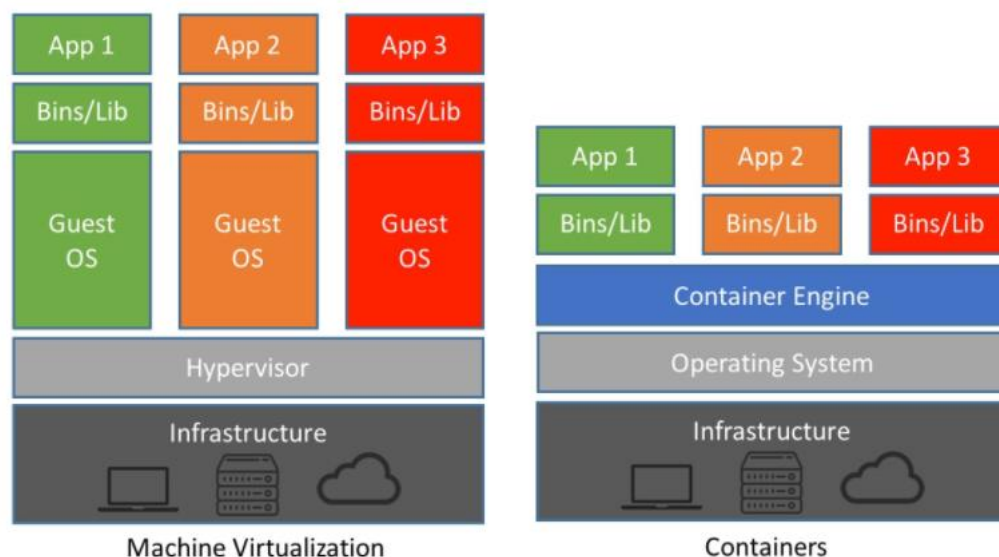


Figure 4- Container vs VM

Docker is an OS containerization platform. Devs can create containers without docker, but this platform makes it easier and safer.

Docker is essentially a toolkit that enables developers to build, deploy, run, update, and stop containers using simple commands and work-saving automation through a single API. (IBM, n.d.)

Docker tools and terms:

- DockerFile
- Docker images
- Docker containers
- Docker Hub
- Docker Daemon
- Docker Registry

2.2.4.2.2 Cluster - Kubernetes

Kubernetes is a portable, extensible, and open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. (Kubernetes, 2021)

Kubernetes provides us with a framework to run distributed systems resiliently, such as:

- **Service discovery and load balancing** - Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes can load balance and distribute the network traffic so that the deployment is stable.
- **Storage orchestration** - Allows us to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- **Automated rollouts and rollbacks** - We can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, we can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
- **Automatic bin packing** - We provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.

- **Self-healing** - Kubernetes restarts containers that fail, replaces containers, kills containers that do not respond to your user-defined health check, and does not advertise containers to clients until they are ready to serve.
- **Secret and configuration management** - Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

2.2.4.2.3 Cloud Provider - Azure & AWS

For host we use Azure and AWS.

Azure is a cloud computing platform with solutions such as: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) that can be used for services such as analytics, virtual computing, storage, networking, ... (Microsoft Azure, 2021)

AWS, just like Azure is a cloud computing platform and offers services for app development and analytics, such as Amazon EC2:

- Server configuration and hosting;
- Amazon S3: Data storage and movement
- AWS 101: Elastic Load Balancing for scalable performance
- AWS Overview: Using CloudFront to deliver a better user experience
- Elastic Block Store (EBS): Low-latency instance access
- Amazon Route 53: The AWS DNS service
- Cloudwatch: Monitor your AWS environment

2.2.4.2.4 System - Ubuntu

Ubuntu is a complete Linux OS. In my case I used Ubuntu 20.04 because it was easier for me to perform tests or commands, since I'm more used to work with Linux Terminal rather than Windows PowerShell or Windows CMD.

2.2.5 SAST

Source Code Analysis (SCA) tools, also known as Static Application Security Testing (SAST) Tools, can help analyze source code or compiled versions of code to help find security flaws. (OWASP, 2021)

SAST tools can be added into your IDE. Such tools can help you detect issues during software development. SAST tool feedback can save time and effort, especially when compared to finding vulnerabilities later in the development cycle.

Examples of SAST tools:

- Klocwork
- SpectralOps
- Checkmarx
- Veracode
- Dependency-Check

Strengths:

- Scales well – can be run on lots of software and can be run repeatedly (as with nightly builds or continuous integration).
- Identifies certain well-known vulnerabilities, such as:
 - Buffer overflows
 - SQL injection flaws
- Output helps developers, as SAST tools highlight the problematic code, by filename, location, line number, and even the affected code snippet.

Weaknesses:

- Difficult to automate searches for many types of security vulnerabilities, including:
 - Authentication problems
 - Access control issues
 - Insecure use of cryptography

- Current SAST tools are limited. They can automatically identify only a relatively small percentage of application security flaws.
- High numbers of false positives.
- Frequently unable to find configuration issues since they are not represented in the code.
- Difficult to ‘prove’ that an identified security issue is an actual vulnerability.
- Many SAST tools have difficulty analyzing code that can’t be compiled.
 - Analysts frequently cannot compile code unless they have:
 - Correct libraries
 - Compilation instructions
 - All required code

2.2.5.1 OWASP Dependency-Check

2.2.5.1.1 Overview

OWASP Dependency-Check is a SCA tool. It attempts to detect publicly disclosed vulnerabilities contained within a project’s dependencies. This tool does this by determining if there is a Common Platform Enumeration (CPE) identifier for a given dependency. If found, it will generate a report linking to the associated CVE entries. (OWASP, 2021)

Dependency-Check can be used to scan applications (and their dependent libraries) to identify any known vulnerable components. It has a command line interface, a Maven plugin, which I’ll talk about later when integrated on the pipeline, Ant task and a Jenkins plugin.

Later we’ll be able to see its implementation on the pipeline in more detail and check the results, but this tool downloads all CVE entries from the NVD database and in case there is any dependency with a vulnerability the build will fail according to the CVSS level that we provide on that is serious enough to cause a failure.

2.2.5.2 OWASP Dependency-Track

2.2.5.2.1 Overview

Dependency-Track is an intelligent **Component Analysis** platform that allows organizations to identify and reduce risk in the software supply chain. Dependency-Track takes a unique and highly beneficial approach by leveraging the capabilities of SBOM. This approach provides capabilities that traditional Software Composition Analysis (SCA) solutions cannot achieve. (OWASP, 2021)

2.2.5.2.2 SBOM

A “Software Bill of Materials” (SBOM) is effectively a nested inventory, a list of ingredients that make up software components. The following documents were drafted by stakeholders in an open and transparent process to address transparency around software components and were approved by a consensus of participating stakeholders. (NTIA, 2021)

There are 4 level of details for SBOMs:

- Licenses
- Modules
- Patch Levels
- Backports

Most of the discussion about SBOMs is roughly at the license level.

But who needs a SBOM? Any organization that builds software needs to maintain a software BOM for their codebases. Organizations typically use a mix of custom-built code, commercial off-the-shelf code, and open-source components to create software.

2.2.5.3 Dependency-Check vs Dependency-Track

	Dependency-Track	Dependency-Check
Software Type	Platform	Library with multiple implementations: <ul style="list-style-type: none"> • Command line interface • Build plugins (Maven, Ant, etc) • Jenkins plugin
Approach	Software Bill-of-Materials (SBOM) which can be automatically generated at build-time or obtained from vendors	Scans files on filesystem and extracts evidence with varying degrees of confidence
Vulnerability Intelligence	<ul style="list-style-type: none"> • Precise matching via NVD • Sonatype OSS Index • NPM Audit API • VulnDB 	<ul style="list-style-type: none"> • Fuzzy matching via NVD • Sonatype OSS Index • NPM Audit API • Retire.js
Outdated Version Identification	<ul style="list-style-type: none"> • Composer 	None

	<ul style="list-style-type: none"> • Hex • RubyGems • Maven • NPM • NuGet • PyPi 	
Ecosystems supported	Ecosystem agnostic (all ecosystems supported)	10+ with varying degrees of maturity
Reporting	Dynamic intelligence and metrics delivered via REST API or web interface	Per-project statically generated HTML, XML, JSON, and CSV reports
License Support	Resolves over 500 SPDX license IDs as well as supporting unresolved license names	Unresolved license names as evidence
Jenkins Plugin	Yes (bidirectional)	Yes (unidirectional)
Sonarqube Plugin	No	Yes

Vulnerability aggregation	<ul style="list-style-type: none"> • Defect Dojo (vendor supported) • Kenna Security (natively supported) • Fortify SSC (natively supported) • Security Compass (vendor supported) • ThreadFix (vendor supported) 	<ul style="list-style-type: none"> • CodeDx (vendor supported) • Defect Dojo (vendor supported) • Nucleus Security (vendor supported) • Orchestron (vendor supported) • Security Compass (vendor supported) • ThreadFix (vendor supported) • ZeroNorth (vendor supported)
Notification Support	<ul style="list-style-type: none"> • Slack • Microsoft Teams • Webhooks • Email 	None
Auditing	Per-project and global auditing workflow supporting analysis decisions, comments, and	Suppression file with support for CPE, filename, and regex pattern matching

	suppressions that are captured and tracked in a per-finding audit log	
Private Vulnerability Repository	Yes	No
Perspectives	<ul style="list-style-type: none"> • Portfolio of projects (applications, services, devices, etc) • Project • Dependency • Component • Vulnerability • License 	<ul style="list-style-type: none"> • Project • Dependency • Vulnerability

Table 2- Dependency-Check vs Dependency-Track (OWASP, 2021)

2.2.6 DAST

Dynamic Application Security Testing (DAST) is a procedure that actively investigates running applications with penetration tests to detect possible security vulnerabilities. (Rapid7, 2022)

DAST tools provide insight into how your web applications behave while they are in production, enabling your business to address potential vulnerabilities before a hacker uses them to stage an attack.

As your web applications evolve, DAST solutions continue to scan them so that your business can promptly identify and remediate emerging issues before they develop into serious risks.

DAST Tools continually search for vulnerabilities in a web app that is in production.

Upon identifying a vulnerability, a DAST solution sends automated alerts to the appropriate teams so they can prioritize and remediate it.

Businesses can also use DAST to assist with PCI compliance and other types of regulatory reporting.

2.2.7 Aquasec Trivy

Trivy (tri pronounced like **trigger**, vy pronounced like **envy**) is a simple and comprehensive scanner for vulnerabilities in container images, file systems, and Git repositories, as well as for configuration issues. Trivy detects vulnerabilities of OS packages (Alpine, RHEL, CentOS, etc.) and language-specific packages (Bundler, Composer, npm, yarn, etc.). In addition, Trivy scans Infrastructure as Code (IaC) files such as Terraform, Dockerfile and Kubernetes, to detect potential configuration issues that expose your deployments to the risk of attack. (Aquasec, 2021)

Figure 5 represents the “skeleton” of this tool.

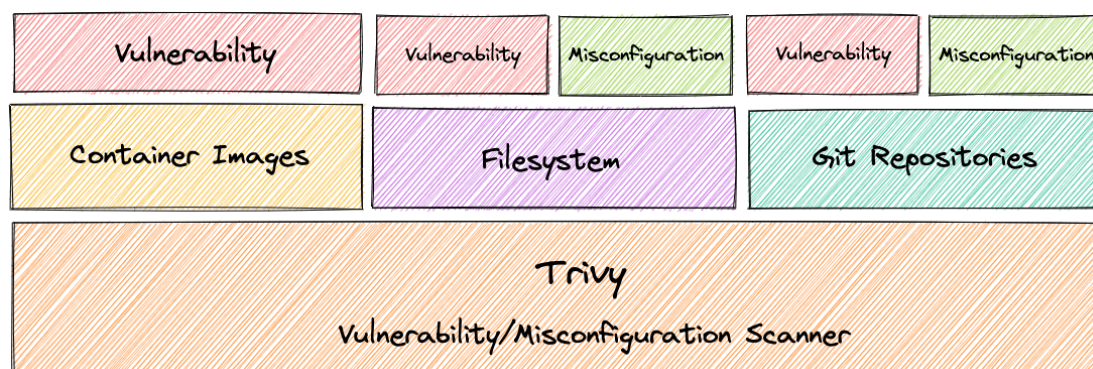


Figure 5- Aquasec Trivy

Trivy can easily be integrated to common CI/CD tools like Jenkins, GitLab CI, Circle CI, Travis CI, and more.

2.2.8 Threat Modeling

Threat modelling is a key element to integrate security into software systems. It enables us to identify critical areas of design which needs to be protected. Over the time various threat modelling approaches and methodologies have been developed and are being used in the process of designing secure web applications. This approach varies from conceptual frameworks to practical methodologies. The following paragraphs present a brief survey of existing threat modelling methodologies and techniques. (Shafiq Hussain, 2014)

As a simple example of a threat model, in figure 6, it's represented a diagram that I've shown on a presentation about threat modelling to the team I'm working with:

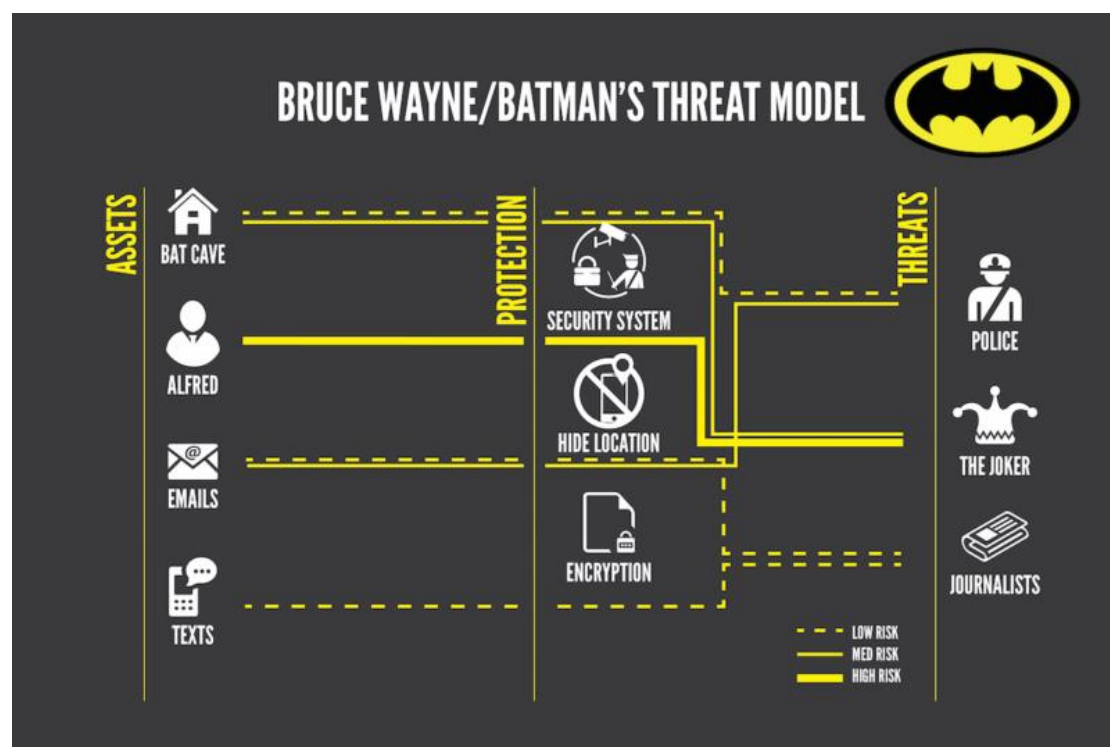


Figure 6- Batman's Threat Model

This is a funny and a simple way to understand how a threat model works. Here we can see that the Bat Cave has a low risk of being found by the police and a mid-risk of being found by the joker. Alfred represents a high risk for batman if the joker finds him, even though his location is unknown, ...

2.2.8.1 STRIDE

It's a lightweight, practical, and user-friendly technique proposed by Microsoft.

STRIDE stands for:

- **Spoofing (S):** impersonating as a legitimate user
- **Tampering (T):** modifying legitimate information
- **Repudiation (R):** denying a particular event or action performed in the system
- **Information disclosure (I):** unauthorized exposure of confidential information
- **Denial of Service (D):** making the system unavailable for legitimate user
- **Elevation of Privilege (E):** getting higher privilege than granted for a particular user

STRIDE analyzes weakness against each component of the system that can possibly be exploited for threats, thereby compromising the whole system. (Sirajuddin Ahmed, 2019)

The steps are:

1. Decompose the system into components
2. Sketch the data-flow diagram for each component of the system
3. Identify threats for each element in the data-flow diagram using the STRIDE mnemonics
4. Suggest possible countermeasures to the threat

There are other methodologies to conduct a threat model, such as CIA (Confidentiality, Integrity, Availability) or LINDDUN (Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of Information, Unawareness, Non-Compliance).

There are many tools that can help us develop this threat model, such as:

- Microsoft Threat Modeling Tool
- OWASP Threat Dragon
- Cairis
- IriusRisk
- Kenna
- Threagile

In my case I used the OWASP Threat Dragon.

2.2.8.2 OWASP Threat Dragon

Threat Dragon is a free, open-source, cross-platform threat modelling application including system diagramming and a threat rule engine to auto-generate threats/mitigations. It is an OWASP Lab Project and follows the values and principles of the threat modeling manifesto. The roadmap for the project is a simple UX, a powerful rule engine and integration with other development lifecycle tools.

Threat Dragon supports STRIDE / LINDDUN / CIA, provides modeling diagrams and implements a rule engine to auto-generate threats and their mitigations. (OWASP, 2021).

In figure 7 we can see an example of a Threat Model developed in this tool:

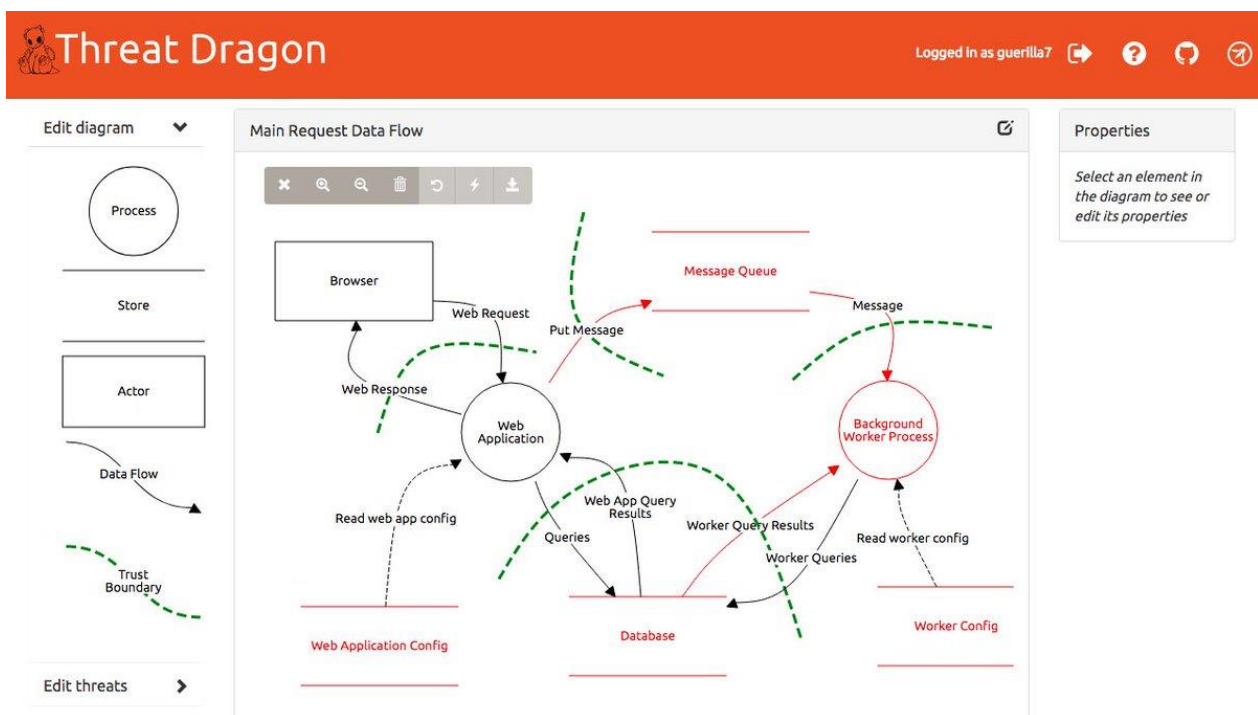


Figure 7- Threat Dragon diagram

3 Work Environment

This chapter is going to describe the work environment in which I spent these last months.

3.1 Planning

Here at CTW we use the Scrum framework from Agile methodology

Scrum is a framework that helps people, teams and organizations generate value.

Briefly, to implement this framework we'll need:

- Scrum Master: We call them Scrum Knights, which is many times confused with a leader of the team, but he is simply the person that ensures the scrum framework is followed. When the team needs him, he's most of the times the "go-to" guy and he's a "coach" that ensures the team is self-organized.
- Product Owner: We call them Product Visionaries, which is who makes the bridge between the stakeholders and the development team. This PO orders the work for the product wanted by the stakeholders into a Product Backlog.
- Scrum Team: This team must be independent from operations and security team to meet the deadlines and turns a selection of the work into an increment of value during a sprint.

The following picture illustrates the scrum framework with its many events.

Each sprint starts with a Sprint Planning where all the members from a team gather (SM, PO, Scrum Team) to discuss which USs should be added to the Sprint Backlog (each sprint has normally a duration of around 2/3 weeks) from the Product Backlog to deliver value.

During the sprint, the whole team gathers at a defined time, usually in the morning, and each team member says what they have done the day before, what they are planning on doing that day and if they have some impediments that could be blocking them from continuing with their tasks.

The increment corresponds to the value added to the product at the ending of each sprint.

In the end of the sprint, a Sprint Review and a Sprint Retrospective are usually done. The Sprint Review is a meeting with the stakeholders where the team shows the value added to the product on that sprint and the stakeholder can give its feedback, which many times doesn't match what the team was expecting, generating more USs to the Product Backlog.

Finally, the team gathers again to the Sprint Retrospective where they decide what went well and what went wrong on that same sprint.

Figure 8 represents the whole cycle of Scrum events.

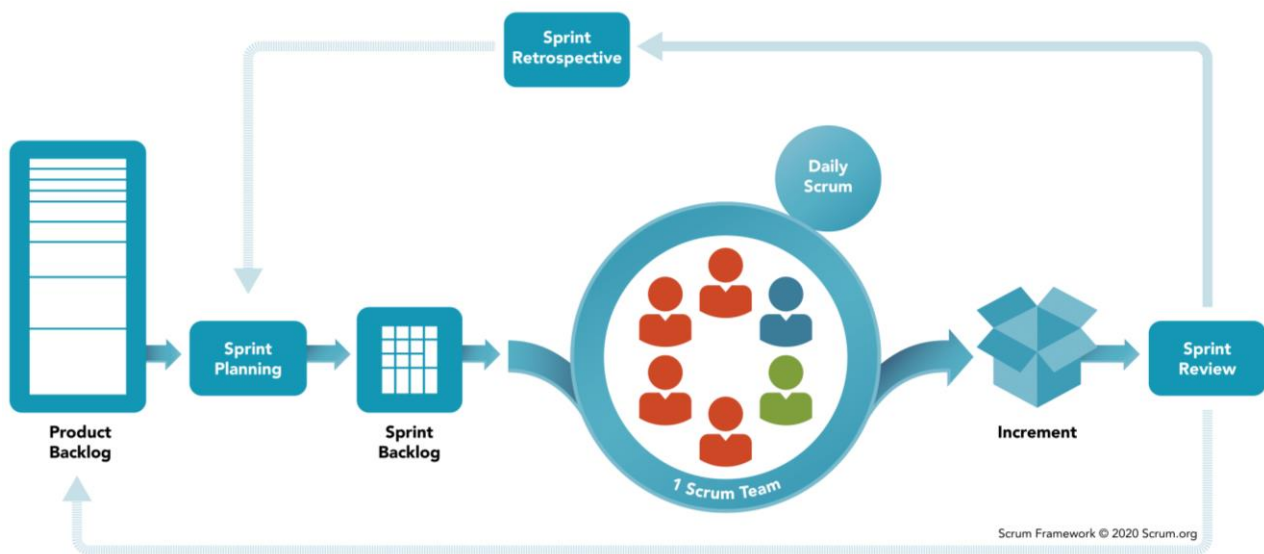


Figure 8- The Scrum cycle

3.1.1 Scrum or Waterfall?

Another common methodology is Waterfall, there's Kanban as well, which is similar to Scrum with the main difference being the use of continuous flow instead of sprints where the tasks are continuously added to their "Sprint" backlog as the previous USs are finished, but let's focus on Waterfall.

Waterfall is a sequential development process that flows like a waterfall through all phases of a project (analysis, design, development, testing, deployment, and maintenance for example), with each phase.

The main advantages from using this methodology are the chance for developers to catch design errors during analysis and design stages and this way they can avoid them in the implementation, the total cost of the project can be accurately estimated, it's easier to measure progress according to clearly defined milestones, customers don't keep adding new requirements to the project, delaying production.

The main disadvantages are the deadline creep, that means that if one phase in the process is delayed, all other phases get delayed as well, clients do not get involved in design and implementation stages, so the client can ask for a car and we deliver them a motorbike. Client's many times don't get satisfied with the frontend, and in an Agile methodology the client's going to be able to explain exactly what he wants.

Figure 9 illustrates this scenario, that many of us must've already seen it, as I record seeing in one theoretical class of ESOF.

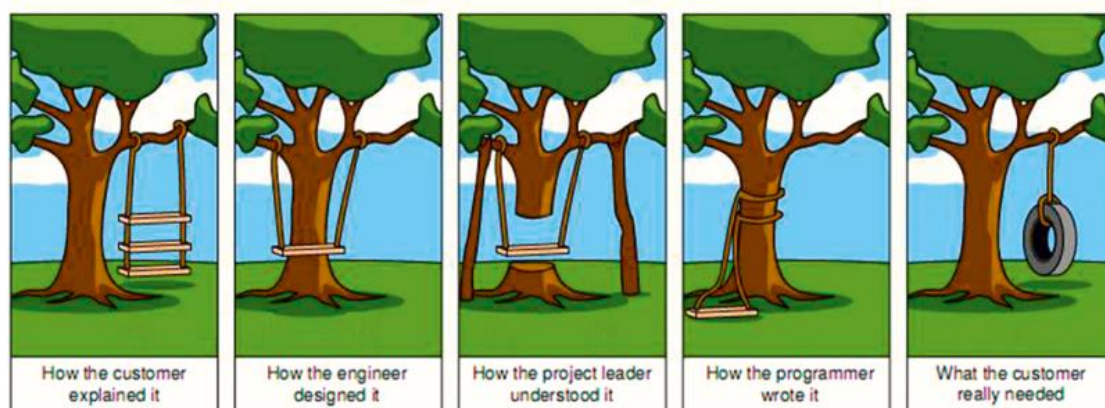


Figure 9- Reasons against Waterfall

3.1.2 User Stories

Jira was used to keep track of the User Stories by our POs. On figure 10, we can see what a board looks like in our team:

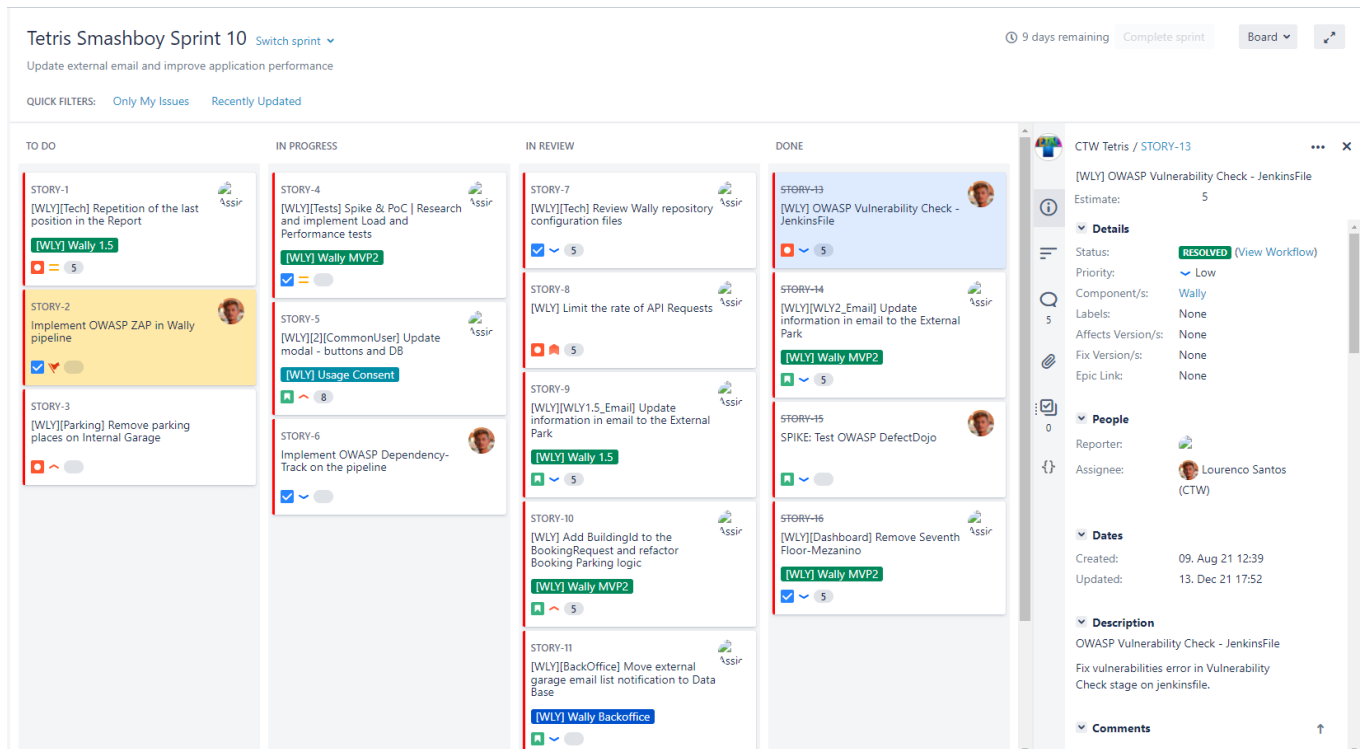


Figure 10- Jira active sprint

3.2 Communication

At CTW we all communicate via Microsoft Teams, even though there are other means to do that, this is our preferred tool to interact with each other. Beside Teams we also use:

- Outlook – Used for announcements to all staff, schedule meetings and every time we have some technical problem, we have an email of the company which we often use to get help.
- Slack – Used inside our team for non-work-related talk.

- Yammer – This is a kind of social media only for the employees of CTW where we can share our ideas.
- Confluence – Used to publish spikes/articles to share knowledge inside BMW Group.

3.3 Tools

Beside the tools we use to communicate there are a lot of other tools we are required to install when we first arrive at CTW, such as:

- Docker - Providers of the virtual machines used for testing.
- IntelliJ – Mostly used to work on the backend of some apps.
- Visual Studio Code – Mostly used to work on Frontend, Jenkins Files, Docker Files, ...
- Ubuntu/WSL – For those who work with Windows, so we have a Linux shell.
- Oh-my-zsh – Not mandatory, but most of us installed it as a shell theme.
- VirtualBox – For a Virtual Environment inside our PC.
- Postman – To perform HTTP requests
- Bitbucket – Git Code Management

In my case, there was a few other tools I installed, and which I'll talk about later, such as:

- Threat Dragon – To design a threat model.
- OWASP ZAP – To perform passive/active scans to our tools and find vulnerabilities on the frontend of our applications.
- Clair - Open-source tool that lets you scan containers and docker images for potential security problems
- Semgrep - Fast, open-source, static analysis tool for finding bugs and enforcing code standards at editor, commit, and CI time.

3.3.1 Before and After

When I first entered at this team, I was given a “starter kit guide” to know what I had to install, what were the common practices of the team, regarding Scrum events and Communication inside and outside the team.

Now, the newcomers, beside all that stuff that was already given before I entered this team, are given the threat model of the tools they will be working in, so that they become aware of the risks and threats to the applications they are developing. They are forced to comply with the security standards stablished on the pipeline, this way if they implement something with known vulnerabilities, they'll be alerted by the tools that were implemented on the pipeline, such as Dependency-Check, ZAP and Trivy and won't be able to finish their releases if the security standards are not followed or lowered, which is not the objective.

Last, but not least, they have on the team documentation all the documents I've been gathering by the past couple of months about how to maintain the security tools implemented and what could be the next possible approaches for the future.

4 Analysis and Design

4.1 Dependency-Check Integration

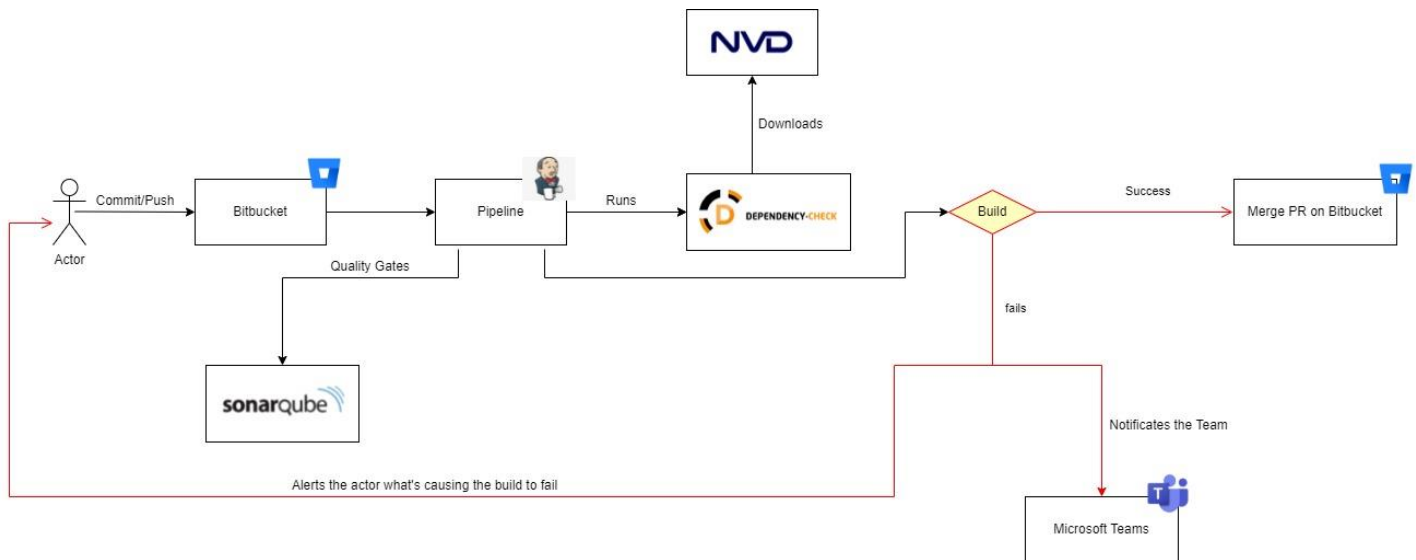


Figure 11- Dependency-Check Integration

When a user commits his code to Bitbucket and tries to build it with Jenkins, Jenkins does a Quality Check performed by SonarQube and downloads the NVD database to check if there's any vulnerability on the project related to outdated dependencies or unpatched plugins.

Then, the build fails if there's any found vulnerability above the specified CVSS on the project's pom and the user is notified by Jenkins logs which CVE is causing problems.

If the build passes the user is now able to make a PR and eventually merge their work if the team approves it.

4.2 Dependency-Track Integration

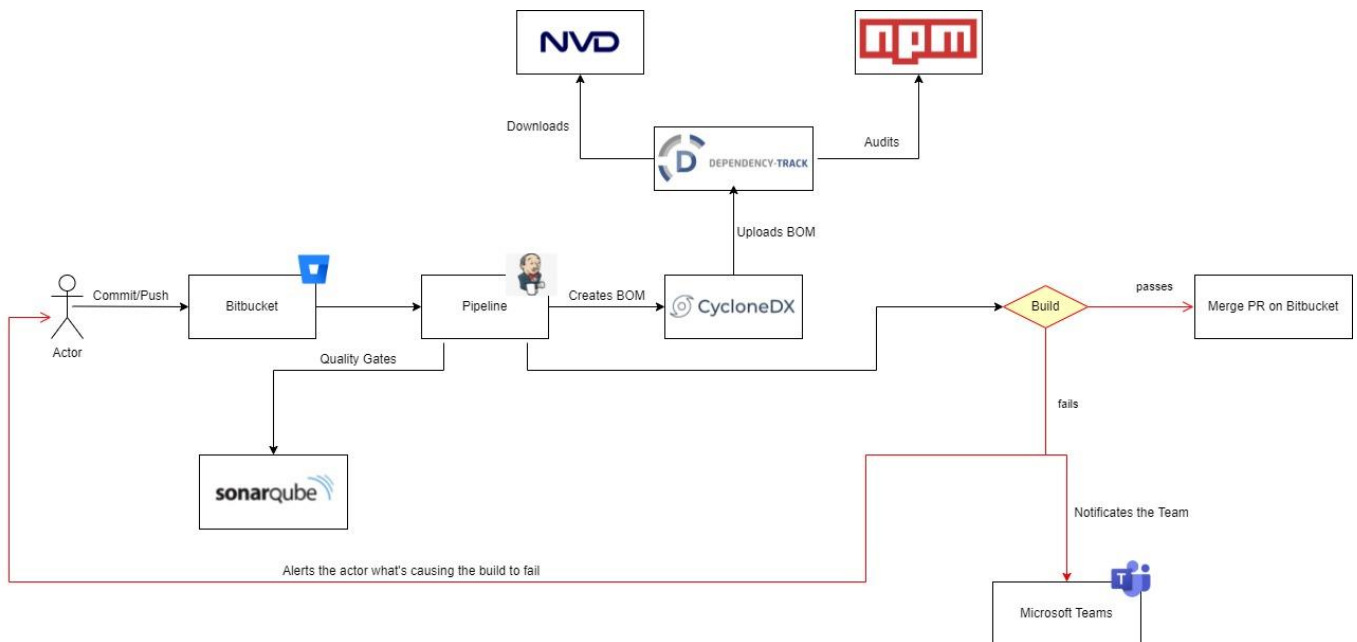


Figure 12- Dependency-Track Integration

When a user commits their code to Bitbucket and tries to build it with Jenkins, Jenkins does a Quality Check performed by SonarQube, then it creates a Bill-of-Material (BOM) with CycloneDX to upload to Dependency-Track, this tool downloads the NVD database to check if there's any vulnerability on the project related to outdated dependencies or unpatched plugins and performs an audit with NPM.

Then, the build fails if there are any vulnerability rated as high or critical, or, for example, if there are 5 vulnerabilities rated as medium risk or 10 vulnerabilities rated as low risk, that can be defined on pom as well, and sends an alert notification via Microsoft Teams to the team.

If the build passes the user gets the message of "Build Success" and they're now able to make a PR or merge it to the source code in Bitbucket if the team approves their work.

4.3 Aquasec Trivy Integration

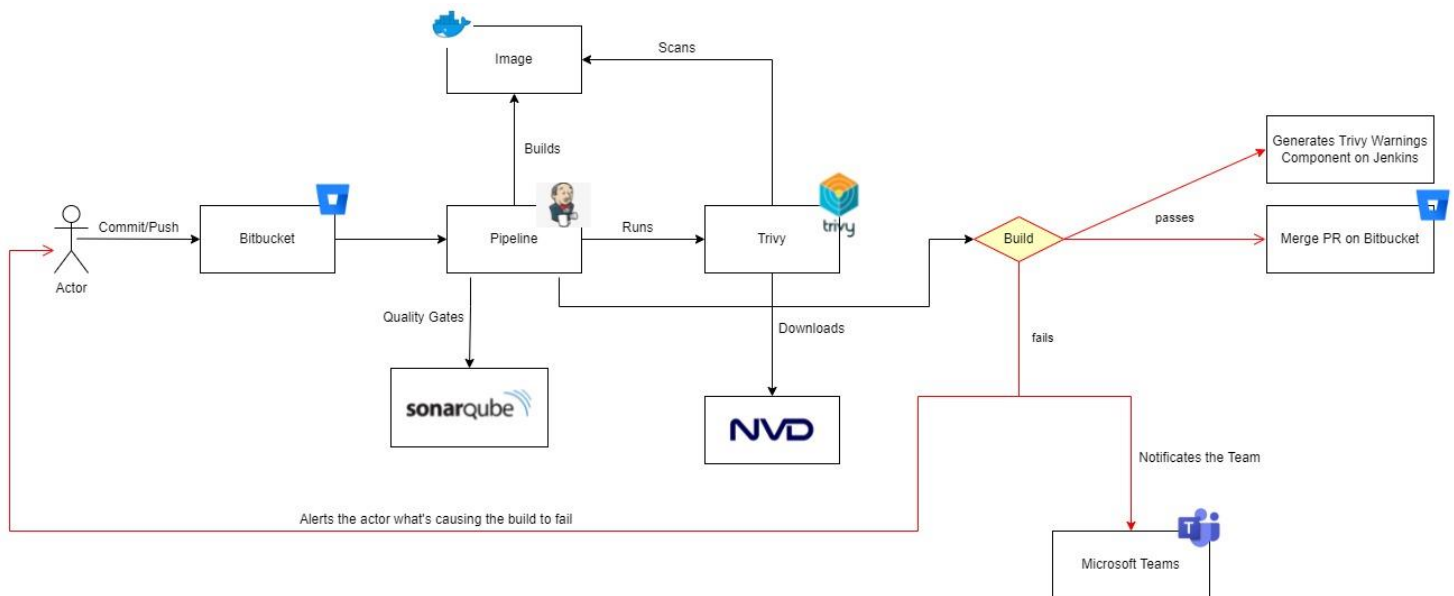


Figure 13- Aquasec Trivy Integration

The user commits their code to Bitbucket and runs the pipeline on Jenkins.

Jenkins, beside everything else that's running, builds the image with a docker build command.

Trivy, then, scans that image to search for known vulnerabilities in the container and downloads a database to compare if there's any CVE to report.

If Trivy and everything else in the pipeline runs successfully it generates a component on Jenkins to see the full report of Trivy generated by a JSON file.

Finally, the user can merge their PR after the team's approval.

4.4 OWASP ZAP Integration

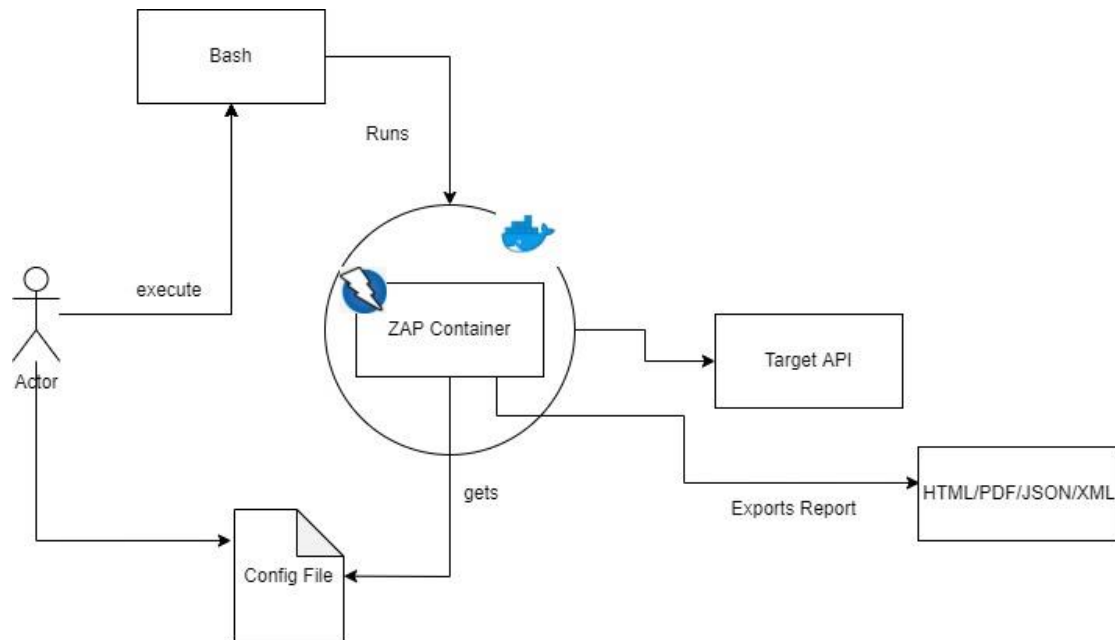


Figure 14- OWASP ZAP Integration

Regarding the OWASP ZAP integration, I couldn't automate this process, because I discussed with one of my technical managers the options that we had to integrate this in the pipeline and there were two problems.

The first one was that the pipeline is already taking around 15 minutes to build, and, including this would cause the pipeline to take 3-5 minutes longer.

The other problem with this tool was that, to perform an authenticated scan, we had two options:

1. Disable authentication in the dev environment
2. Create an API Gateway so that I could pass, in the configuration file, an API Key to bypass the authentication

It was decided that, right now, the best option was to implement a manual process. As we can see we just run the ZAP Container inside a Docker volume and scan the target API, which will retrieve the information as output in our machine.

4.5 Full Pipeline Integration

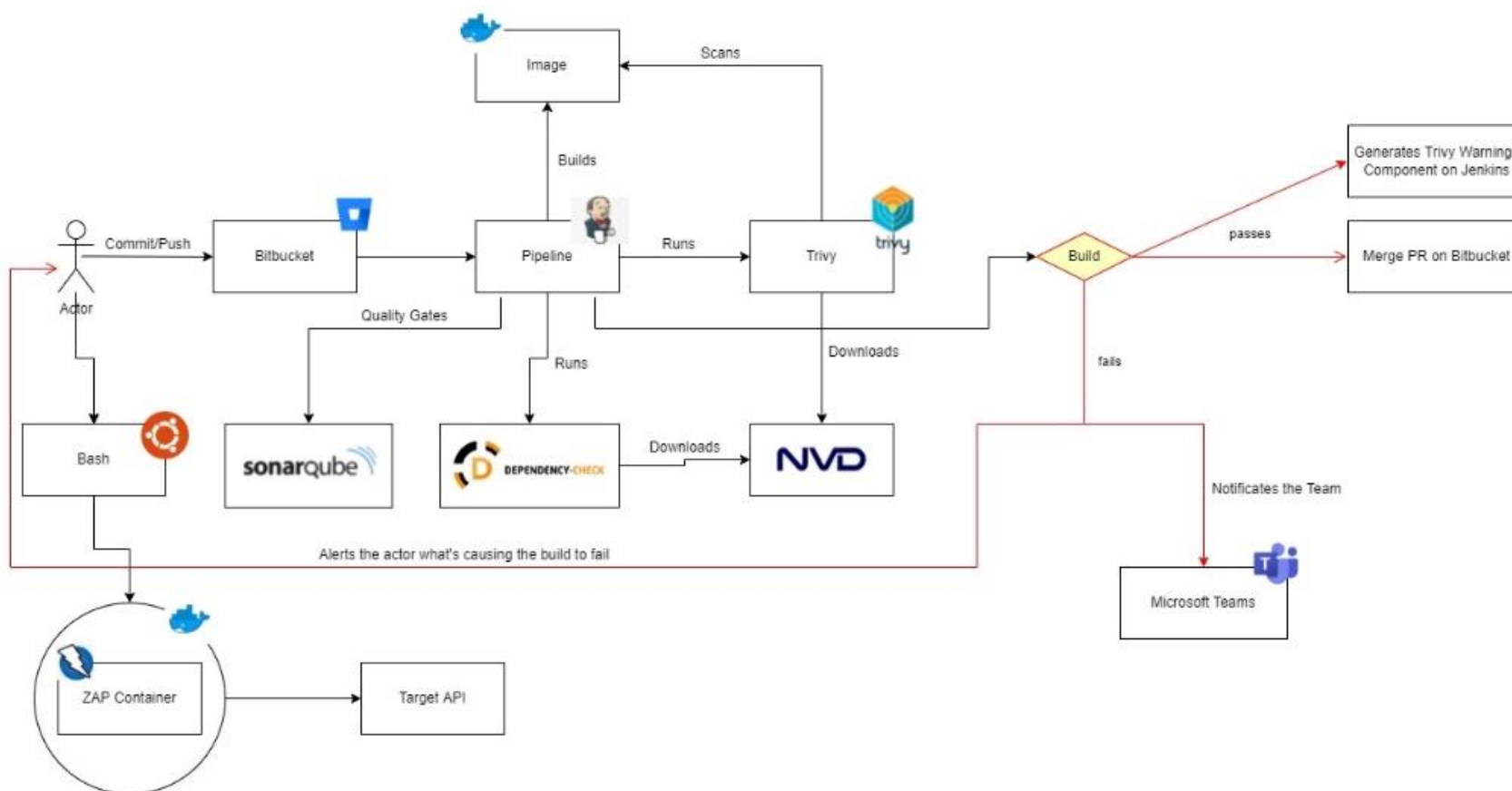


Figure 15- Full Pipeline Integration

Last, but not least, we have the data flows, which are those arrows pointing from one thing to another. If they are dashed, means that they are out of scope, otherwise they are in scope and represented in red, because they have threats associated, such as MITM (Man-in-the-middle) or Vulnerable Transfer Protocols.

5 Solution Implementation

This chapter describes the implementation of the solution presented. It covers details about usage and deployment, details about the implementation, the development process and problems, testing, and the evaluation of the solution.

5.1 Injecting Security into CI/CD Pipelines

Continuous delivery pipelines are implementations of the continuous everything paradigm and help validate every commit our teams make. Integrate automated security checks with the pipeline to give you early warnings and monitor escaped security vulnerabilities relentlessly. (Atlassian, 2022)

Integrated continuous security approaches scale as your business expands. Both unit tests and static code analysis operate closest to source code and run checks without executing the code. Remember, the cost of a defect is low in test, medium in staging, and high in production. So, invest in security unit tests and static analyzers, since these are inexpensive and fast, and can save trouble further down the pipeline.

5.1.1 OWASP Dependency-Check

5.1.1.1 Installation and HTML Report

The installation is simple, you just have to download the latest file of this tool, when I downloaded it, the version was 6.5.0, then extract the zipped file and run the following command:

```
sudo ln -s $(pwd)/dependency-check-6.5.0-release/dependency-check/bin/dependency-check.sh /usr/bin/dependency-check.sh
```

Finally, just check if you don't have a VPN blocking the communication between your machine and `nvd.nist.gov`, which, accidentally, happened in my case, and to get an html vulnerability report just run:

```
dependency-check.sh --scan project_folder
```

And you'll get a report like figure 17:

Project:

Scan Information ([show all](#)):

- *dependency-check* version: 6.5.0
- *Report Generated On*: Thu, 11 Nov 2021 17:16:11 GMT
- *Dependencies Scanned*: 13 (13 unique)
- *Vulnerable Dependencies*: 2
- *Vulnerabilities Found*: 2
- *Vulnerabilities Suppressed*: 0
- ...

Summary

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Cour
jaeger-tracer-lib-jar-with-dependencies.jar (shaded: com.squareup.okhttp3:okhttp:3.9.0)	cpe:2.3:a:squareup:okhttp:3.9.0:*:*:*:*:* cpe:2.3:a:squareup:okhttp:3.9.0:*:*:*:*:	pkg:maven/com.squareup.okhttp3/okhttp@3.9.0	MEDIUM	1	Highest	9
jaeger-tracer-lib-jar-with-dependencies.jar (shaded: org.apache.httpcomponents:httpclient:4.4.1)	cpe:2.3:a:apache:httpclient:4.4.1:*:*:*:*:	pkg:maven/org.apache.httpcomponents/httpclient@4.4.1	MEDIUM	1	Highest	11

Dependencies

jaeger-tracer-lib-jar-with-dependencies.jar (shaded: com.squareup.okhttp3:okhttp:3.9.0)

Figure 17- Dependency-Check Report

5.1.1.2 Integrating Dependency-Check on the Pipeline

To integrate this tool on the pipeline we'll have to add the maven plugin of the Dependency-Check to pom.

Pom.xml should look like figure 18:

```
<plugin>
  <groupId>org.owasp</groupId>
  <artifactId>dependency-check-maven</artifactId>
  <version>${dependency-check-maven-plugin.version}</version>
  <executions>
    <execution>
      <goals>
        <goal>check</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <failBuildOnCVSS>8</failBuildOnCVSS>
    <skipProvidedScope>false</skipProvidedScope>
    <skipRuntimeScope>false</skipRuntimeScope>
    <skip>${vulnerabilitycheck.skip}</skip>
    <suppressionFile>dependency-check-suppression.xml</suppressionFile>
    <format>XML</format>
  </configuration>
</plugin>
```

Figure 18- Dependency-Check Maven plugin

The `<failBuildOnCVSS>` defined at 8 means that any vulnerability rated as 8/10 or higher will cause the build to fail.

Another point that we should look at is the `<suppressionFile>` because as I mentioned before these kinds of tools have a lot of false positives and this way we create a file that will suppress these false positive vulnerabilities.

Figure 19 represents a shortage from `dependency-check-suppression.xml`:

```
<suppress>
  <notes>
    <![CDATA[
      file name: quarkus-security-*.jar
    ]]>
  </notes>
  <gav regex="true">^io\.quarkus\.security:quarkus-security.*.*$</gav>
  <cve>CVE-2021-26291</cve>
  <cve>CVE-2020-1714</cve>
</suppress>
```

Figure 19- Supression of a false positive

Then we'll have to add the following code to a Jenkins File, because on the program we're integrating this tool we make the builds on Jenkins.

```
stage('Dependency Check') {
    failFast true
    parallel {
        stage('OWASP Vulnerability Check') {
            steps {
                sh script: 'mvn verify -Dcheckstyle.skip -DskipTests', label: 'OWASP Vulnerability Check'
            }
        }

        stage('Dependency Updates Check') {
            steps {
                sh script: 'mvn versions:display-dependency-updates', label: 'Dependency updates check'
            }
        }

        stage('Plugin Updates Check') {
            steps {
                sh script: 'mvn versions:display-plugin-updates', label: 'Plugin updates check'
            }
        }

        stage('Property Updates Check') {
            steps {
                sh script: 'mvn versions:display-property-updates', label: 'Property updates check'
            }
        }
    }
}
```

Figure 20- Dependency Check on JenkinsFile

As a proof that this integration was well implemented, on December 9th of 2021 there was a vulnerability with Log4j that was known globally for its danger.

CVE-2021-44228, which is the dictionary entry of this CVE consisted of an attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. (NVD, 2021)

We had an internal tool that used this dependency, and when we built the pipeline, it failed because of this vulnerability.

In this case I've set the failBuild at 10, as we can observe on figure 21, so that it doesn't disclose other vulnerabilities that we already know and are taking measures to fix them.

```
[ERROR] Failed to execute goal org.owasp:dependency-check-maven:6.5.0:check (default) on project car-pool:
[ERROR]
[ERROR] One or more dependencies were identified with vulnerabilities that have a CVSS score greater than or equal to '10.0':
[ERROR]
[ERROR] log4j-api-2.13.3.jar: CVE-2021-44228
[ERROR]
[ERROR] See the dependency-check report for more details.
[ERROR]
```

Figure 21- Build fail due to log4j vulnerability

5.1.2 OWASP Dependency-Track

5.1.2.1 Installation

3. Run these 2 commands:

```
curl -LO https://dependencytrack.org/docker-compose.yml  
docker-compose up -d
```

4. Open docker and start the image
5. Go to localhost:8080
6. Sign in with credentials "admin:admin"
7. Change password
8. Login with new password

5.1.2.2 Integrating Dependency-Track on a Maven Project

First, if you notice on figure 22, you'll need to add the CycloneDX plugin to pom.xml, because Dependency-Track needs a BOM to be uploaded to analyze the project:

```

154 | <plugin>
155 |   <groupId>org.cyclonedx</groupId>
156 |   <artifactId>cyclonedx-maven-plugin</artifactId>
157 |   <version>2.5.3</version>
158 |   <executions>
159 |     <execution>
160 |       <phase>package</phase>
161 |       <goals>
162 |         <goal>makeAggregateBom</goal>
163 |       </goals>
164 |     </execution>
165 |   </executions>
166 |   <configuration>
167 |     <projectType>library</projectType>
168 |     <schemaVersion>1.3</schemaVersion>
169 |     <includeBomSerialNumber>true</includeBomSerialNumber>
170 |     <includeCompileScope>true</includeCompileScope>
171 |     <includeProvidedScope>true</includeProvidedScope>
172 |     <includeRuntimeScope>true</includeRuntimeScope>
173 |     <includeSystemScope>true</includeSystemScope>
174 |     <includeTestScope>false</includeTestScope>
175 |     <includeLicenseText>false</includeLicenseText>
176 |     <outputFormat>all</outputFormat>
177 |     <outputName>bom</outputName>
178 |   </configuration>
179 | </plugin>

```

Figure 22- CycloneDX Pom Configuration

Then, let's add the Dependency-Track plugin to pom.xml, indicated in figure

23:

```

180 | <plugin>
181 |   <groupId>io.github.pmckown</groupId>
182 |   <artifactId>dependency-track-maven-plugin</artifactId>
183 |   <version>1.1.0</version>
184 |   <configuration>
185 |     <dependencyTrackBaseUrl>http://localhost:8081</dependencyTrackBaseUrl>
186 |     <apiKey>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</apiKey>
187 |     <pollingConfig>
188 |       <enabled>true</enabled>
189 |       <pause>500</pause>
190 |       <attempts>40</attempts>
191 |       <timeUnit>MILLIS</timeUnit>
192 |     </pollingConfig>
193 |     <findingThresholds>
194 |       <critical>0</critical>
195 |       <high>0</high>
196 |       <medium>5</medium>
197 |       <low>10</low>
198 |     </findingThresholds>
199 |   </configuration>
200 | </plugin>

```

Figure 23- Dependency-Track Pom Configuration

After the pom is well configured with the API Key, that you can find on your front-end localhost by Administration > Access Management > Teams > Administrators, you'll be able to upload the BOM to your localhost by the following cmd:

```
mvn dependency-track:upload-bom -Ddependency-  
track.projectName=arbitrary-name -Ddependency-  
track.projectVersion=99.99
```

You can print the findings by using this following cmd:

```
mvn dependency-track:findings -Ddependency-  
track.projectName=arbitrary-name -Ddependency-  
track.projectVersion=99.99
```

You can print the risk score which is the sum of all CVSS by using this cmd:

```
mvn dependency-track:score
```

You can get the metrics by using this cmd:

```
mvn dependency-track:metrics
```

Last, but not least, you can delete the project by running this cmd:

```
mvn dependency-track:delete-project
```

5.1.3 Aquasec Trivy

5.1.3.1 Installation

The process of installation is as simple as running these 5 commands on your Ubuntu machine:

- `sudo apt-get install wget apt-transport-https gnupg lsb-release`
- `wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -`
- `echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list`
- `sudo apt-get update`
- `sudo apt-get install trivy`

5.1.3.2 Jenkins Integration

To integrate Trivy in Jenkins pipeline I'd to install the binary file and then execute trivy.

Other relevant topic is that, to use the recordIssues, I had to install a plugin in Jenkins's pipeline called "Next Generation Warnings".

Following the order of the pipeline, first we must build a package and then the image to proceed to the image scan by Trivy, as it's notable on Figure 24:

```
stage('Build Package') {  
    steps {  
        sh script: 'mvn -B -DskipTests -Dcheckstyle.skip -Dformat.skip -Dvulnerabilitycheck.skip clean package', label: 'Build package'  
    }  
}  
  
stage('Build Image') {  
    steps {  
        sh 'docker build -t ${L_DOCKER_IMAGE} -t ${L_DOCKER_LATEST_IMAGE} --build-arg DEPLOY_ENV=${L_ENV_NAME} .'    }  
}
```

Figure 24- Build package and image to use with trivy

Then, we can stage the Trivy Scanning for the image built, as we see in figure 26. First, we need to get the binary file and extract it, so that after that we can execute trivy.

The parameters passed on trivy are -f (format json) -o (to save the output on a file named results.json) and --ignore-unfixed (ignores CVE specified in .trivignore, represented in Figure 25).

```

# .trivignore
You, 2 weeks ago | 1 author (You)
1 # a libc vulnerability in the base image, marked as **DISPUTED** by the NVD and non-existent on CVE Repository
2
3 CVE-2019-1010022

```

Figure 26- .trivignore file

```

// Scan for vulnerabilities in images
// Fail Build if at least 1 HIGH vulnerability is found; Warns the user if at least 2 NORMAL vulnerabilities are found
stage("Trivy Image Scanning"){
  steps [
    sh """
    wget https://github.com/aquasecurity/trivy/releases/download/v0.22.0/trivy_0.22.0_Linux-64bit.tar.gz
    tar -xf trivy_0.22.0_Linux-64bit.tar.gz
    ./trivy image -f json -o results.json --ignore-unfixed ${L_DOCKER_IMAGE} |
    """
    recordIssues(qualityGates: [[threshold: 1, type: 'TOTAL_HIGH', unstable: false], [threshold: 2, type: 'TOTAL_NORMAL', unstable: true]], tools: [trivy(pattern: 'results.json')])
  ]
}

```

Figure 25- Trivy stage on JenkinsFile

This CVE that we're ignoring was a false positive and we can see that by searching this vulnerability on the CVE repository and getting no answers.

Besides that, we can search it on the NVD repository, and we'll get a result like figure 27, that this CVE is now classified as ****DISPUTED****.

🚩 CVE-2019-1010022 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Current Description

**** DISPUTED **** GNU Libc current is affected by: Mitigation bypass. The impact is: Attacker may bypass stack guard protection. The component is: nptl. The attack vector is: Exploit stack buffer overflow vulnerability and use this bypass vulnerability to bypass stack guard. NOTE: Upstream comments indicate "this is being treated as a non-security bug and no real threat."

Figure 27- Ignored CVE by Trivy

Finally, we can see that I've defined some quality gates in the figure 28, which can be changed in the future. This way, as it's configured now, the build on Jenkins will fail if there's any HIGH/CRITICAL vulnerability found, and if there are at least 2 NORMAL vulnerabilities the build will raise a Warning.

Stage View

		Checkout SCM	Notify Bitbucket of build start	Build & Test	Archive Artifacts	Process Test Results	Process Lint Results	Process FindBugs Results	Notify Bitbucket of build result
Average stage times:		1min 48s	708ms	8min 35s	4s	1s	938ms	4s	748ms
#11	Nov 14 16:01 1 commits	1s	565ms	7min 24s	3s	1s	933ms	4s	638ms
#10	Nov 14 15:31 1 commits	2min 7s	1s	8min 49s	4s	1s	1s	4s	1s

Figure 28- Warning in Jenkins

5.1.3.3 Output

To check the output, we just have to enter in the last successful build and click on a button called “Aquasec Trivy Warnings”, and we’ll get something like Figure 29 (the files were all named as “libs” for security reasons):

Aquasec Trivy Warnings

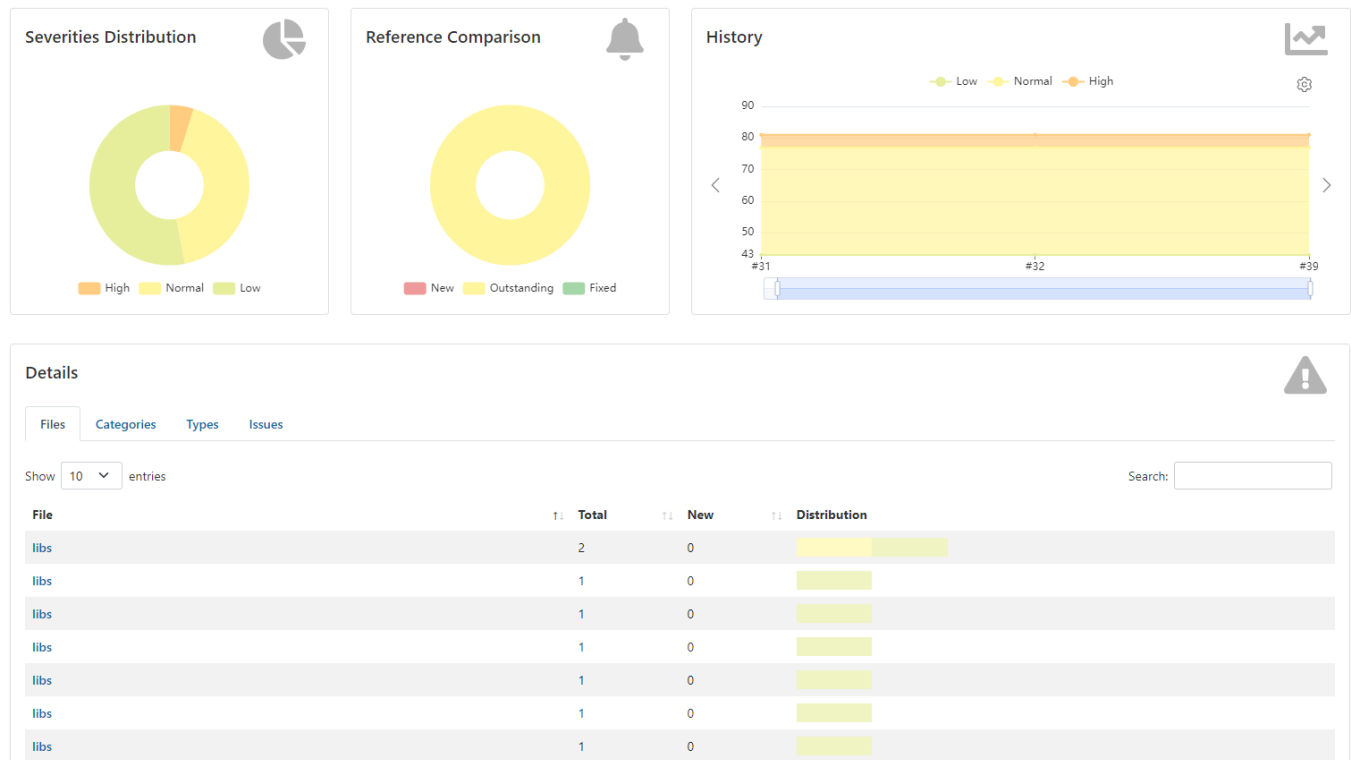


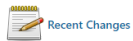
Figure 29- Aquasec Trivy Output

5.1.3.4 Case of Success

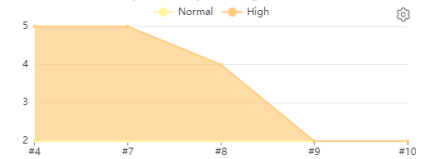
A couple weeks after this tool has been implemented, we were able to see it in action, when a developer, working with microservices, opened a PR and his build failed because of a CRITICAL vulnerability, and probably the most known vulnerability of 2021 (CVE-2021-44228 Log4J), mentioned in Figure 30 and 31.

Pull Request PR-1

Full project name: Notification/notification/PR-1



Aquasec Trivy Warnings Trend



Stage View

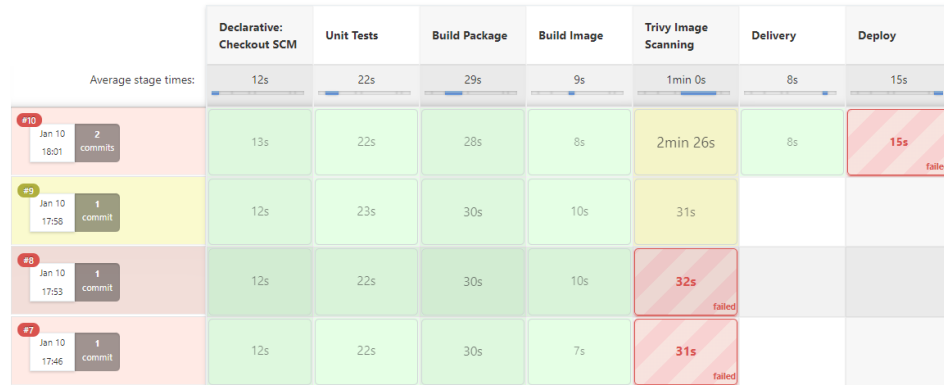


Figure 30- Pipeline fail due to vulnerability

Aquasec Trivy Warnings

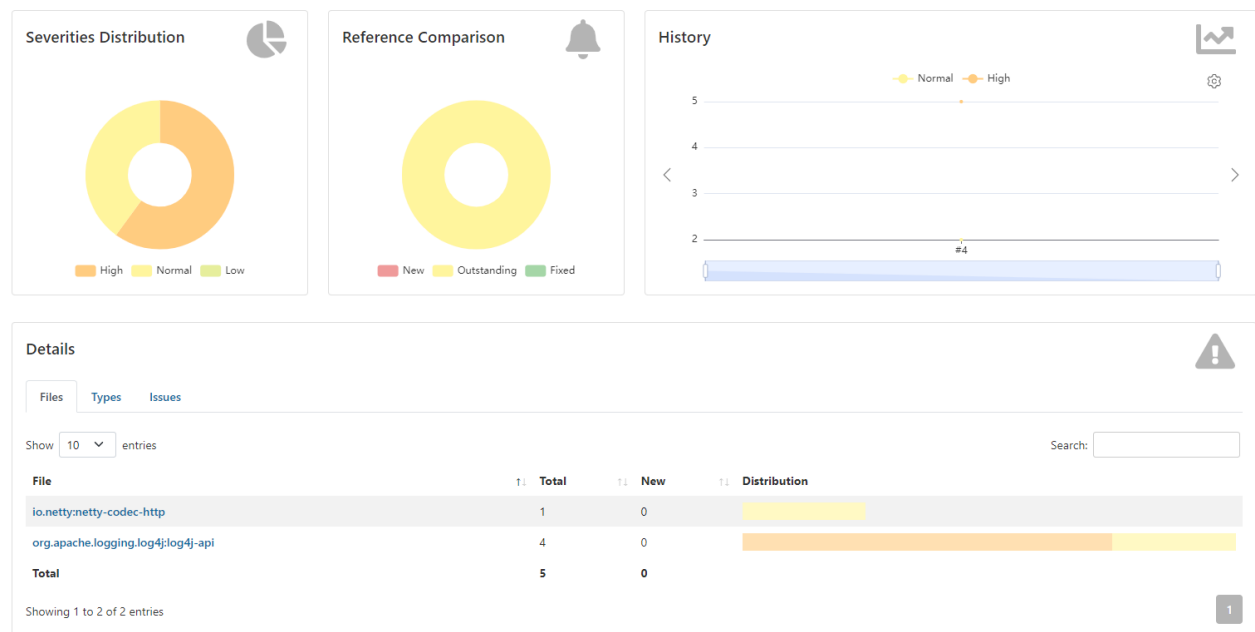


Figure 31- CVE-2021-44228 caught by Trivy

5.1.4 OWASP ZAP

5.1.4.1 Installation

1. `docker pull owasp/zap2docker-stable`
2. Install `zap.sh` from `zapproxy` site.

5.1.4.2 Usage

First, we want to decide what type of scan we're going to use. There are three types of scans: **Baseline Scan**, **Full Scan** and **API Scan**.

The **Baseline Scan** runs the ZAP spider against the specified target for (by default) 1 minute and then waits for the passive scanning to complete before reporting the results. This scan it's a less evasive scan, since it doesn't perform any actual attacks against the target.

The **Full Scan** runs the ZAP spider against the specified target (by default with no time limit) followed by an optional ajax spider scan and then a full active scan before reporting the results. In this case we're performing an attack against the target.

The **API Scan** is tuned for performing scans against APIs defined by OpenAPI, SOAP, or GraphQL via either a local file or a URL. It imports the definition that we specify and then runs an Active Scan against the URLs found. (OWASP, 2022)

In this case we're going to perform a **Full Scan** against the dev environment of one of our tools.

5.1.4.3 Testing

First make sure that you have docker running on background and open your Linux terminal, in this case Ubuntu 20.04.

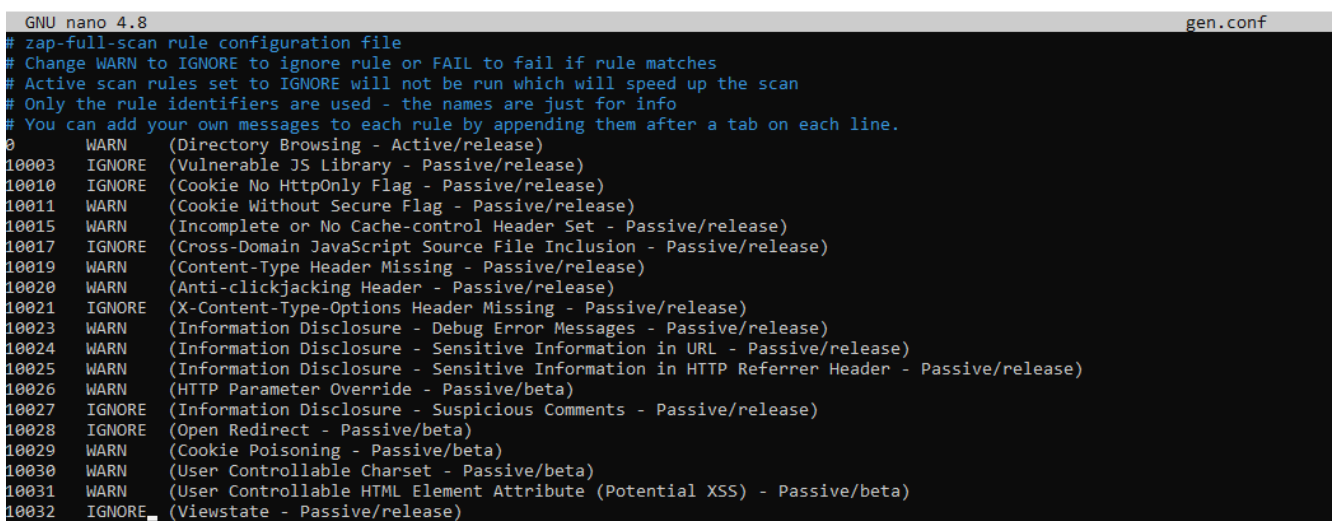
Once we're on our machine and after selecting the type of scan we're going to perform we should set the parameters of the scan.

Example:

```
docker run -u 0 -v $(pwd):/zap/wrk/:rw -it
owasp/zap2docker-stable zap-full-scan.py -g gen.conf -t
https://dev-environment.criticaltechworks.bmw.com/
```

From “docker run” to “zap-full-scan.py” we're passing the docker parameters, and, in this case, we're setting a volume, which means that is the docker machine in which we'll perform the scan.

After defining the type of scan, I wanted to generate a configuration file where I can choose to ignore some warnings with the -g parameter, as we can see here in Figure 32.



```
GNU nano 4.8 gen.conf
# zap-full-scan rule configuration file
# Change WARN to IGNORE to ignore rule or FAIL to fail if rule matches
# Active scan rules set to IGNORE will not be run which will speed up the scan
# Only the rule identifiers are used - the names are just for info
# You can add your own messages to each rule by appending them after a tab on each line.
0      WARN    (Directory Browsing - Active/release)
10003  IGNORE  (Vulnerable JS Library - Passive/release)
10010  IGNORE  (Cookie No HttpOnly Flag - Passive/release)
10011  WARN    (Cookie Without Secure Flag - Passive/release)
10015  WARN    (Incomplete or No Cache-control Header Set - Passive/release)
10017  IGNORE  (Cross-Domain JavaScript Source File Inclusion - Passive/release)
10019  WARN    (Content-Type Header Missing - Passive/release)
10020  WARN    (Anti-clickjacking Header - Passive/release)
10021  IGNORE  (X-Content-Type-Options Header Missing - Passive/release)
10023  WARN    (Information Disclosure - Debug Error Messages - Passive/release)
10024  WARN    (Information Disclosure - Sensitive Information in URL - Passive/release)
10025  WARN    (Information Disclosure - Sensitive Information in HTTP Referrer Header - Passive/release)
10026  WARN    (HTTP Parameter Override - Passive/beta)
10027  IGNORE  (Information Disclosure - Suspicious Comments - Passive/release)
10028  IGNORE  (Open Redirect - Passive/beta)
10029  WARN    (Cookie Poisoning - Passive/beta)
10030  WARN    (User Controllable Charset - Passive/beta)
10031  WARN    (User Controllable HTML Element Attribute (Potential XSS) - Passive/beta)
10032  IGNORE  (Viewstate - Passive/release)
```

Figure 32- ZAP config file

Finally, I set a target so that I can perform this scan, which happens to be the <https://dev-environment.criticaltechworks.bmw.com/>. (This URL doesn't exist, just used as a demonstration)

And we'll get an output like this one, presented in figure 33:

```
PASS: Possible Username Enumeration [40023]
PASS: SQL Injection - SQLite [40024]
PASS: Cross Site Scripting (DOM Based) [40026]
PASS: SQL Injection - MSSQL [40027]
PASS: ELMAH Information Leak [40028]
PASS: Trace.axd Information Leak [40029]
PASS: .htaccess Information Leak [40032]
PASS: .env Information Leak [40034]
PASS: Hidden File Finder [40035]
PASS: Source Code Disclosure - Git [41]
PASS: Source Code Disclosure - SVN [42]
PASS: Source Code Disclosure - File Inclusion [43]
PASS: Script Active Scan Rules [50000]
PASS: Script Passive Scan Rules [50001]
PASS: Path Traversal [6]
PASS: Remote File Inclusion [7]
PASS: Insecure JSF ViewState [90001]
PASS: Charset Mismatch [90011]
PASS: XSLT Injection [90017]
PASS: Server Side Code Injection [90019]
PASS: Remote OS Command Injection [90020]
PASS: XPath Injection [90021]
PASS: XML External Entity Attack [90023]
PASS: Generic Padding Oracle [90024]
PASS: Expression Language Injection [90025]
PASS: SOAP Action Spoofing [90026]
PASS: Cookie Slack Detector [90027]
PASS: Insecure HTTP Method [90028]
PASS: SOAP XML Injection [90029]
PASS: WSDL File Detection [90030]
PASS: Loosely Scoped Cookie [90033]
PASS: Cloud Metadata Potentially Exposed [90034]
WARN-NEW: Incomplete or No Cache-control Header Set [10015] x 4
    https: .com/ (200 OK)
    https: .com/robots.txt (200 OK)
    https: .com/sitemap.xml (200 OK)
    https: .com/manifest.json (200 OK)
WARN-NEW: Missing Anti-clickjacking Header [10020] x 2
    https: .com/ (200 OK)
    https: .com/sitemap.xml (200 OK)
WARN-NEW: X-Content-Type-Options Header Missing [10021] x 11
    https: .com/ (200 OK)
    https: .com/robots.txt (200 OK)
```

Figure 33- ZAP output

5.2 Threat Model

5.2.1 Installation

The installation of this tool could be a little tricky, but here's how I managed to do it:

1. Log into your GitHub account, go to Settings -> 'Developer settings' -> 'OAuth Apps' -> 'New OAuth App'
2. Fill out the form with the following:
 - **Application name:** A unique identifier for the application. This is not critical; we suggest something like 'Threat Dragon'
 - **Homepage URL:** For local development, use `http://localhost:3000`
 - Threat Dragon defaults to port 3000 but is configurable. If you plan to run it on another port, be sure to use that port instead!
 - **Application description:** A description for your OAuth app. This is not critical; we suggest something like 'Threat Dragon for local development'
 - **Authorization callback URL:** `http://localhost:3000/oauth/github`
 - Again, if you plan to run Threat Dragon on another port, use that port instead!
3. Create a client secret
4. Install openssl if not installed at:
<https://slproweb.com/products/Win32OpenSSL.html>
5. Generate a key using: `openssl rand -hex 16`
6. Edit the file `example.env` with all the info, ex:

```
GITHUB_CLIENT_ID=6fewqcdxf5465g4
GITHUB_CLIENT_SECRET=4rcwesdrf6r6evfr6sdgvbtf70
GITHUB_SCOPE=repo
NODE_ENV=development
```

```

SESSION_STORE=local

SESSION_SIGNING_KEY=878fh23478hfrhf2340

SESSION_ENCRYPTION_KEYS= [{"isPrimary": true, "id": 0,
"value": "878fh23478hfrhf2340"}]

```

7. Go to powershell inside threat dragon folder and paste this cmd with your path:

```

docker run -it --rm -p 3000:3000 --env-file
C:/Users/ctw01669/Documents/threat-dragon/.env owasp-threat-
dragon:dev

```

8. Go to `http://localhost:3000` and sign in with your github account.

5.2.2 Threat Model for the Internal Tools

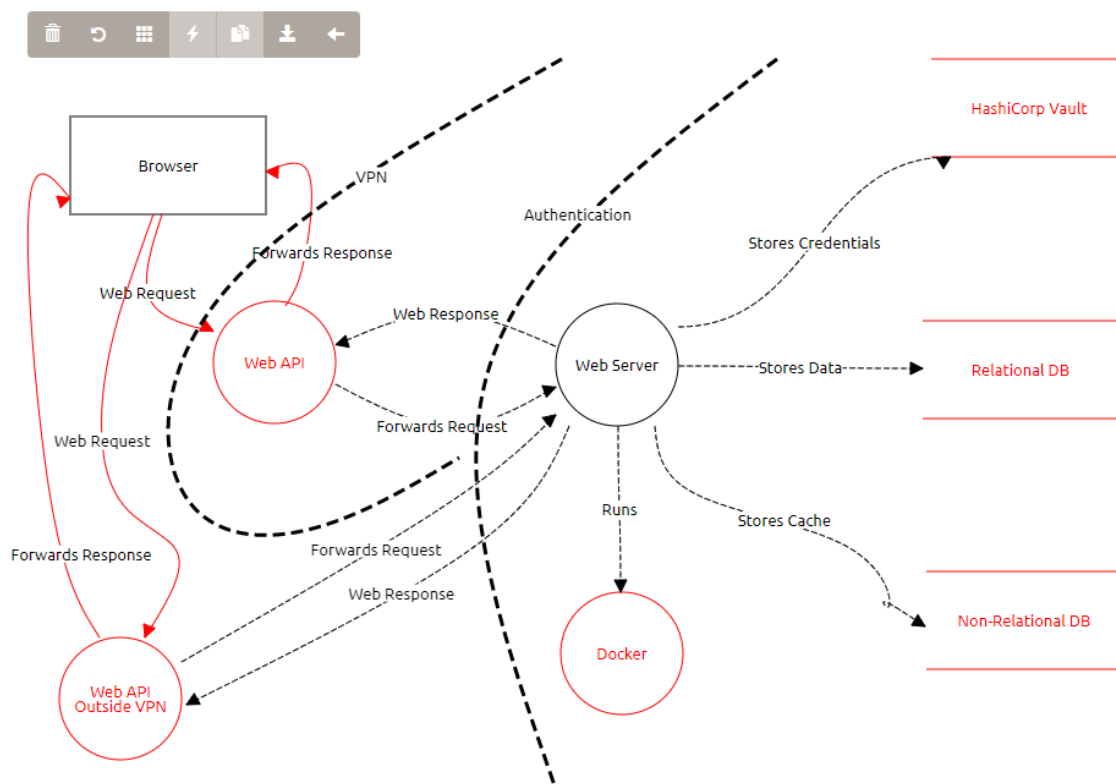


Figure 34- Internal Tools Threat Model

After finishing the threat model, as we can see in figure 34, we can export a report in pdf for better visualization of the threats, the description of the threats, its mitigation, ...

As we all know SQL Injection is still one of the biggest threats to any Web Application and can run down a company if their database is exposed online, so I'll paste a part of the report, in figure 35, where I've explained what's SQLi (SQL injection) and defined some of the ways to mitigate it.

Relational DB (Data Store)

Description:

SQLi

Information disclosure, Open, High Priority

Description:

SQL injection, also known as SQLI, is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. This information may include any number of items, including sensitive company data, user lists or private customer details.

Mitigation:

- Trust no-one: Assume all user-submitted data is evil and validate and sanitize everything.
- Don't use dynamic SQL when it can be avoided: used prepared statements, parameterized queries or stored procedures instead whenever possible.
- Update and patch: vulnerabilities in applications and databases that hackers can exploit using SQL injection are regularly discovered, so it's vital to apply patches and updates as soon as practical.
- Firewall: Consider a web application firewall (WAF) – either software or appliance based – to help filter out malicious data. Good ones will have a comprehensive set of default rules, and make it easy to add new ones whenever necessary. A WAF can be particularly useful to provide some security protection against a particular new vulnerability before a patch is available.
- Reduce your attack surface: Get rid of any database functionality that you don't need to prevent a hacker taking advantage of it. For example, the xp_cmdshell extended stored procedure in MS SQL spawns a Windows command shell and passes in a string for execution, which could be very useful indeed for a hacker. The Windows process spawned by xp_cmdshell has the same security privileges as the SQL Server service account.

Figure 35- Part of Threat Dragon Report

5.3 Results

In the following graph, represented in figure 37, we can compare the security measures that our team already had implemented before I joined them and the points, I was able to implement while I worked here.

We may notice that the “Nice-to-have column” suffered a downgrade, because many of these decisions to implement these security measures must be discussed with a superior and it was decided that this wasn’t the moment, or it was just against the company policy to, for example, nominate a “security champion on each team” or to organize a “simple mob hacking”.

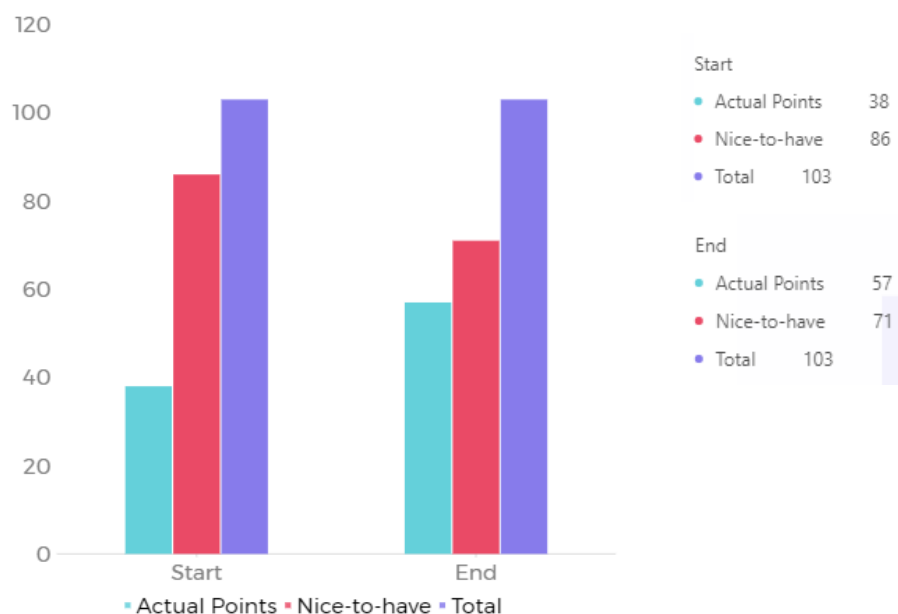


Figure 36- DSOMM Points Comparison

As we can see there’s a huge approximation from the blue to the red column, which are the values we must take into consideration.

These results were calculated by giving one point to each level 1 practice and two points to each level 2 practice of the DSOMM. This is just a way to visually explain how the security awareness and practices inside the team have raised substantially, according to what was implemented during these couple of months.

6 Conclusions

The documented internship was aimed to implement security before deployment, and as we've seen by the results it was a success.

This chapter summarizes the work performed, its process and suggests an approach for future work.

6.1 The Process

When I first got here, I had no clue what I was going to do, and even though I had a great orientation from my DevOps colleagues, they were no experts in this security field, so I had to do a lot of research work.

My first two months at the company were dedicated to study what was done on other big companies and to test some tools that I could use later.

I've tested tools that I didn't integrate on this process before SSDLC, such as:

- SpotBugs
- OWASP Defect-Dojo
- OWASP Security Knowledge Framework (even though it was presented to the developers)
- Clair
- Semgrep

I didn't talk about ESLint, which is a SAST tool that was added its dependency to the package.json of a React project, because it's not automated.

I added two scripts as well: `yarn lint` and `yarn lint html`, and included this information on the README file, so that the developers can perform a simple static analysis scan and export it as a HTML report, or just read it on their console.

After these months of research, I've finally started implementing what I've been studying with the help of the DevOps team that already had the knowledge of Jenkins, Docker, ...

My last task on this internship was to document everything I did and to provide a folder, with all the research that I've done, as we can see in figure 36, so that the developers can maintain these tools and maybe, if someone continues what I've done here at CTW, this person doesn't have to start from scratch.



















	DAST Tools		13/01/2022 10:33	File folder	
	Maturity Model		13/01/2022 10:32	File folder	
	SAST Tools		13/01/2022 10:30	File folder	
	SCA Tools		13/01/2022 10:32	File folder	
	Threat Modelling		13/01/2022 10:34	File folder	
	Integrating Tools in Pipeline		13/12/2021 20:26	Microsoft Word D...	464 KB
	OWASP Defect Dojo		14/12/2021 15:13	Microsoft Word D...	220 KB
	OWASP Security Knowledge Framework		15/11/2021 17:35	Microsoft Word D...	103 KB
	Tools to scan containers for vulnerabilities		13/01/2022 10:38	Microsoft Word D...	1,285 KB

Figure 37- Folder provided with all the research

6.2 Future Work

As we know there are a lot of other practices that can be implemented in this SSDLC, such as security unit tests, smoke tests, check for malware, ...

The first thing I'd give priority would be to implement an API Gateway on the dev environment, so that we could integrate in the pipeline a DAST tool, such as OWASP ZAP, that right now is not able to bypass the authentication with an API key.

Another common thing that the dev team usually does is modifying the quality gates or even comment some security stages to make the build pass, which is not a good practice at all. Sometimes is just better to delay a release if it comes to that point.

Another practice that I've noticed that it's common is to add dependencies without checking its composition. Sometimes we only need to spend some more minutes implementing something that is secure, instead of adding some dependency without checking its source and its dependency tree. If you want to add a dependency, make sure that you run Dependency-Track or simply the command `mvn dependency:Tree`, if you're working with maven, to check the dependencies attached to the dependency imported or to the plugin.

Finally, don't let the threat model be forgotten. I think it would be useful to include in a starter kit of the team, so that everyone's aware of what they're dealing with.

References

- Aquasec. (2021, 12 28). *Trivy*. Retrieved from Github:
<https://github.com/aquasecurity/trivy>
- Atlassian. (2022, 1 14). *DevSecOps*. Retrieved from Atlassian:
<https://www.atlassian.com/continuous-delivery/principles/devsecops>
- BMW. (n.d.). *Critical Techworks*. Retrieved from BMW:
<https://www.bmw.pt/pt/topics/fascination-bmw/critical-techworks.html>
- CI/CD. (n.d.). Retrieved from Wikipedia: <https://en.wikipedia.org/wiki/CI/CD>
- Critical Techworks. (2021, 10 30). *About*. Retrieved from Critical Techworks:
<https://www.criticaltechworks.com/>
- DevOps*. (2021, 12 5). Retrieved from Atlassian: <https://www.atlassian.com/devops>
- Docker (software)*. (n.d.). Retrieved from Wikipedia:
[https://en.wikipedia.org/wiki/Docker_\(software\)](https://en.wikipedia.org/wiki/Docker_(software))
- Docker. (n.d.). *Docker overview*. Retrieved from Docker: <https://docs.docker.com/get-started/overview/>
- Fowler, M. (2014, 08 26). *MaturityModel*. Retrieved from martinowler.com:
<https://martinfowler.com/bliki/MaturityModel.html>
- Gallagher, P. (2021, 4 30). *What are the benefits of DevSecOps*. Retrieved from Blog Good Learning: <https://blog.goodelearning.com/subject-areas/devsecops/benefits-of-devsecops/>
- Henriquez, M. (2021, 12 9). *The top data breaches of 2021*. Retrieved from Security Magazine: <https://www.securitymagazine.com/articles/96667-the-top-data-breaches-of-2021>
- Holmes, A. (2021, 4 3). Insider. *533 million Facebook users' phone numbers and personal data have been leaked online*. Retrieved from

<https://www.businessinsider.com/stolen-data-of-533-million-facebook-users-leaked-online-2021-4?r=DE&IR=T>

IBM. (n.d.). *Docker*. Retrieved from IBM: <https://www.ibm.com/topics/docker>

Jones, D. (2018, Mar 16). *Containers vs. Virtual Machines (VMs): What's the Difference?* Retrieved from NetApp: <https://www.netapp.com/blog/containers-vs-vms/>

Kubernetes. (2021, Jul 23). *What is Kubernetes?* Retrieved from Kubernetes: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

Kubernetes. (n.d.). *What is Kubernetes?* Retrieved from kubernetes: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

McCoy, L. (n.d.). *Microsoft Azure Explained: What It Is and Why It Matters*. Retrieved from CCB Technology: <https://ccbtechnology.com/what-microsoft-azure-is-and-why-it-matters/>

Microsoft Azure. (2021, 11 30). *What is a Container?* Retrieved from Azure: <https://azure.microsoft.com/en-us/overview/what-is-a-container/#overview>

Mukherjee, J. (n.d.). *Devsecops*. Retrieved from Atlassian CI/CD: <https://www.atlassian.com/continuous-delivery/principles/devsecops>

Nadeau, D. S. (2019, Sep 26). *3 DevSecOps success stories*. Retrieved from CSO: <https://www.csoononline.com/article/3439737/3-devsecops-success-stories.html>

NTIA. (2021, 12 20). *SBOM*. Retrieved from NTIA: <https://www.ntia.gov/SBOM>

NVD. (2021, Dec 10). Retrieved from NIST: <https://nvd.nist.gov/vuln/detail/CVE-2021-44228>

OWASP. (2021, 12 20). *Dependency Track*. Retrieved from Dependency Track: <https://docs.dependencytrack.org/>

OWASP. (2021, 11 10). *Dependency-Check*. Retrieved from OWASP: <https://owasp.org/www-project-dependency-check/>

- OWASP. (2021, 12 23). *Dependency-Check Comparison*. Retrieved from Dependency-Track: <https://docs.dependencytrack.org/odt-odc-comparison/>
- OWASP. (2021, 12 22). *OWASP*. Retrieved from OWASP Threat Dragon: <https://owasp.org/www-project-threat-dragon/>
- OWASP. (2021, 10 30). *OWASP Devsecops Maturity Model*. Retrieved from OWASP: <https://owasp.org/www-project-devsecops-maturity-model/#>
- OWASP. (2021, 11 3). *Source Code Analysis Tools*. Retrieved from OWASP: https://owasp.org/www-community/Source_Code_Analysis_Tools
- OWASP. (2022, 1 10). *Zap Docker Documentation*. Retrieved from ZAP: <https://www.zaproxy.org/docs/docker/>
- PostgreSQL. (n.d.). Retrieved from PostgreSQL: <https://www.postgresql.org/about/>
- Rapid7. (2022, 1 13). *DAST*. Retrieved from Rapid7: <https://www.rapid7.com/fundamentals/dast/>
- Scrum.org. (n.d.). *What is Scrum?* Retrieved from Scrum.org: <https://www.scrum.org/resources/what-is-scrum>
- Shafiq Hussain, A. K. (2014). ISSN 1013-5316; CODEN: SINTE 8 . *THREAT MODELLING METHODOLOGIES: A SURVEY*, pp. 1607-1609.
- Sirajuddin Ahmed, S. A. (2019). *Smart Cities- Opportunities and Challenges*. Hina Zia.
- Sumo Logic. (2019, May 21). *AWS*. Retrieved from Sumo Logic: <https://www.sumologic.com/insight/aws/>
- Synopsis. (n.d.). *DevSecOps*. Retrieved from Synopsys: <https://www.synopsys.com/glossary/what-is-devsecops.html#4>
- What is PostgreSQL?* (n.d.). Retrieved from IBM: <https://www.ibm.com/topics/postgresql>
- Workfront. (n.d.). *Waterfall Methodology*. Retrieved from Workfront: <https://www.workfront.com/project-management/methodologies/waterfall>