

Peder Hovdan Andresen
Patrick Eilert Krosby
Anders Christoffer Westby

Auspex

May 2022

NTNU

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Design

Bachelor's thesis

2022



Peder Hovdan Andresen
Patrick Eilert Krosby
Anders Christoffer Westby

Auspex

Bachelor's thesis
May 2022

NTNU

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Design



Norwegian University of
Science and Technology

Auspex

Peder Hovdan Andresen, Patrick Krosby, Anders Christoffer Westby

2022

Abstract

Telenor, one of the leading telecommunication companies in Norway, provides a wide range of services. Many of these services rely on microservices running as Docker containers. For this reason, Telenor has a need to continually evaluate security for the microservices they provide, using various tools that unfortunately produce hard to understand and overly extensive output. To help solve this issue, we we will develop develop an application called Auspex. The application will parse the output from scans of container images, and create human-readable reports of the findings. This will give an overview of the security state of their containers and thus make this information more accessible to a wider audience.

This thesis will cover the process of development for Auspex in its entirety, from the initial planning stages through implementation. The first three chapters covers introductory information regarding the topics explored in this thesis, as well as how the need for our application came to be. The next four chapters will cover the development of our application. This includes requirements, system design, development process and implementation. Finally, the last two chapters will be dedicated to discussing some of the decisions made, concluding the thesis and suggesting further work.

Sammen drag

Telenor, en av de ledende telekommunikasjonsselskapene i Norge, tilbyr et bredt spekter av tjenester. Mange av disse tjenestene er avhengige av mikrotjenester som kjører i Docker containere. Av denne grunn har Telenor behov for å kontinuerlig evaluere sikkerheten i de mikrotjenestene de tilbyr, ved å bruke diverse verktøy som produserer kompliserte dataresultater som er vanskelig å forstå. Målet for dette prosjektet er å utvikle en applikasjon kalt Auspex. Applikasjonen kan gjennomgå dataresultatene fra scanninger av containere, og lage menneskelige forståelige rapporter av funnene som gir et overblikk av sikkerhetstilstanden for en container som dermed gjør denne informasjonen mer tilgjengelig for et bredere publikum.

Denne oppgaven vil dekke prosessen med å utvikle Auspex i sin helhet, fra de innledende planleggingsstadiene, til implementasjonen. De tre første kapitlene dekker introduserende informasjon rundt temaene som er dekket i denne oppgaven, samt hvordan behovet for vår applikasjon oppstod. De neste fire kapitlene dekker utviklingen av vår applikasjon. Dette inkluderer systemkrav, systemdesign, utviklingsprosess og detaljer om implementasjonen. Til slutt, de siste to kapitlene diskuterer noen av valgene som er gjort, konkluderer oppgaven og kommer med forslag til fremtidig arbeid.

Contents

Abstract	i
Sammendrag	ii
Contents	iii
Figures	ix
Tables	xi
Code Listings	xii
Glossary	xiv
1 Introduction	1
1.1 Project Background	1
1.2 Task Description and Project Goals	2
1.2.1 Application Description	2
1.3 Project Scope	3
1.4 Project Group	3
1.5 Relevant Hyperlinks	4
1.5.1 GitHub Repository	4
1.5.2 GitLab Repository	4
1.5.3 Service URLs	4
1.6 Thesis Structure	4
2 Background	6

- 2.1 Cloud Computing and Containers 6
 - 2.1.1 What is a Container? 7
 - 2.1.2 Benefits of Using Containers 7
 - 2.1.3 Containers and Scaling 8
- 2.2 Monolithic vs Microservices 9
 - 2.2.1 Monolithic 10
 - 2.2.2 Microservices 10
 - 2.2.3 Microservices and Containers 10
- 2.3 Container Security 11
 - 2.3.1 Main Vulnerability Sources 11
 - 2.3.2 Container Security Scanning Tools 12
 - 2.3.3 CVE 13
- 2.4 Applicability of our Application 14
 - 2.4.1 Issues with the Current Solution 14
 - 2.4.2 How Our Application Solves These Issues 15
- 3 Related Work 16**
- 4 Requirements 18**
 - 4.1 Functional Requirements 18
 - 4.2 Non-Functional Requirements 19
- 5 System Design 21**
 - 5.1 Application Overview 21
 - 5.2 Detailed Component Description 23
 - 5.2.1 Clients and API Gateway 23
 - 5.2.2 Scanner 25
 - 5.2.3 Reporter 25

- 5.2.4 Automated Report Creation 26
- 6 Development Process 28**
- 6.1 Development Model 28
- 6.2 Documentation 29
 - 6.2.1 Kanban Board 29
 - 6.2.2 Thesis Writing 30
 - 6.2.3 Source Code 30
 - 6.2.4 Other Documentation 30
 - 6.2.5 Meetings and Minutes of Meeting 30
 - 6.2.6 Time Tracking 31
 - 6.2.7 Expenses 31
- 6.3 Routines 32
 - 6.3.1 Work Policies 32
 - 6.3.2 Tools 32
 - 6.3.3 Communication 32
- 7 Implementation 33**
- 7.1 Repository Structure 33
 - 7.1.1 Monorepo and Microservices: an Anti-pattern? 33
 - 7.1.2 Choosing a Repository Structure 35
 - 7.1.3 Implementing Monorepo Builds With Docker and Poetry . . 35
- 7.2 Code Quality and Formatting 36
 - 7.2.1 Mypy 36
 - 7.2.2 Black 36
- 7.3 Data Validation With Pydantic 37
- 7.4 Service: Scanner 38

- 7.4.1 Endpoints 38
- 7.4.2 Running Snyk In A Container 40
- 7.4.3 Storing Scan Results 41
- 7.5 Service: Reporter 42
 - 7.5.1 Endpoints 42
 - 7.5.2 Initiating Report Creation 44
 - 7.5.3 Parsing Scans 46
 - 7.5.4 Single vs Aggregate 47
 - 7.5.5 Generating a PDF Report 51
 - 7.5.6 Creating Tables 55
 - 7.5.7 Creating Plots 57
 - 7.5.8 Final Output 58
- 7.6 Service: API Gateway 59
 - 7.6.1 Endpoints 60
- 7.7 Testing 63
 - 7.7.1 Pytest 64
 - 7.7.2 Fuzzing with Hypothesis 65
 - 7.7.3 Coverage 66
- 7.8 Deployment 67
 - 7.8.1 CI/CD 68
 - 7.8.2 Cloud Run Configuration 70
- 7.9 Sending Emails 70
- 8 Evaluation 75**
 - 8.1 Survey 75
 - 8.1.1 Survey Results 76

- 8.1.2 Survey Feedback and Improvements 77
- 8.1.3 Other Feedback 81
- 8.2 Does It Meet the Requirements? 82
 - 8.2.1 Evaluation of Functional Requirements 82
 - 8.2.2 Evaluation of Non-Functional Requirements 84
- 9 Discussion 87**
 - 9.1 Why Auspex Was Developed 87
 - 9.2 Website 88
 - 9.3 Cloud Run 88
 - 9.4 Email Workflow 89
 - 9.5 Choice of Database 89
- 10 Closing Remarks 91**
 - 10.1 Learning Outcome 91
 - 10.1.1 Project 91
 - 10.1.2 Teamwork and Communication 91
 - 10.1.3 Writing the Thesis 92
 - 10.2 Conclusion 92
 - 10.3 Further Work 92
 - 10.3.1 Website 92
 - 10.3.2 Email Workflow 93
 - 10.3.3 Optimizing Service Docker Images 93
 - 10.3.4 Other Scanning Tools 93
 - 10.3.5 Supporting Multiple Container Registries 93
 - 10.3.6 Feedback 93
- Bibliography 94**

- A Additional Material 97**
 - A.1 Early Report Example 130
 - A.2 Example of a Single Report 135
 - A.3 Example of a Aggregate Report 142
 - A.4 Auspex Survey 150
 - A.5 Auspex Survey Result 161
- B Schemas 168**
 - B.1 POST /reports Response Schema 169
 - B.2 Scan Metadata Schema 171
 - B.3 Report Metadata Schema 172
 - B.4 Reporter Query Parameter Schema 173
- C Minutes of Meeting 176**

Figures

2.2	Public and private registry use case	8
2.3	Virtual Machines (left) vs Containers (right) resource usage	9
2.4	Microservice (left) vs Monolithic (right) Architecture	11
3.1	Some of the reasons for how Snyk calculates Priority Score	17
3.2	An example of how Snyk Code presents found vulnerabilities	17
5.1	Overview of all the components and microservices of Auspex	22
5.2	API Gateway communicates with Scanner and Reporter	24
5.3	Simple illustration of Scanner storage including the schema for Scan Database	26
5.4	Simple illustration of Reporter storage with schema. (Some keys are omitted to ease presentation of the schema.)	27
7.1	API endpoints exposed by the Scanner service	38
7.2	API endpoints exposed by the Reporter service	43
7.3	Reporter service requests a scan log from Scanner	45
7.4	The table data rendered as a LaTeX table	57
7.5	A scatter plot of CVSS mean score for current and previous reports of the same image, with a trend line	58
7.6	Structure of a report	59

7.7	API endpoints exposed by the API gateway	60
7.8	Screenshot of Pytest collecting and running tests	64
7.9	The coverage of the largest source files of the Reporter service . . .	67
7.10	Example of code that is not covered by tests	67
7.11	Cloud Build repository trigger configuration	70
7.12	Visual representation of the workflow.	74
7.13	Visual representation of the workflow.	74
8.6	An example of clickable CVSS ID's in a table listing vulnerabilities .	81

Tables

2.1 CVSSv3 Severity Levels	14
--------------------------------------	----

Code Listings

1	Shared document retrieval function with backoff	34
2	Example of Pydantic usage showing composition of models	37
3	Schema for POST /scans request body	39
4	Schema for GET /status response body	40
5	Running Snyk as a subprocess with Python	41
6	Data structure for documents representing scans in the database . .	42
7	Report request schema for the Reporter service	44
8	Snyk JSON scan log (abbreviated)	46
9	The Pydantic-derived class that corresponds to Listing 8 (abbreviated)	47
10	The interface shared by all report data structures (Abbreviated; ori- ginally >150 lines long)	48
11	Implementation of most_severe_n for an image scanned with Snyk	49
12	Implementation of most_severe_n for the aggregate report data structure	50
13	The VulnerabilityType interface (abbreviated)	51
14	Custom class wrapping a PyLaTeX document (abbreviated)	52
15	LatexDocument method that adds a section containing a table show- ing exploitable vulnerabilities	52
16	Utility method for adding a section containing a longtable	53
17	Utility method for adding a longtable to the document	54
18	The TableData data structure	55
19	Example of a function that returns TableData	56
20	The PlotData data structure	57
21	Schema for requests to the POST /reports endpoint	62
22	Response schema for POST /reports endpoint (most keys omitted)	63
23	Test using assert statement	65
24	Example of fuzzing with Hypothesis (abbreviated)	66
25	Build and deployment script for the Reporter service (abbreviated)	68
26	Retrieving a secret in a workflow.	71
27	Requesting scan and report creation for images in a workflow. . . .	72
28	Workflow step that sends an email.	73
29	Workflow step that sends an email.	73

Glossary

API Application Programming Interface. An abstraction layer over a more complex and opaque underlying implementation that can be used by developers.. 35

artifact Used to describe the output from a process and/or service. . 21

Auspex The name of our application. When used alone, it should be taken to mean the application as a whole, not its constituent parts. 2, 3, 5, 11, 18, 19, 21, 23, 24, 27, 33, 76, 83, 84, 91, 92

boilerplate Refers to code implemented repeatedly with small or no variation to achieve minor functionality. . 57

CI/CD Pipeline Continuous Integration/Continuous Deployment Pipeline. Method for automating integration and deployment of code. 15, 22, 24

CLI Text-based computer interface for input of commands. 25

Cloud Native Describes an approach to building cloud applications that utilize and take advantage of technologies such as containers and microservices to deploy applications in the cloud. 18, 19, 84, 85

coupled applications whose parts/modules are dependent on each other to able to function. xv

Docker Open source containerization platform to deliver software as packages called containers. 6–8, 24, 25

Firestore Managed NoSQL database provided Google Cloud Platform. See: NoSQL. 21, 25

Google Cloud Platform GCP is Google’s cloud platform, offering various cloud services within the categories of compute, network and storage.. 19, 31, 84

- HTTP** Hypertext Transfer Protocol. Text-based application-level protocol primarily used for communication between hosts on a network. 35
- JSON** JavaScript Object Notation. Text-based format used to store and transmit data. 19, 23, 25, 35, 40, 83
- LaTeX** Typesetting system and language with markup tagging. Can be compiled to a wide range of output formats. Widely used in academia . ix, 25, 26, 30, 51, 52, 54, 55, 57
- Matplotlib** Python library used for mathematical typesetting and to create diagrams and figures. 25, 57
- NoSQL** Database design paradigm that emphasizes storing data as key-value pairs, rather than relationally. xiv, 21, 34
- NTNU** Norwegian University of Science and Technology. 3, 30
- PyLaTeX** Python library used for creating and compiling LaTeX files. xii, 26, 51, 52, 55, 57
- Service Level Agreement** An agreement between service provider and customer specifying the level of service that will be provided.. 18
- Snyk** Tool for scanning and finding vulnerabilities in container images. 2, 3, 12, 14, 23, 25, 48
- strongly coupled** see: coupled. 10

Chapter 1

Introduction

This chapter introduces the background and purpose of the project. It also outlines the scope of the project and everyone who participate in the project.

1.1 Project Background

Telenor, one of the leading telecommunication companies in Norway, provides wide array of services and products to about 2.8 million customers in Norway [1]. The products and services ranges from mobile phone plans, to providing internet and TV subscriptions, to both private and commercial customers. As technology has evolved, so has the repertoire of products and services Telenor offers its customers. For instance, Telenor has started offering a tracking service for pets, cloud-storage for its customers, and various other data-collection services with its expansion into the Internet of Things (IoT) market [2]. Additionally, they deliver a wide range of security products and solutions to both private consumers and businesses. A large part of these solutions are cloud based solutions using providers such as Amazon Web Services (AWS)¹, Google Cloud Platform (GCP)² or Microsoft Azure³, and organized in containers, typically created using Docker⁴, and orchestrated using container orchestration solutions such as Kubernetes⁵.

This rapid expansion has lead to new realities and challenges. Their increasingly wide range of services offered, has necessitated a more modern approach to how they run these services and how their IT-infrastructure is organised. Their solu-

¹<https://aws.amazon.com/>

²<https://cloud.google.com/>

³<https://azure.microsoft.com/en-gb/>

⁴<https://www.docker.com/>

⁵<https://kubernetes.io/>

tion, as for many others facing similar challenges, has been a container-based architecture that takes advantage of ease of setup, re-usability, and high availability. However, this comes with security issues, and the tools that are used to solve and mitigate these issues often are not as helpful as they could be, due to the expertise level required to fully comprehend the output from these tools. Therefore, we have been tasked with creating a solution to this problem by Telenor, hereafter referred to as our employer.

1.2 Task Description and Project Goals

Our task is to create a service, called Auspex, that can capture and parse the result of a container vulnerability scan, and produce a corresponding report.

The report produced by the Auspex service should be both readable, and understandable to employees without a specialized technical background. Though this seems straightforward in terms of what our goal is, it does come with certain challenges.

Firstly, we need to develop Auspex, deploy it on on Google Cloud Platform, and make sure it can automatically capture all important information from the scan and generate a report.

Secondly, we need to make sure that the generated report is human-readable and understandable, especially for those who do not have much security background. We plan to follow iterative feedback from our employer on the report produced by Auspex since they will know what information they want to obtain from the report.

1.2.1 Application Description

For this project we will be using Snyk, which is a collection of tools that can automatically detect vulnerabilities in source code, dependencies, container images and Infrastructure as Code definition files [3]. More specifically, we will use the feature of Snyk that scans Docker images for vulnerabilities [4]. In this thesis we will refer to Snyk as a vulnerability scanning tool.

Our application, Auspex is designed to parse the output of Snyk on a single container image or a collection of container images. This is facilitated through a microservice with access to Snyk that scans container images. The resulting scan data is then passed to another microservice that parses this data and presents

it as figures and tables in a PDF document using tools such as Matplotlib⁶ and PyLatex⁷ so as to display it in a readable and understandable format. Utilising Google Workflows⁸, the whole process can be automated, and expanded if need be.

As our employer is mostly hosting their services on GCP, it is beneficial for ease of service integration with their existing infrastructure, if Auspex is hosted on GCP as well.

1.3 Project Scope

This project will focus on the aspects of container security that can be discerned from analysing container images in a non-runtime environment. For this purpose, our application will use the container scanning tool Snyk. Thus, while our analysis is limited to the Common Vulnerabilities and Exposures (CVE) program's list of identified vulnerabilities [5], this list is constantly updated and added to by a dedicated and professional community [nist2022]. There are certain vulnerabilities that our analysis will not include, such as vulnerabilities in the software itself, the system- and network architecture, and how persistent data is stored. This means that our analysis will not be addressing the handling of personal data under the General Data Protection Regulation (GDPR), nor will it evaluate other privacy related issues.

1.4 Project Group

The project owner is Telenor, and our employer contact at Telenor is Eirik Stephansen, Tech Lead IoT at Telenor Mobil. His role is to define our task, clarify requirements and answer any questions we might have. Jia-Chun Lin (Kelly), assistant professor at the Norwegian University of Science and Technology (NTNU) in Gjøvik, is the supervisor for this bachelor project.

Our group consists of three students, Peder Hovdan Andresen, Anders Christoffer Westby, and Patrick Krosby. All of us are following the study program Digital Infrastructure and Cybersecurity at NTNU Gjøvik. We chose this assignment based on our previous experiences with using container-based infrastructure, and orchestration tools such as Kubernetes. Additionally, we have experience developing a service that scans and parses code to receive feedback, which undoubtedly is useful for this project.

⁶<https://matplotlib.org/>

⁷<https://jeltef.github.io/PyLaTeX/current/>

⁸<https://cloud.google.com/workflows>

1.5 Relevant Hyperlinks

This section contains hyperlinks relevant to the project, such as source code repositories and the various URLs associated with the deployed application.

1.5.1 GitHub Repository

We moved to GitHub⁹ after discovering GCP provided native CI/CD support for building applications hosted in GitHub repositories

Repository URL: <https://github.com/auspex-ntnu/auspex>

1.5.2 GitLab Repository

We started off developing the project on NTNU's `gitlab.stud.idi.no` GitLab¹⁰ instance, but after discovering that GCP did not provide a satisfactory CI/CD solution for GitLab repositories. Furthermore, we wanted to use GitHub to ease the process of handing over the repository to our employer in the future.

Repository URL: <https://gitlab.stud.idi.ntnu.no/containers-bachelor/repo>

1.5.3 Service URLs

The relevant Auspex URLs are the following:

Auspex API Gateway (preferred) <https://restapi-qk6stf4ejq-lz.a.run.app>

Scanner service <https://scanner-qk6stf4ejq-lz.a.run.app>

Reporter service: <https://reporter-qk6stf4ejq-lz.a.run.app>

1.6 Thesis Structure

- **Chapter 1 - Introduction:** Describes the main background for this thesis, scope, description of the task and the project goals, application description, all involved in the project, and the structure of the thesis.

⁹<http://github.com/>

¹⁰<https://gitlab.stud.idi.ntnu.no/>

- **Chapter 2 - *Background*:** Describes and goes in to detail about the main technologies and concepts for this bachelor project.
- **Chapter 3 - *Related Work*:** Describes the projects and solutions with similar goals to our project.
- **Chapter 4 - *Requirements*:** Describes the requirements for Auspex.
- **Chapter 5 - *System Design*:** Describes the main components and architecture of Auspex.
- **Chapter 6 - *Development Process*:** Describes how this project was completed.
- **Chapter 7 - *Implementation*:** Describes implementation details of the application, code and tools.
- **Chapter 8 - *Evaluation*:** Describes and evaluates the feedback received from our survey as well as other feedback received from both supervisor and employer.
- **Chapter 9 - *Discussion*:** Describes and discusses various decisions made throughout the project.
- **Chapter 10 - *Closing Remarks*:** Describes the learning outcome from this work, our conclusion and future work for the application.

Chapter 2

Background

This chapter aims to explain the industry evolution that led to the demand for the application we were tasked with developing, as well as introduce core concepts that will be repeatedly discussed throughout this thesis.

2.1 Cloud Computing and Containers

In order to fully understand what container technology is, we must first understand what purpose containers serve, and by extension why they became so popular. The rapid rise of cloud computing in the early to middle 2010's set the stage for new technologies to take advantage of this revolutionary way of delivering services and computing resources. An in-depth explanation of all the reasons for this meteoric rise in popularity is out of scope for this thesis.

It is important to note that in addition to Docker being established, the popularity boost of containers is clearly linked to the cloud computing revolution [6]. This was partly due to how cloud computing services were sold to consumers. The prices of these services were based on actual usage of resources, and therefore no longer based on the maximum capacity of the physical hardware used to host web applications. Naturally, not using more computing resources than needed became increasingly important for companies moving their services to cloud-based infrastructure. As we will see, this is one of the main benefits of using containers, and most likely contributed to their widespread adaptation.

2.1.1 What is a Container?

As with many concepts within the field of information technology, a universal definition of a container is non-existent. However, Intel defines it as "... *an abstract unit of software that is a stand-alone, executable unit that has everything needed to run an application: code, runtime, system tools, and system libraries.*" [7]. While this definition is concise and to the point, it lacks in explanatory power.

Though clearly over-simplified and purely for illustrative purposes, a way to think about containers can be as cakes at different stages in the baking process. Like a cake, a container has a recipe that specifies what ingredients it needs, called a container definition file. Once needed, the container is *built*. Meaning all its dependencies specified are translated to container orchestration commands and installed, in the same way a cake's ingredients are put together and put in the oven. When all instructions in the definition file are executed successfully, we call the resulting file a container *image*, which when deployed with a container runtime environment is simply referred to as a container, or in keeping with our illustrative analogy, a finished cake. Thus, a container is run just as an isolated processes on its host system, containing what it needs, *and only* what it needs, to fulfill the purpose for which it was built. This process is illustrated in Figure 2.1¹.

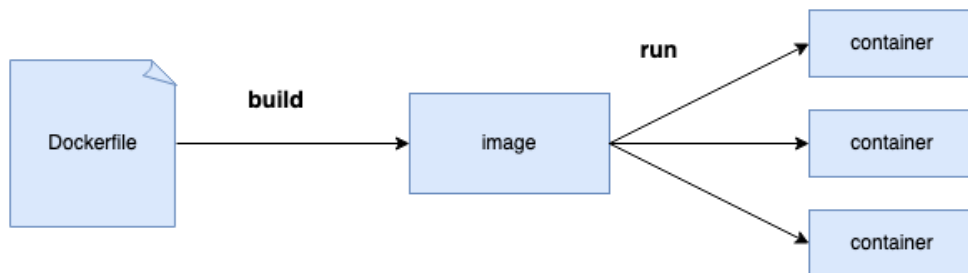


Figure 2.1: Simplified illustration of the containerization process

2.1.2 Benefits of Using Containers

Another important reason for the popularisation of containers is its portability and ease of implementation. While the technology had existed for a while, the founding of Docker Inc. was an important milestone for its widespread use. Docker was the first container orchestration Perhaps the most revolutionary feature of Docker, was the invention of the *Dockerfile*. The *Dockerfile* abstracts away the need for the user to manually create container namespaces, as it manages the whole container creation process. This is the power of Docker. It lets users be concerned with the *what* of their containers, and handles most of the *how*. Some

¹Figure 2.1 is based on: <https://mariadb.com/kb/en/creating-a-custom-docker-image>

of the "how" includes things like managing caching of individual build steps to improve the speed of repeated instructions, and injecting secrets which automates authentication. In keeping with our cake analogy, the Dockerfile is the equivalent to the cakes' recipe, specifying all the needed ingredients, and Docker makes puts them together in the tray ready to be put in the oven for baking.

Note that the Dockerfile not only specified the dependencies for the service, but also allows the creation of process namespaces to keep the container-process isolated. The only dependency remaining therefore, is the host system's kernel. Meaning containers are platform agnostic, and can in most cases run on any OS, allowing for widespread sharing of standardised container images between developers. This allowed developers to start building up libraries of standardised services that they could deploy to their production environments, without having to "re-invent the wheel" every time they wanted to expand their functionality.

Again, Docker inc further enhanced these possibilities by creating *Docker registries*, which allows developers to easily push and pull premade images from the *public* Docker Hub, or from their own *private* registries [8]. If we return to our cake analogy, a Docker registry is like a cookbook containing recipes for specific cakes. Whether that is a public cookbook or a closely guarded family secret, it functions in a similar way. Figure 2.2 illustrates a use case where a base image is pulled from a public registry as specified by a dockerfile, built and pushed to a private registry.

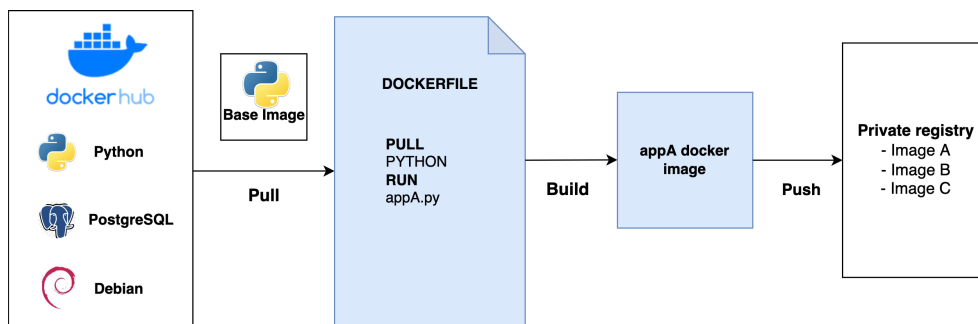


Figure 2.2: Public and private registry use case

2.1.3 Containers and Scaling

So far, we have examined some of the contributing factors to why containers are convenient, but containers also have benefits in the form of performance and scalability. To understand this, we need to examine the differences between containers and their alternative (and in many cases, predecessors), *virtual machines*. Firstly, virtual machines, or VM's for short, are partitioned parts of computing resources that run their own operating system, and can therefore do anything a

physical computer can do, within the bounds of its allocated computing resources.

The partitioned computing resources a virtual machine has access too are fixed however, and will therefore not scale usage with how much it actually needs to perform its tasks. For comparison, since a container runs as any other process, it has access to all of its host systems computing resources but will only use them as they are needed. Additionally a virtual machine can run any operating system. While this is undoubtedly useful for certain things, it also means that there is an added minimal computing resource cost of each virtual machine instance. There are steps one can take to minimize this cost, such as running a Linux distribution that is specifically designed to be as lightweight as possible. However in the context of cloud computing, where not wasting computing resources is so crucial, the fixed partition cost of each VM becomes a significant downside, as illustrated in Figure 2.3.

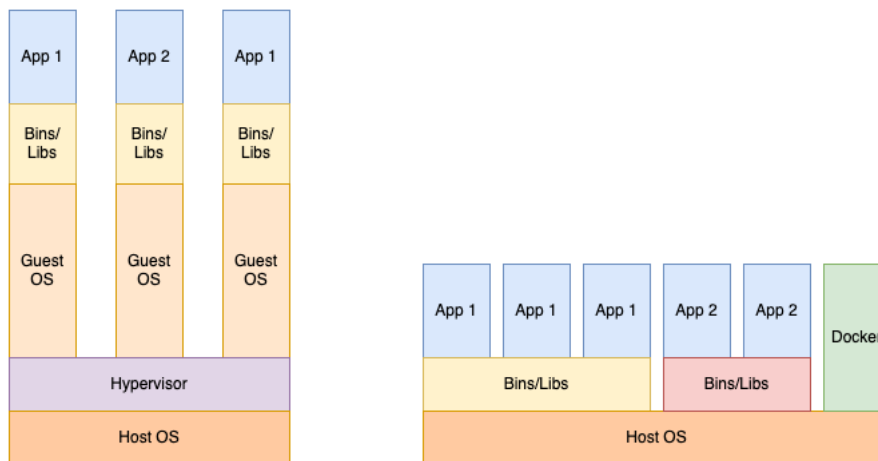


Figure 2.3: Virtual Machines (left) vs Containers (right) resource usage

2.2 Monolithic vs Microservices

To understand the purpose of our application and thesis, we first need to explain the rise of microservices, and why container technology compliments this type of architecture so well. We will begin by explaining the differences between monolithic and microservice oriented architectures, before moving on to microservices and containers.

2.2.1 Monolithic

A monolithic application is an application that can have many different functionalities, but only has one executable. A prudent programmer will often divide the different functionalities the program provides into *modules* for simplicity purposes. Yet, even though different modules of the software may do very different things, none of them can work without each other, as the program only has one executable. We call applications with these characteristics *strongly coupled*, referring to each module being dependant on the others in order to function.

2.2.2 Microservices

Microservices are not radical patented inventions that someone explicitly came up with, but rather the result of continually evolving best-practices to suit the ever expanding landscape of available technologies. For this reason, a universally agreed upon definition of the term does not exist, but there have been attempts. Drogani et al. (2017) defines it as such: "A *microservice is a cohesive, independent process interacting via messages*", and a microservice architecture as "... a *distributed application where all its modules are microservices*." [9]. Cohesive in this case, refers to each service or application being specifically designed for a singular function, which is a core distinction between a microservice oriented software architecture and a monolithic one.

What this means in practice, is that microservices are independent from each other. One service can be taken down without the others being affected, meaning that for a large complex web-application, microservices add a crucial layer of resiliency. We call this type of architecture, *loosely coupled*, which is the opposite of its monolithic *strongly coupled* counterpart. Additionally, microservice oriented architecture makes making changes to the software, particularly adding functionality, far easier than before. In fact, certain types of functionality can be added to a web-application without even taking the application down while doing so, allowing for much faster development cycles and agility. The Figure 2.4 visualises the difference between the architecture types.

2.2.3 Microservices and Containers

Recall the portability and reusability benefits that containers add. The use of premade container definition files, allow developers to share and re-use their software amongst themselves without having to write any code. If the software within these shared containers were monolithically programmed, i.e. containing many different functionalities and dependencies, their reusability is significantly

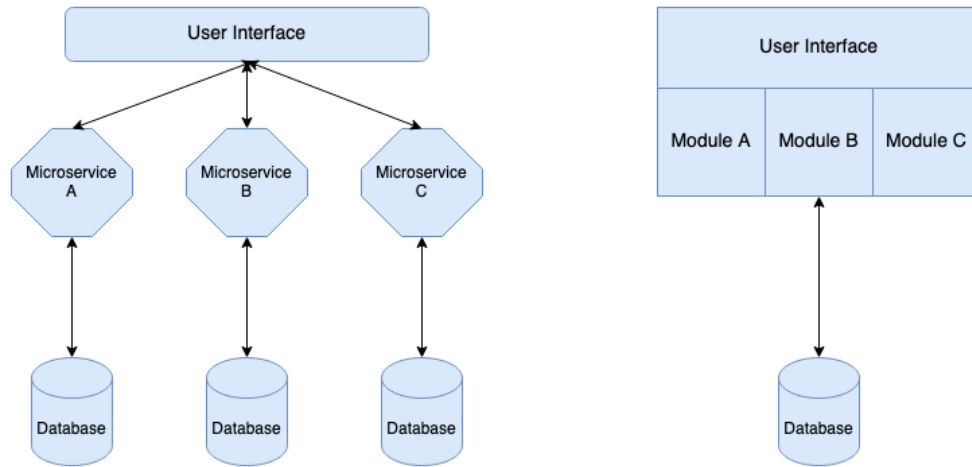


Figure 2.4: Microservice (left) vs Monolithic (right) Architecture

reduced in the context of cloud computing due to their cost to both computing resources and resiliency. Additionally, containers makes the the different functionalities that make up a web-application more shareable. In keeping with our cake analogy, a recipe for a cake that is split into multiple independent parts that are subsequently combined, can be used by far more people and for more cake types, than a recipe for a wedding cake. Thus, the benefits of using containers are more or less negated when designing software with a monolithic architecture in web-applications.

2.3 Container Security

So far, we have examined the benefits of using containers, and their role in the cloud computing revolution. However, we have alluded to the downsides of using containers, namely security. It should be noted, that when discussing this topic we do so in a relative sense. Containers are not known to be a particularly vulnerable technology, but they are nonetheless more vulnerable than some of their alternatives, particularly virtual machines [10]. In this section, we will elaborate on the reasons for this, before explaining how this relates to the requirements of Auspex, the application we are developing.

2.3.1 Main Vulnerability Sources

The main reason for the higher vulnerability of containers compared to virtual machines, is that a container runs on its host operating system, which means it is using the host system's kernel. A virtual machine, by contrast, needs its own

operating system, libraries, dedicated resources and applications to run anything. The added layer of abstraction means that at least theoretically, malware and other malicious code executed on a virtual machine will only affect its own operating system and applications, and not its host's.

This is not necessarily the case with containers however, as they share the underlying kernel and run on the host's operating system. There is thus a non-zero chance of malicious code spreading outside of the container's namespace, assuming an attacker manages to escape it [10]. Further, given the mass scale and deployment of containers in cloud computing, these types of incidents can and do occur. There are of course, layers of protection in containers as well, and certain security related practices that developers are highly encouraged to follow to avoid the most common pitfalls.

2.3.2 Container Security Scanning Tools

As with most vulnerabilities, most originate from running outdated versions of software. In cloud computing, such issues are exasperated further by the sheer volume of clients they serve, thus exposing themselves to a proportionally large amount of malicious actors. Also, given the ease of deployment for pre-made containers, a container definition file can specify outdated and or vulnerable versions of software to run, which in turn can be deployed en-masse. Fortunately though, since the container definition files specify the versions of all software, we can use container security scanning tools to catch commonly known vulnerabilities before they are deployed to production. Intuitively, this can be thought of as scanning a cake-recipe for dangerous ingredients before baking and serving hundreds of them, as opposed to serving first and testing for danger later.

These scanning tools include, but are not limited to *Docker Bench Security*, *Kube-bench*, and *Snyk*. They all work in similar ways, but have certain implementation related differences. For our project we use Snyk, as this tool was recommended by our employer, and is shipped with Docker as default. [4].

The only issue with these tools is the interpretation of the output they produce. It is very extensive in length, and difficult to understand without thorough analysis by someone who is familiar with its structure from before. Given the availability of qualified personnel to do this for each scan, it may not be a problem. However, this approach negates both the speed and agility a microservice oriented approach provides when taking services in and out of production or making changes. An automated approach to container security for every image deployed to production seems a better solution, and is one of the main functionalities our application aims to implement.

2.3.3 CVE

So far, we have explained what scanning tools are available, and to a lesser extent our reasoning behind choosing to use Snyk for our application. What we have eluded to however, is how Snyk identifies each vulnerability and its severity, and what factors contribute to what extent. In this section, we will elaborate further on the some of the intricacies of these systems, due to their critical role in the functionality of our application.

Snyk uses an open database called Common Vulnerability and Exposures (CVE), that keeps records of all publicly known vulnerabilities. We will refer to each vulnerability as CVEs throughout this thesis. Each CVE has a unique ID which enables third party systems and databases such as Snyk to accurately identify them regardless of context [11]. Additionally, each CVE uses the Common Vulnerability Scoring System (CVSS) to calculate and rank the severity of each vulnerability. The CVSS is not a a system specifically designed for use in container scanning tools, but rather a system for objectively ranking *any* software vulnerabilities. It allows users of this system to prioritise which issues are the most pressing, and thus theoretically allow for an optimized distribution of security measures.

The different factors that make up the final severity score can be separated into one *base* metric, and three modifying categories, *temporal*, *impact*, and *environment* [12]. The base metrics are calculated from an assessment of the degree of difficulty to gain access. Taking into account access vectors, access complexity and to what degree authentication is required. This base score, as the name suggests, is used as the base value in the calculation of all the subsequent categories. The impact metrics are calculated based on confidentiality, integrity and availability. Further, the temporal metrics are an attempt at an objective way of measuring how a vulnerability changes over time, as exploits, patches and mitigation techniques are developed. These include report confidence, remediation levels, and crucially whether or not a proven exploit has been found. Finally, the environment factors refers to how much horizontal mobility an attacker would gain in the event of a successful attack [12].

The minute detail of each mathematical calculation being made to reach the final CVSS score are out of scope for this thesis, but knowing what factors it takes into account is still important information to security decision makers. The final CVSS score is a number that ranges between 0 and 10 with one decimal, with a corresponding tier system ranging from 'low' to 'critical', as illustrated in Figure ???. Note that the CVSS score is not a cumulative score reflecting the security of the environment as a whole, but rather a score reflecting the severity of *each* CVE in a container image [12].

CVSS Score	Severity Rating
0.0 - 0.1	None
0.1 - 3.9	Low
4.0 - 6.9	Medium
7.0 - 8.9	High
9.0 - 10.0	Critical

Table 2.1: CVSSv3 Severity Levels

2.4 Applicability of our Application

So far, we have elaborated on the core benefits of microservices and the use of containers, as well as examined related issues regarding security. Further, we have seen that there are currently available tools such as Snyk that address these issues, but these tools have certain aspects that limit their usefulness to our employer. Telenor's expansion into cloud computing services has made this apparent to them, and these limitations lead to them forming the subject of this thesis. In this section, we will first elaborate on what these issues are, before concluding this chapter with how our application aims to address them.

2.4.1 Issues with the Current Solution

As previously outlined, one of the main benefits a microservice architecture brings, is the added agility with which its developers can operate. The ability to rapidly make changes and push these to production environments, is a radical new way of doing what used to be a comparatively slow and cumbersome process. Naturally, the ease of which changes can be made under such agile development models, allows for the possibility of more frequent mistakes. Moreover, this highlights the need for fast and thorough automatic testing frameworks to be included in the pipelines of developers. Ensuring that any change made to a service, will not result in the loss of its functionality. This automatic testing framework should ideally include the use of container security scanning tools to ensure equally high standards being applied to security, but the overly extensive output the tools produce unfortunately makes this a sub-optimal solution.

Another issue with the current solution, is its inability to benchmark the security state of an environment over time. Part of the agile development model is to constantly review and improve on your practices, and this should once again, include security as well. The inconsistent structure of the output, combined with the manual input required to log the results, gives a confusing picture of how the developers are performing over time regarding the security of their environment.

Due to the ever changing landscape of vulnerabilities and exploits, this will still be a difficult issue to fix, but we certainly hope our software will make this aspect easier.

Lastly, the extensive nature of the scans' output makes it difficult to comprehend by personnel without specialised technical knowledge. The vast majority of the information it gives is useless to most people, even developers. The need to highlight what is vulnerable, and what can be done to fix it is therefore a prevalent issue, and needs to be effectively conveyed to more than a few people with specialised knowledge within our employer's organisation.

2.4.2 How Our Application Solves These Issues

First and foremost, our application needs to be able to parse output from a container security scanning tool, and highlight the most important information they provide in a concise way. This will allow us to model the data, meaning we can re-shape it to into figures and other easily presentable statistics, thereby eliminating the issue of having only specialised personnel being able to understand and use the output. We will also need to implement a persistent logging service to help with benchmarking performance over time, and allow personnel to review trends in their environment's security. Finally, we hope to to make the whole process of producing report fully automated, so it can be included in a CI/CD Pipeline, as well as triggered on a fixed time schedule. The details of how we intend to achieve these things will be further explained in Chapters 4, 5 and 7.

Chapter 3

Related Work

The aim of this chapter is to explore different alternatives to our application, in order to evaluate its originality. There may be other solutions available with a similar purpose, or solutions that are not publicly available.

Our research has revealed no service, neither commercial nor open-source that offers the same functionality as our application. That is, a service that parses the output of container scanning tools and compiles a PDF highlighting the most important information. This does not mean that there are no private solutions that achieve the same outcome, but since we can not verify their existence, we will instead focus on what is available.

The closest thing we have found is Snyk's own web based solution, Snyk Code¹. Like our application, it highlights the most severe vulnerabilities, but unlike our application allows users to interactively hide, group and sort vulnerabilities by certain categories.

Snyk Code has its own scoring system which they call *Priority Score*, as shown in the information popup in Figure 3.1. The Priority Score is based on multiple criteria, the discussion of which is outside the scope of this thesis. The purpose of the Priority Score is clear however, to give the context of why a vulnerability is more severe, and therefore posing a greater risk, than another severity.

What is lacking in this solution however, is a statistical overview of the most important information. It simply shows each vulnerability's name, a short description, and its corresponding severity score. This means that any details the user wishes to extract beyond the aforementioned ones, requires expanding each vulnerability and read through a large amount of information. Figure 3.2 shows a screen capture of how the web based solution showcases the vulnerabilities found.

¹Snyk Code: <https://snyk.io/product/snyk-code/>

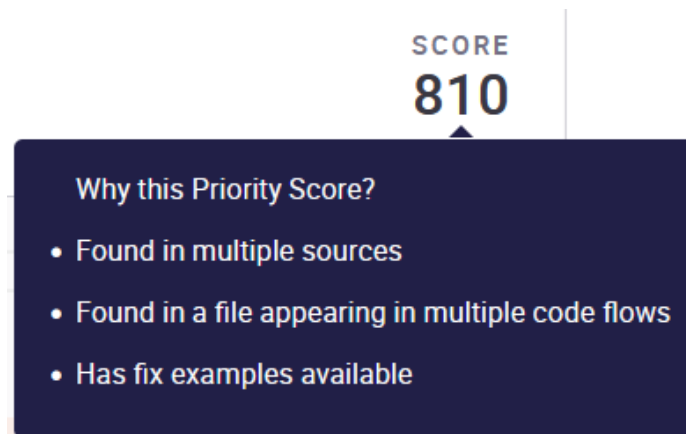


Figure 3.1: Some of the reasons for how Snyk calculates Priority Score



Figure 3.2: An example of how Snyk Code presents found vulnerabilities

There are more key ways in which our application differs from Snyk’s web based solution. One of which is persistent storage, and the ability to query the information on demand. Because of the latter, users will be able to use past data to identify trends in how the security level of their environment is evolving over time. Additionally, if a user wants more information on something other than what can be discerned from the PDF report our application generates, they will also be able to query the raw scanning output in a JSON format.

Chapter 4

Requirements

This chapter will define the requirements for the application. The requirements are divided into functional and non-functional requirements that each specify different aspects of the application. Functional requirements are typically related to the technical details of a system and its components [13]. These requirements help define how the system should behave and function. On the other hand, non-functional requirements are complementary to functional requirements, and they are used to define the criteria of how a system should perform in terms of quality [14].

It is important to note that the leading design principle of our application is to be Cloud Native. Cloud native means that the application is developed first and foremost with a cloud platform deployment target in mind, and all the infrastructure choices that brings with it.[15]. Cloud platforms bring numerous advantages in the form of improved availability and resiliency, but also removes some agency on our part for ensuring the service is available. Both our functional and non-functional requirements are thus directly affected by the guarantees of our cloud platform provider's Service Level Agreement (SLA).

4.1 Functional Requirements

The following list shows all the functional requirements of Auspex.

- F-1 Auspex should scan container images in a connected Container Registry.
 - Auspex needs to be connected to an existing Container Registry that contains all the container images Auspex can scan. Auspex does not have an internal Container Registry itself, but is connected to the Con-

tainer Registry of the environment Auspex pulls the container images from.

- F-2 Auspex's Scanner microservice should perform a vulnerability scan on the given container images and output a JSON formatted response body.
 - Given that we structure our application components as multiple microservices, one microservice should be able to pass information to another microservice, without interfering or knowing each other's internal functionality and inner workings. Passing the data in a common and predictable format allows the data to be used in different ways, but remain unchanged.
- F-3 Auspex's Reporter microservice should transform JSON formatted responses into human-readable PDF reports detailing vulnerabilities for a given container image.
 - The report that Auspex generates is at the core of the application. This report is generated based on data from a scan of a container image and used to highlight and convey key information about vulnerable container images in a human-readable presentation.

4.2 Non-Functional Requirements

The following list shows all the non-functional requirements of Auspex.

- N-1 Auspex should be as available, as the guarantees given by the cloud platform provider.
 - Auspex will be deployed on Google Cloud Platform, and is thus tightly connected with the availability of GCP and its services. The availability is defined as a Service Level Objective (SLO) in the SLA, outlining the objectives for the service, such as uptime, latency and error rate [16]. The main services that is used by Auspex are Cloud Run and Cloud Storage, and both define their Monthly Uptime Percentage as 99.95%¹². To achieve this requirement, our application needs to be Cloud Native so that the aforementioned SLO terms are guaranteed.
- N-2 Auspex must ensure confidentiality and privacy.
 - Since Auspex deals with the security of our employer's container image environment, it is of vital importance that our application does not disclose any sensitive information to potential attackers. To this end, we strive to design the application to be Cloud Native, which in turn

¹<https://cloud.google.com/run/sla>

²<https://cloud.google.com/storage/sla>

will allow our employer to utilize the many different security options provided by cloud platforms.

N-3 Auspex reports must show all vulnerabilities detected in a scan.

- The generated reports must show all vulnerabilities detected from a scan of container images. It is vitally important that our report does not omit any information, as this will undermine the credibility of Auspex's reports.

N-4 Auspex must produce readable and understandable reports.

- The Auspex report needs to be human-readable and understandable for personnel within Telenor with varying levels of expertise. Therefore, we need to ensure a minimal threshold for readability, by omitting unnecessary information, and highlighting more important elements.

Chapter 5

System Design

This chapter introduces the microservices of *Auspex* and its main components. It then goes further into explaining the main components in detail and their function.

Clarifications

The name of our application is *Auspex*. The origin of the name comes from ancient Rome. In ancient Rome, an *Auspex* was a priest responsible for interpreting omens and signs to guide decisions. We chose this name due to the similarities between priests interpreting omens and our software uncovering vulnerabilities to warn of potential future disasters.

For the purposes of this thesis we use 'artifact' to describe the output of a process, intermediate or otherwise. An artifact can be the data produced by a tool, the output of a microservice or the end-result of a pipeline or process. Wherever the specific contents or format of an output is not relevant, we will use the term artifact, but otherwise we will use a more descriptive name where applicable.

5.1 Application Overview

To better understand the application, one should first know its functionality. Figure 5.1 shows an overview of the microservices, illustrated as blue hexagons, that deliver the functionality of *Auspex*. The yellow cylinders represent Firestore databases, and the grey circles represent a storage bucket. Firestore is an organized, cloud-based NoSQL database hosted by Google where data is stored in documents that contain key-value pairs, and each key is automatically indexed. These doc-

uments are stored in collections, similar to how a row is stored in a table in a SQL database. Collections are used to organize documents and enables clients to query and filter documents based on these collections [17]. Buckets are used to store artifacts produced by the different microservices. A bucket is a cloud-based storage solution that allows clients to store unstructured data [18]. More details on the implementation of these microservices can be found later in this chapter. Note that Auspex can also take a request to scan multiple container images at once, and create an *aggregate report* from the output. For simplicity however, Figure 5.1 shows a scan on a single image which creates what we call a *single report*, with the corresponding steps listed below.

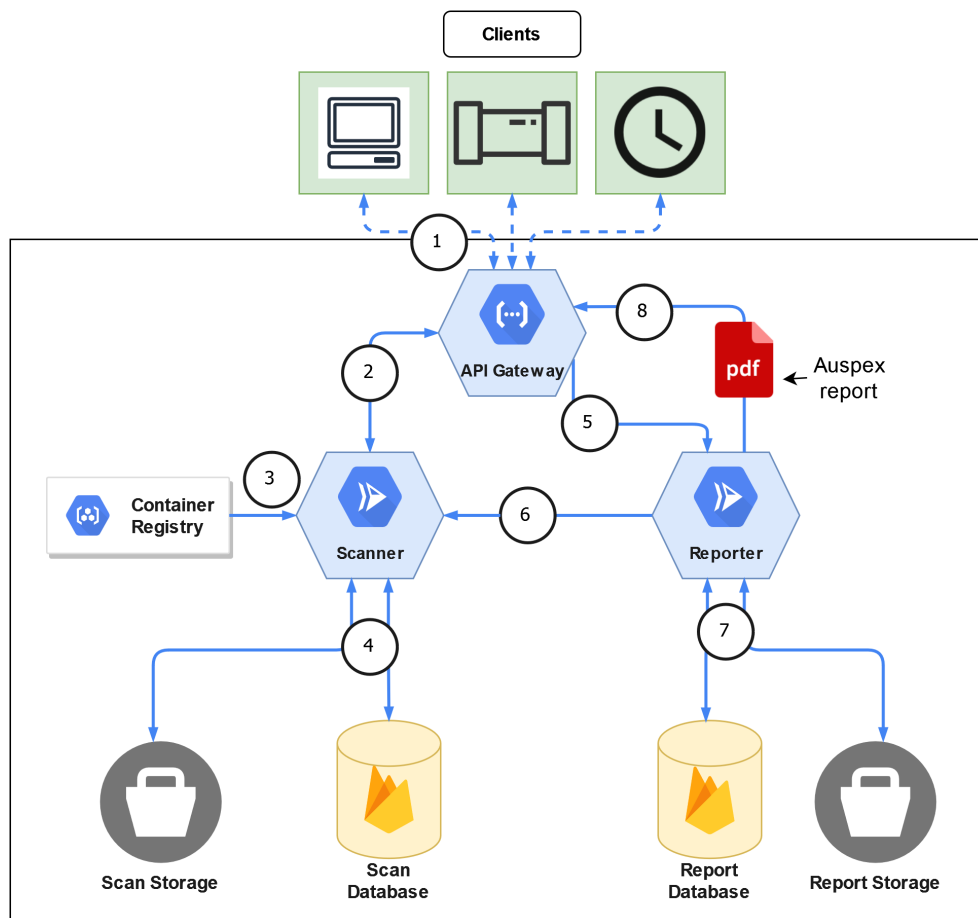


Figure 5.1: Overview of all the components and microservices of Auspex

1. The process starts with a request issued to the *API Gateway*. The API Gateway is a REST API that takes incoming requests via its available endpoints. See Section 7.4.1 for more details. The request is primarily issued on a schedule, or by a CI/CD Pipeline. Additionally, a scan can be triggered manually by a user.

2. The API Gateway issues a request to *Scanner* for a container image to be scanned.
3. The requested image is pulled from the connected *Container Registry*, which is a repository of container images. The container image is then subsequently scanned by *Scanner*, using *Snyk* as the scanning tool, which is integrated into our application, *Auspex*. In short, *Snyk* goes through all the dependencies and libraries of the container image, and checks for vulnerabilities or other issues.
4. The full information output from *Scanner* is collected and sent to *Scan Storage*, as a JSON formatted file, where it is stored. A snippet of this information is stored in *Scan Database* with a reference and link to the full output in the *Scan Storage* bucket.
5. The API Gateway issues a request to *Reporter* for a report to be made of the container image that has been scanned.
6. *Reporter* requests the scan data produced by *Scanner* in order to produce its artifact, an *Auspex report*, which is a readable PDF document of the findings from the scanned image. The document shows key information about relevant security metrics for the detected vulnerabilities of a scanned container image.
7. The created artifact is stored in the *Report Storage* bucket, and a reference to this artifact as well as metadata such as time, date and scores is stored in the *Report Database*.
8. The *Auspex* PDF report is returned in the response to the caller.

5.2 Detailed Component Description

Auspex consists of multiple components as illustrated in Figure 5.1. The purpose of these components working together, is to take in a request for an image or multiple images to be scanned, and pass the request over to a specially created cloud function which will analyze the image or images. The output from this scan will both be stored in a raw format, and also used to generate a report in a human-readable format. This report will also be stored. The subsequent subsections will go into further detail on the components of the application.

5.2.1 Clients and API Gateway

We have defined three types of clients that can invoke requests to the application, by a *pipeline trigger*, on a *schedule* or manually by a *user*. The client invokes a request for a scan, either manually by a user, automatically on a schedule or as a part of a pipeline. The main scenario is an automatic scan triggered on a schedule, or by an event, such as a change or update to an existing container image. The

requests are issued to the API Gateway which provides a limited number of endpoints for interaction with the microservices of Auspex and is hosted as a Cloud Run function. The endpoints allows for scanning of a single container image as well as scanning multiple container images. In addition there are endpoints for querying stored information such as previous scans and reports which is available through the Scanner and Reporter microservices. See Section 7.4.1 for more detail on the endpoints.

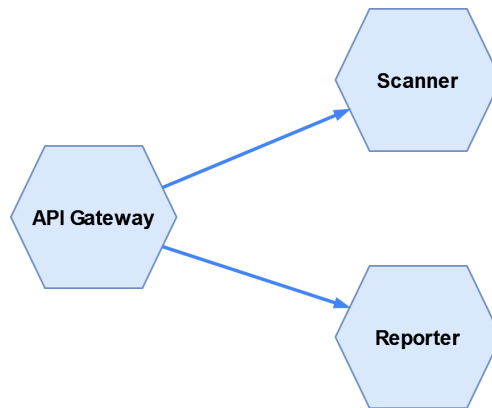


Figure 5.2: API Gateway communicates with Scanner and Reporter

The purpose of Auspex is to have it automatically scan container images, not requiring user input or interaction. Manual operations takes time and can be inconsistent. Removing and automating manual processes frees up time and resources for other tasks and work, as well as ensuring consistency for operations.

A pipeline trigger or a schedule can be setup to achieve automation by integrating a CI/CD Pipeline trigger for Auspex in the Git repository¹ for an application which builds Docker containers, defined in Dockerfiles, as part of its deployment. When new code is pushed to the source code repository and a new deployment is started, Auspex can be triggered to scan the Dockerfiles and create a report which can be sent via email to the developers of the application.

The scheduled scan is what we determine to be one of the main use cases of Auspex. Auspex can be set up to scan a connected Container Registry on a schedule. The schedule can be freely determined by the user, or an organization, depending on what is appropriate for their application. Factors such as frequency of builds, the security requirements for the containers and personal preferences can influence the intervals for scans.

Scans can be triggered manually if needed by a user. This is not one of the main intended use cases, but it is accommodated for to allow for testing purposes, and

¹Most other code repositories offers CI/CD Pipelines, but for this project we only consider Git

querying for scans and reports in storage. The endpoints are the same as those the microservices Scanner and Reporter use, but of course requires manual input by a user. See Section 7.4.1 for details on how the endpoints are constructed and the options available.

5.2.2 Scanner

The Scanner microservice is deployed on Cloud Run and takes in requests for a scan. A request may invoke a single image to be scanned or multiple images to be scanned. The requested image, or images, respective Dockerfiles will be scanned and generate an output of information. The output of the scan is returned as a JSON encoded string in the response payload. The Scanner service itself is a Docker container based on the Python 3.10 slim image, with theSnyk CLI installed as well. The Snyk CLI is used to invoke commands to scan for security vulnerabilities [19]. Scanner uses the `docker scan` command to scan the Dockerfile of an image for vulnerabilities. Scanner relies on a Container Registry to function. Container Registry stores the container images with their respective Dockerfiles. An incoming request will call for a specific image or multiple images that is subsequently pulled from Container Registry and scanned.

The information artifacts produced by Scanner is organized and stored, and may also be sent to the Reporter microservice if such a request was issued via the API Gateway. The full information artifacts from the scan is stored as a JSON file in the *Scan Storage* bucket, and select data is stored as metadata in the *Scan Database*. The metadata is stored in a Firestore database. Due to limitations on the file sizes in Firestore, all the data from the scan cannot be stored in such a database. To circumvent this we generate a link to the matching JSON file in the storage bucket, the complete scan, and store this link in the metadata. The schema for the metadata can be seen in Figure 5.3. The purpose of storing and keeping the scan data is to use it later on, for example to compare trends over time, and thus be able to assert if security state of a given container image is improving or worsening over time.

5.2.3 Reporter

Once a scan is complete, the output from Scanner may be sent to Reporter which is also deployed on Cloud Run. Reporter is a Docker container based on the `texlive/texlive`², with Python3.9 and Matplotlib installed. TeX Live is a distribution of the TeX typesetting system, which includes LaTeX [20]. Thus, the tools within TeX live, is used to typeset our generated reports, using the same typesetting as this doc-

²<https://hub.docker.com/r/texlive/texlive>

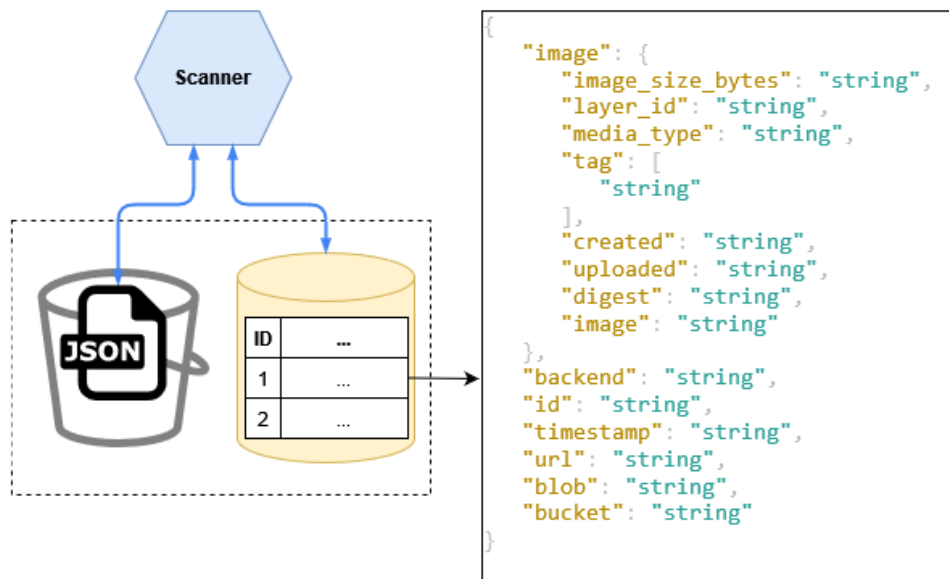


Figure 5.3: Simple illustration of Scanner storage including the schema for Scan Database

ument. Matplotlib is used for mathematical typesetting and creating graphs and plots with Python [21]. This is all combined using PyLaTeX, an interface between Python and LaTeX for creating and compiling LaTeX files [22]. The end result is a report document showcasing how many vulnerabilities are found, their severity and distribution and more.

The storage of the artifacts produced by Reporter is very similarly organized as Scanner, however the artifacts stored are very different. By default we are preferring to store the artifacts produced by Reporter as PDF documents in an unorganized bucket. This is due to the same file size constraint as describe in Section 5.2.2. In addition to the storage bucket for documents, the metadata of the documents is stored in a database. This metadata includes such data as an ID for the database entry, time and date of the report and a link to the corresponding document stored in the bucket. In Figure 5.4, part of the schema for the metadata Reporter stores can be seen.

5.2.4 Automated Report Creation

In order for Auspex to work autonomously, we envision the application to run on a schedule, which can be facilitated by scheduling and automation tools such as

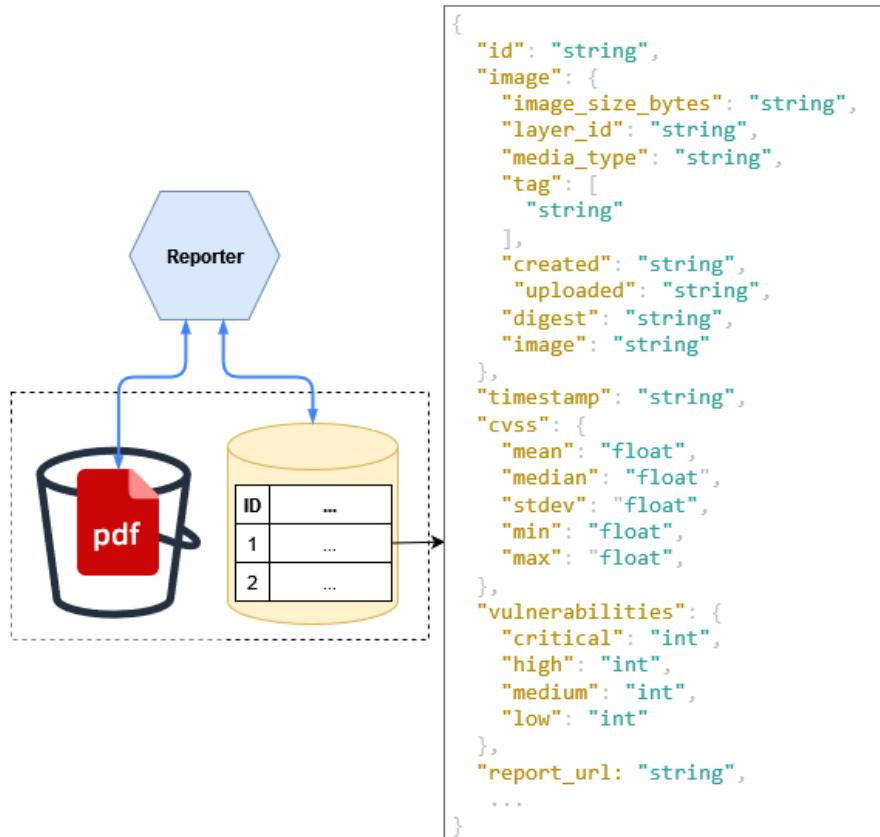


Figure 5.4: Simple illustration of Reporter storage with schema. (Some keys are omitted to ease presentation of the schema.)

Google Cloud Scheduler³ and Google Cloud Workflows⁴

The Email Service is setup to receive a document from the Reporter microservice and deliver this document via email. The Email Service works hand in hand with the automated scanning capabilities of Auspex. For example when the scheduled workflow for Scanner scans a container image and Reporter creates the corresponding Auspex report, The service takes in the finished report from Reporter and uses a predefined list of recipients to send the report via email. The Email Service is implemented as a Google Cloud Workflow workflow, with predefined steps.

³<https://cloud.google.com/scheduler>

⁴<https://cloud.google.com/scheduler>

Chapter 6

Development Process

This chapter gives an overview of the development process and work methods for the application and the bachelor thesis. In addition we also describe how we documented our work as well as our routines.

6.1 Development Model

Choosing the most suitable development model for this project proved challenging. We discussed multiple different models that might have been appropriate for our work. We needed a development model that allowed for quick, frequent iterations and allowed us to quickly respond to changes in scope or requirements based on feedback from our supervisor, our employer or ourselves as we progressed. The Agile methodology as a whole focuses on continuous development and working in iterations when developing software [23]. As discussed in our Project Plan section 4.1 Development Model (ref to appendix), we decided against the rigid Waterfall model [24] and the Agile Lean methodology [25]. Instead we decided for the Agile methodology of Kanban [23].

Kanban is a more flexible, continuous approach to development, utilizing cards organized on a Kanban board to define work items [26]. The cards define the tasks that must be completed and, optionally, their priority. This approach allows for new tasks to be added continuously, while also allowing for changes in tasks at any time.

We decided Kanban was a suitable method for our approach and what we were trying to achieve with our work, working iteratively. This varied somewhat from how we defined our approach in our Project Plan, a combination of Scrum and Kanban, but we found out very quickly that Kanban was better suited for us and

allowed for a less rigid work structure. We found that it was difficult to have all the necessary requirements ready and defined before the sprint start. As described previously, we wanted an approach that allowed us to work iteratively, which allowed for changes in scope and requirements without being constrained to assumptions and processes defined early in the project. Despite this flexibility, we needed to approach this in a structured manner. Thus, we found it appropriate to follow the Kanban approach.

We started to utilize a Kanban board, Trello¹, to define our task and keeping track of them. Later on we decided to switch to using the built-in issue board on GitLab². The issue board helped us keep track of the tasks that needed to be worked on.

6.2 Documentation

Various documentation has been produced throughout this project. This chapter gives an overview of the main pieces of documentation and the work routines used to achieve this work.

6.2.1 Kanban Board

We utilized a Kanban board for the purpose of keeping track of all the various tasks that needed to be completed, what was being worked on, and which tasks that had been completed. This was separated into the main categories *TODO*, *DOING* and *DONE*. A new task would start in *TODO* where a title, and optionally a description, would be added, as well as relevant labels for the task. A task would then be moved to the *DOING* category when it was being worked on, and finally to the *DONE* category when completed.

As previously described, we started by using Trello as our Kanban board and later switched to using the built-in issue board in our GitLab repository. We transferred all the cards that we had on Trello over to GitLab. Then we split the task into two issue boards, one for tasks related to writing the thesis, and one for application related tasks.

¹<https://trello.com/>

²https://docs.gitlab.com/ee/user/project/issue_board.html

6.2.2 Thesis Writing

This thesis was written using the online LaTeX³ editor Overleaf⁴ which allowed writing the thesis collaboratively. Using Overleaf ensured that everyone worked on the correct version of the thesis at all times. Furthermore, it ensured consistent typesetting and formatting across the entire document. In addition, we used a LaTeX thesis template created by Community of Practice for Computer Science Education (CoPCSE) at NTNU⁵ available free of charge under an open license.

6.2.3 Source Code

The source code for the application is kept in a online GitLab⁶ repository hosted by NTNU. We used the built-in Issue tracker in GitLab as our Kanban board. Git takes care of versioning and also serves as a backup for the source code. Additionally, the source code is also hosted in GitHub to ensure that it is available to Telenor, see Chapter 1

6.2.4 Other Documentation

Documentation written over the duration of the project, such as notes, status reports, minutes of meeting and any other documents has mostly been written in Google Docs and stored in a shared Google Drive folder. The shared Google Drive folder is only accessible to the members of the team. No sensitive information was stored in the folder, as per guidelines received by our employer.

Other documentation such as technical documentation and user manual was written both manually and created automatically from our source code. MkDocs, a static site generator for creating project documentation [27], was used to create a website on GitLab for technical documentation. The user manual was written manually and features guides and tutorial on how to use the application and how it works.

6.2.5 Meetings and Minutes of Meeting

We had semi-regular weekly meetings with our supervisor each Wednesday from 1pm to 1:30pm. Beforehand we would hand in a weekly status report outlining

³<https://www.latex-project.org/>

⁴<https://www.overleaf.com/>

⁵<https://github.com/COPCSE-NTNU/thesis-NTNU>

⁶<https://about.gitlab.com/>

the work we had done, if there were any issues or problems encountered, any questions that we might have, as well as our plans for the coming week. The purpose of these meeting was to review our progress.

We had semi-regular meetings with our employer, the purpose of which was to keep them updated on our progress and clarifying any questions we might have regarding our work. In addition to this we also conducted communication through Slack, a desktop messaging application⁷, for various questions we had that did not require a meeting of its own.

In addition to these meetings, Eirik arranged multiple private presentations for us that gave us greater insight on how Telenor works with, and approaches development and operations (DevOps) and related practices. We also received a behind-the-scenes look at the development of Mitt Spor, a service and device for tracking pets [2], and how they use GCP to organize and run the microservices and workflows. The presentation gave us an interesting insight in to how professionals approach development and some inspiration for our own application.

For each meeting, both with our supervisor and with our employer, we wrote minutes of meeting and took personal notes. The purpose of minutes of meeting is to act as a summary of our meetings and to be a reference for any feedback we receive. These documents can be found in Appendix C. The only exception from this was the private presentations Eirik arranged, for which we needed to sign a confidentiality agreement, which can be seen in Appendix A.

6.2.6 Time Tracking

For tracking time we used Clockify⁸, an online time tracker and timesheet app. All time spent on the project was logged by each team member individually. The total time spent on this project can be viewed in Appendix A

6.2.7 Expenses

For this project we did not anticipate any notable expenses, and any incurred cost would have been split equally amongst ourselves. As Telenor graciously gave us access to their Google Cloud Platform plan, no costs were incurred on our part for cloud services.

⁷<https://slack.com/>

⁸<https://clockify.me/>

6.3 Routines

6.3.1 Work Policies

As part of the project we wrote a Project Plan outlining the expectations for every team member in regards to work, both quality and quantity. These are guidelines set by us in common agreement. Each team member has been expected to contribute 30 hours per week and log this in Clockify.

6.3.2 Tools

We used an agreed upon set of tools for work and communication. Adaptations or changes in these has been permitted if we found the tools not beneficial or that other tools might be more beneficial. We also sought to be flexible in regards to adapting to Telenor's tool set.

6.3.3 Communication

We were in a unique situation where all three of us lived together and this of course influenced our approach to the work and our routines. This was also beneficial for our inter-group communication as we could have discussions around the dinner table and work tightly together. For external communication we used email for the introductory communication and used this to establish communication channels on Microsoft Teams, Slack and Google Meet, the latter two preferred by Telenor.

Chapter 7

Implementation

This chapter goes in depth in the technical implementation and details of Auspex.

7.1 Repository Structure

Traditionally, a repository was used to store the source code for a single application along with its documentation and tests. This made sense with regards to monolithic applications that are self-contained. However, as developers increasingly moved towards microservice architectures, the need to rethink the structure of source code repositories arose.

We believe this to be one of the key reasons for the emergence of *monorepos*. A monorepo is a single monolithic repository that contains the source code of *all* the microservices that define a modern application built on a microservice architecture. One of the main benefits of a monorepo is that multiple applications can share certain internal libraries or data structures that are located in a "core" or "shared" directory in the repository. This pattern reduces code duplication and bloat when building microservices, and ensures that developers spend less time writing the same code repeatedly.

7.1.1 Monorepo and Microservices: an Anti-pattern?

Microservice architectures emphasize loose coupling, which means the the state of one service should not affect the state of another service. Code changes in one service should therefore not affect other services in the application. The concept of a core directory that is used by multiple different microservices goes against

this idea. Modifying a library or data structure in the core directory could affect multiple services, not just one - thereby violating the principle of loose coupling.

This is a valid concern, and on the surface it does indeed affect the degree to which each microservice is self-contained. However, what the core directory contains is highly relevant with regards to judging its impact on the overall architecture.

In order to reduce the impact on the application as a whole due to changes to the core directory, it should contain no business logic; business logic should be confined to the microservices themselves. The core module should only be concerned with defining certain shared data structures and utility functions. In effect, the core directory becomes a vendored dependency of general-purpose code.

The core module can be used to share generalized utility functions that may be used by multiple microservices. An example of such a utility function is a function that retrieves a document from a NoSQL database and performs rudimentary error checking and logging as well as providing retry on failure with exponential backoff. Given that multiple microservices will be using a NoSQL database, having access to such a function in every microservice frees developers up from having to redefine or copy functions that already exist in other microservices.

```
@backoff.on_exception(
    backoff.expo,
    exception=aiohttp.ClientResponseError,
    max_tries=5,
    jitter=backoff.full_jitter,
)
async def get_document(collection_name: str, document_id: str) ->
    DocumentSnapshot:
    db = get_firestore_client()
    # Get firestore document
    docpath = f"{collection_name}/{document_id}"
    logger.debug(f"Fetching {docpath}")
    d = db.document(docpath)
    doc = await d.get() # type: DocumentSnapshot
    if not doc.exists:
        raise ValueError(f"Document '{document_id}' not found.")
    return doc
```

Listing 1: Shared document retrieval function with backoff

The code snippet above demonstrates the general-purpose nature of a function defined in the core module. The function makes no assumptions about the shape

or size of the data; it only checks if the document exists, and then subsequently returns it if it does. In the event of an HTTP error with a code of 400 or above, the function retries up to 5 additional times, thereby ensuring a degree of protection against intermittent downtime on the cloud platform's part.

7.1.2 Choosing a Repository Structure

Most of our microservices will be interacting with Google Cloud Platform APIs in some capacity, and each service will use Google Firestore for database purposes. Furthermore, inter-service communication is facilitated by HTTP requests with JSON-encoded payloads. In order to validate data going in and out of the services, the services will need to define a range of shared data structures modeling this input/output. Given that we are not using Protocol Buffers¹ to define these data structures across services, we need to do this in-code with native Python data structures (classes). To share these definitions across all services, each service must have access to this code through an importable Python module.

For the reasons stated, the need to share data structures and interface definitions between multiple services, we will be using a monorepo repository structure for our application. The benefits of sharing certain code between services, as well as having all of our application code in a single repository will greatly simplify the development process, and reduces duplication of data structures. (utvid litt mer her)

7.1.3 Implementing Monorepo Builds With Docker and Poetry

Poetry² is a Python dependency manager and packaging tool that is rapidly gaining popularity in the Python world. Poetry makes it easy to work with a monorepo structure with its built-in support for local, editable dependencies for its projects.

Poetry is not native to Python, and must thus be installed in each Docker image to export its dependencies to the native Python *requirements.txt* format. In order to not ship Poetry with the final Docker image, as well as separating the step of exporting dependencies from the step of installing the application, we used Docker multi-stage builds³. Additionally, we cached the installation of Python dependencies in order to speed up subsequent builds.

¹<https://developers.google.com/protocol-buffers>

²<https://python-poetry.org/>

³<https://docs.docker.com/develop/develop-images/multistage-build/>

7.2 Code Quality and Formatting

Our application code should have a consistent quality throughout all its source code. One aspect of code quality is its correctness, meaning the degree to which it provably does what it should. Another aspect is how readable the code is. This includes the use of consistent formatting across all files, as well as following best practices with regards to writing idiomatic Python code [28].

7.2.1 Mypy

Python is a dynamically typed language. In essence, it means that variables and function parameters can take on any type at runtime. This allows for a great deal of developer freedom, but also introduces the possibility of type errors, where a function or method receives an argument of a type it is unable to handle.

Keeping track of dynamic types in larger projects adds a lot of mental overhead to the development process, and thinking one can avoid all type errors is hubris in line with C/C++ programmers who think they can avoid all memory errors [29] [30].

To rectify this potentially problematic aspect of the Python language, we used mypy⁴, which introduces optional static typing to Python. Mypy runs static analysis on Python code that has type annotations⁵. By adding type annotations to the code, mypy verifies the correctness of the code and warns of possible type errors. This process, however, does not uncover potential logical errors or otherwise incorrect implementations of features.

7.2.2 Black

To ensure consistent formatting of all Python source code, we used the formatting tool Black⁶. Black is self-described as "uncompromising", which we find to be apt given its lack of configuration options. Black produces consistently formatted files across every project it is applied to. Given its ubiquity in the Python ecosystem, we decided to use it as our one and only Python formatter. We configured Black to run client-side on every file we saved in our IDE, Visual Studio Code, but it can also be set up to run as CI hook as well.

⁴<http://mypy-lang.org/>

⁵<https://docs.python.org/3/library/typing.html>

⁶<https://github.com/psf/black>

7.3 Data Validation With Pydantic

In order to parse, validate, serialize and deserialize the different data structures used in the project, we chose to make Pydantic⁷ a central part of all our services. Pydantic provides data validation and settings management using python type annotations, and enforces these at runtime.

```
from pydantic import BaseModel

class OrgLicenseRule(BaseModel):
    licenseType: str
    severity: str
    instructions: str

class LicensesPolicy(BaseModel):
    severities: dict[str, Any] # unknown contents; thus Any
    orgLicenseRules: dict[str, OrgLicenseRule] # key: License
    ↪ name
```

Listing 2: Example of Pydantic usage showing composition of models

Listing 2 shows how Pydantic can be used to define data structures that incoming data must comply with. The example also demonstrates how models can utilize composition to include other models.

Pydantic is able to parse and validate any data that can be expressed in a JSON schema, and as such is a perfect fit for the data we are working with. Furthermore, by defining constraints we can ensure the data only includes values that is valid; for example making sure the value of a CVSSv3 score is between 0 and 10.

Names of fields in Pydantic models must match the name of the data passed in, and vice versa. This is the reason we used camelCase casing for the field names. Pydantic does support converting between snake_case and camelCase, but we decided to follow Snyk key names 1:1, which are all in camelCase.

Throughout this chapter, examples will frequently include data structures created with Pydantic, which is signified by inheriting from `BaseModel`.

⁷<https://pydantic-docs.helpmanual.io/>

7.4 Service: Scanner

The Scanner service is a FastAPI⁸ application running on a Uvicorn⁹ web server inside a Docker container based on the `python:3.10.4` image. The application handles incoming HTTP requests and initiates actions based on the contents of the request. The primary use case of the service is to provide an interface for scanning container images remotely and storing the results in a database. Scanner performs no analysis of the results, and only verifies that the scanning proceeded without any errors. It is up to other services to use the results produced by Scanner for further processing. The service also provides an interface for retrieving previously completed scans.

The service runs as a Cloud Run service configured to handle at most 1 connection per container, meaning each scan request spins up a new container. The reason for this is twofold; it is mainly done to reduce the latency associated with scanning multiple images, ensuring that congestion is not a problem. However a secondary reason is also to avoid out-of-memory errors that could occur as a result of scanning multiple images in parallel on a system that uses an in-memory filesystem, such as Cloud Run. In these situations, downloading and scanning large images takes up a sizable portion of the container's memory. Reducing concurrency to 1 connection per container solves this.

7.4.1 Endpoints

The scanner service exposes its functionality as a range of REST API endpoints. The main purpose of the endpoints is to facilitate creation and retrieval of scans. Figure 7.1 shows the REST API endpoints exposed by the Scanner service.

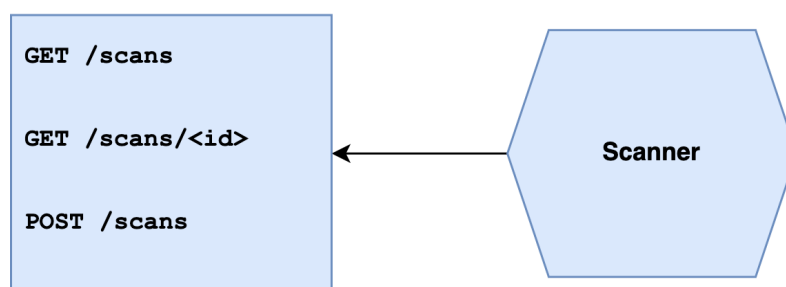


Figure 7.1: API endpoints exposed by the Scanner service

⁸<https://fastapi.tiangolo.com/>

⁹<https://www.uvicorn.org/>

GET /scans

Retrieves all scans, with the option to only return scans for a specific image.

Returns an array of objects following the schema described in appendix B.2.

GET /scans/<id>

Retrieves a scan with the specific ID. Intended to be used by other services to fetch scans for further processing.

See Appendix B.2 for response schema.

POST /scans

Initiates scanning of an image with the desired scanning backend. At the time of writing, Snyk is the only supported scanning backend.

Scan request bodies have the following schema:

```
{
  "image": "string",
  "backend": "snyk"
}
```

Listing 3: Schema for POST /scans request body

See Appendix B.2 for response schema.

GET /status

Returns the status of the service. The status is determined by the service's ability to contact its own database and bucket.

The endpoint returns data with the following schema:

```
{
  "status": "OK",
  "message": "string",
  "url": "string"
}
```

Listing 4: Schema for GET /status response body

7.4.2 Running Snyk In A Container

Snyk is available both as an online SaaS solution¹⁰ and as a standalone CLI tool¹¹. The SaaS version features a wide range of service integrations, from GitHub and GitLab, to Google Container Registry. It has one fatal shortcoming though; it does not produce a structured data file compiling the results of the scan which can be used by our application. All the information is presented in Snyk's web interface, but cannot otherwise be downloaded.

As a result of this, we decided on using the Snyk CLI. The Snyk CLI performs the same scanning operation that the SaaS version does, but additionally has the capability of exporting its data as both JSON¹² and SARIF¹³. Due to the ubiquity of JSON parsing tools, as well as having more experience with JSON than with SARIF, we ended up selecting to use the JSON output option of Snyk CLI.

Although Snyk provides an official Snyk CLI Docker image¹⁴ (`snyk/snyk-cli`), we found it easier to base the Scanner service's image on the official 'python' Docker image¹⁵. Installing the Snyk CLI is as simple as downloading a file, making it executable and moving it to a directory on the system's PATH. Meanwhile, installing all the the various Python packages requires somewhat more work. To that end, we decided to use the `python:3.10.4` image for the Scanner service.

In the code, Snyk is executed as a Python subprocess. The Snyk CLI has no Python ABI, and must thus be executed as a normal program via the shell.

¹⁰<https://snyk.io/>

¹¹<https://docs.snyk.io/snyk-cli>

¹²<https://www.json.org/json-en.html>

¹³<https://docs.oasis-open.org/sarif/sarif/v2.1.0/csprd01/sarif-v2.1.0-csprd01.html>

¹⁴<https://hub.docker.com/r/snyk/snyk-cli>

¹⁵https://hub.docker.com/_/python

```

def run_snyk_scan(image: str) -> SnykScanResults:
    snyk_exe = get_snyk_exe()

    # We use the --json option to pipe the results to stdout
    # and then read it directly into memory.
    snyk_cmd = f'{snyk_exe} container test --json
    ↪ --username=_json_key --password="$(cat
    ↪ {AppConfig().google_credentials})" {image}'
    logger.debug(f"Running snyk command: {snyk_cmd}")
    p = subprocess.run(
        ["/bin/bash", "-c", snyk_cmd],
        capture_output=True,
        text=True,
    )
    return SnykScanResults.from_subprocess(p)

```

Listing 5: Running Snyk as a subprocess with Python

The code snippet above demonstrates how the program initiates a Snyk container scan. In order to authenticate with a container registry hosted on Google Container Registry, Snyk needs to provide a username and password. Google Container Registry supports authenticating with Google Cloud Platform credentials¹⁶ as a JSON file for the password field and `_json_key` as the username.

The code snippet also demonstrates that the program delegates the responsibility of retrieving the account credentials to the shell, instead of reading that information into the program memory. This drastically reduces the likelihood that the program leaks the credentials, as it never processes this data itself.

7.4.3 Storing Scan Results

The result of a completed Snyk scan with the `--json` option is JSON-formatted data printed as text to Stdout¹⁷. See appendix **APPENDIX REF HERE** for the full schema definition. The Python `subprocess.run` function then captures this output and stores it in the program's memory. From here, the program uploads the output to a bucket as a UTF-8 encoded file with a content type of `application/json`. Further, it creates a Firestore document containing the URL of the uploaded file as well as other metadata pertaining to the scan. The data structure for documents is the following:

¹⁶https://cloud.google.com/docs/authentication/getting-started#creating_a_service_account

¹⁷The default file that processes can write to in Unix-like operating systems.

```
class ScanLog(BaseModel):  
    """Model for documents in scanner's collection"""  
  
    image: ImageInfo  
    backend: str # Scanner backend tool used  
    id: str  
    timestamp: datetime  
    url: str # absolute URL of scan file  
    blob: str # scan file blob name  
    bucket: str # scan file bucket name
```

Listing 6: Data structure for documents representing scans in the database

The data structure with the relevant values filled in is then returned to the caller, complete with metadata and a URL for the scan file. This is also the schema that the GET /scans and GET /scans/<id> endpoints follow.

7.5 Service: Reporter

The Reporter service is a FastAPI application running on a Uvicorn web server inside a Docker container based on the `texlive/texlive` image. The application handles incoming HTTP requests and initiates actions based on the contents of the request. The primary use case of the service is to enable users and other services to create PDF reports from the contents of the container image scans performed by the Scanner service.

7.5.1 Endpoints

The reporter provides several endpoints for retrieving and creating reports. Figure 7.2 shows the REST API endpoints the Reporter service exposes.

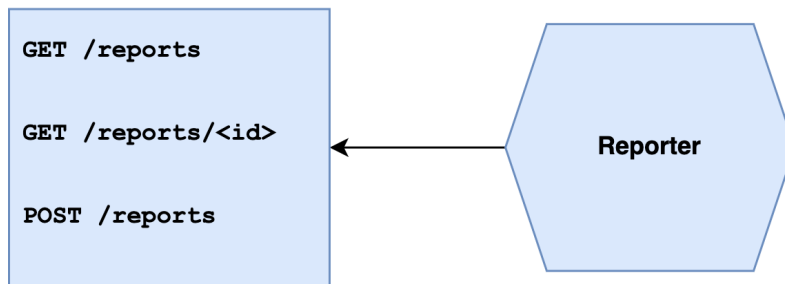


Figure 7.2: API endpoints exposed by the Reporter service

GET /reports

Retrieves existing reports based on a range of filters.

The URL for the endpoint has the following structure:

```
/reports?image=[IMAGE_NAME]
  &aggregate=[true,false]
  &ge=[0-10]
  &le=[0-10]
  &field=[mean,median,stdev,max,min]
  &limit=[>0]
  &order=[newest,oldest]
```

See Appendix B.4 for more information about the query parameters. Otherwise, this documentation can also be accessed on the service's /docs endpoint.

GET /reports/<id>

Retrieves a report by its ID. This endpoint is mostly intended to be used by other services, as users are not expected to know autogenerated report IDs.

POST /reports

Requests the creation of a report given a scan ID and other parameters. See section 7.5.2 for more information about this endpoint.

GET /status

Retrieves the status of the service. The endpoint mainly checks whether or not its own database and buckets are reachable.

7.5.2 Initiating Report Creation

A client initiates the creation of a report by sending a POST request to the service's /reports endpoint. The body of the request must comply with the schema in Listing 7.

```
{
  "aggregate": true,
  "individual": true,
  "ignore_failed": false,
  "format": "latex",
  "trend": true,
  "trend_weeks": 26,
  "scan_ids": [
    "string"
  ]
}
```

Listing 7: Report request schema for the Reporter service

Fields

"aggregate" determines whether or not an aggregate PDF report should be created if two or more scans are provided.

"individual" determines whether or not single reports should be created for the individual scans, or if they should just be parsed and have their results stored in the database without creating an output report. This option is useful if one only wants to create an aggregate report.

"ignore_failed" discards scans with errors and proceeds with the creation of individual and/or aggregate reports for the scans that succeed. By default this is disabled.

"format" determines the output format of the report. As of now, only LaTeX typeset PDF documents are supported, and as such "latex" is the only accepted value for this field.

"trend" controls whether to include the plot showing CVSS mean score trend for the relevant image(s).

"trend_weeks" controls the extent to which the trend plot should show older scans.

"scan_ids" is a list of IDs of scans created by Scanner to parse and create reports for. See section 7.5.2 for more information regarding this field.

Scan IDs vs. Image Names

Users should mainly interact with the API Gateway service, which provides an abstraction over the "scan_ids" field, and provides the ability to use the name of the image that should be scanned and reported, which is a more user-friendly interface in line with the requirements¹⁸. However, Reporter is only concerned with scans that already exist, and as such it takes in specific Scan IDs to parse and report. See section 7.6 for more information about the intended interaction with Auspex as a whole for external clients.

Once Reporter has a list of Scan IDs to retrieve, it queries the Scanner service for information about the scans. The Scanner service returns a JSON data structure containing a list of objects with metadata for each scan. Each scan contains a URL to a bucket which Reporter can use to download the scan log from.

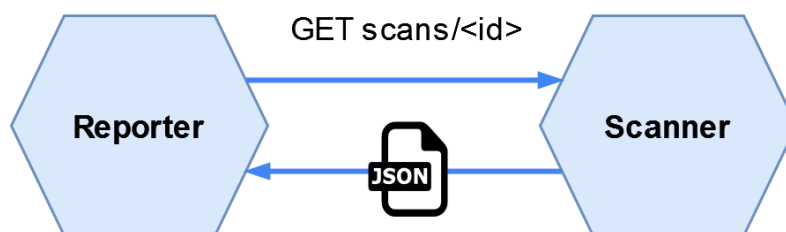


Figure 7.3: Reporter service requests a scan log from Scanner

¹⁸Based on the access control configuration Telenor chooses to employ, the Reporter service might not even be generally available to users, and they instead *must* use the API Gateway.

See Figure 5.3 for more information about the schema used by Scanner to store its scans.

7.5.3 Parsing Scans

After downloading the Snyk JSON scan output from its bucket, the Reporter service parses the contents of the scan with Python's built-in JSON parsing module. As outlined in section 7.3, it then passes the data to a Pydantic-derived class that validates its contents.

```
{
  "vulnerabilities": [
    // ...
  ],
  "ok": false,
  "dependencyCount": 183,
  "org": "pederha",
  "policy": "# Snyk (https://snyk.io) policy file, patches or
  ↪ ignores known vulnerabilities.\nversion: v1.22.1\nignore:
  ↪ {}\npatch: {}\n",
  "isPrivate": true,
  "licensesPolicy": {
    // ...
  },
  "packageManager": "deb",
  "ignoreSettings": null
  // ...
}
```

Listing 8: Snyk JSON scan log (abbreviated)

```
class SnykContainerScan(BaseModel):
    """Represents the output of `snyk container test --json`"""

    vulnerabilities: list[SnykVulnerability]
    ok: bool
    dependencyCount: int
    org: str
    policy: str
    isPrivate: bool
    licensesPolicy: LicensesPolicy
    packageManager: str
    ignoreSettings: Any
    # ...
```

Listing 9: The Pydantic-derived class that corresponds to Listing 8 (abbreviated)

This class implements various methods used to retrieve and aggregate the information from the scan. These methods include retrieving all vulnerabilities of a specific severity, getting the N most severe vulnerabilities, getting all exploitable vulnerabilities, and more. Section 7.5.4 goes more into detail on these methods.

7.5.4 Single vs Aggregate

From the outset, it was clear that we needed to support creating reports for individual images as well as being able to aggregate the results of scanning multiple images into one report. See section 5.1 for more information regarding this requirement and the concept of single- and aggregate reports.

In order to avoid having to do double the work and have to create one set of code for generating single reports and another for generating aggregate reports, we had to find a way to consolidate the code that managed the two report types and support the creation of single- and aggregate reports using shared functionality. To this end, we established an interface that *all* report data structures must follow, which we implemented for both single and aggregate reports.

```

@runtime_checkable
class ReportType(Protocol):
    # ...

    @property
    def vulnerabilities(self) -> Iterable[VulnerabilityType]:
        """All vulnerabilities."""

    # ...

    def most_common_cve(self, n: Optional[int] = 5) ->
        list[tuple[str, int]]:
        """Sorted list of tuples of CVE IDs and number of
        ↪ occurrences."""

    @property
    def most_severe(self) -> Optional[VulnerabilityType]:
        """Most severe vulnerability"""

    def most_severe_n(
        self, n: Optional[int] = 5, upgradable: bool = False
    ) -> Collection[VulnerabilityType]:
        """Returns the `n` most severe vulnerabilities (if any),
        ↪ optionally only upgradable ones."""

```

Listing 10: The interface shared by all report data structures (Abbreviated; originally >150 lines long)

The interface stipulates that classes must implement a wide range of different methods in order to fulfill the contract. Even though the implementation differs between classes, they all must take in the same arguments and return the same type of data. As long as this is fulfilled, they can be used interchangeably in the process of report creation. When we say *all*, we mainly refer to *Snyk* single image scan reports and *Aggregate* reports. However, should a new scanning tool be added in the future, its results can be integrated into aggregate reports along with existing *Snyk* reports as long as it fulfills the interface. This is the power of using interfaces; it allows for multiple different types to all use the same functions as long as they implement a common interface.

An implementation of the method `most_severe_n` for a *single report* is shown in listing 11.

```
class SnykContainerScan(BaseModel):
    """Represents the output of `snyk container test --json`"""
    # ...
    def most_severe_n(
        self, n: Optional[int] = 5, upgradable: bool = False
    ) -> list[SnykVulnerability]:
        v = sorted(self.vulnerabilities, key=lambda v:
            v.cvssScore, reverse=True)
        if upgradable:
            v = list(filter(lambda v: v.isUpgradable, v))
        if n and len(v) > n:
            return v[:n]
        return v
```

Listing 11: Implementation of `most_severe_n` for an image scanned with Snyk

The same `ReportType` interface was implemented for the aggregate report class, shown in listing 12. The difference between the two implementations is that the `AggregateReport` class must fetch *all* vulnerabilities from *all* reports it contains. Similar functionality was implemented for all methods stipulated by the interface, each time having `AggregateReport` aggregate the data from each of its reports.

As such, `AggregateReport` follows the `ReportType` interface and can be used interchangeably with `SnykContainerScan` in all functions and classes that take in `ReportType` objects. This is accomplished without any use of inheritance, thus avoiding murky method resolution behavior typically associated with inheritance in object-oriented programming.

```
@dataclass
class AggregateReport:
    reports: list[ReportType]
    id: str = ""
    timestamp: datetime = field(default_factory=datetime.now)
    # ...
    @property
    def vulnerabilities(self) -> Iterable[VulnerabilityType]:
        """Generator that yields vulnerabilities from all
        ↪ reports."""
        for report in self.reports:
            yield from report.vulnerabilities

    def most_severe_n(
        self, n: Optional[int] = 5, upgradable: bool = False
    ) -> list[VulnerabilityType]:
        vulns = list(self.vulnerabilities)
        vulns.sort(key=lambda v: v.cvssScore, reverse=True)
        if upgradable:
            vulns = list(filter(lambda v: v.is_upgradable, vulns))
        if n and len(vulns) > n:
            return vulns[:n]
        return vulns
```

Listing 12: Implementation of `most_severe_n` for the aggregate report data structure

Furthermore, we also implemented an interface for vulnerabilities (`VulnerabilityType`); again allowing for other scanning tools to be integrated as long as they can model and present the data representing vulnerabilities they detect in a certain shape and with the necessary methods. Listing 13 shows some of the properties a class that fulfills the `VulnerabilityType` interface is expected to have.


```

@runtime_checkable
class VulnerabilityType(Protocol):
    @property
    def cvssScore(self) -> float:
        """CVSSv3 score of the vulnerability."""

    @property
    def title(self) -> str:
        """Title of the vulnerability."""

    @property
    def severity(self) -> str:
        """CVSSv3 severity of the vulnerability."""

    @property
    def exploitable(self) -> bool:
        """Whether the vulnerability is exploitable."""

    # ...

```

Listing 13: The VulnerabilityType interface (abbreviated)

7.5.5 Generating a PDF Report

The end-result produced by Auspex should be a human-readable report. Considering that one of the main requirements stipulated by our employer was the ability to show the results from the service in physical meetings, we needed to find a format and presentation that worked on all devices, as well as having the capability of being presented in a printed format.

To that end, we had to find a way to produce written documents with consistent typesetting, support for tables and figures, and the ability to automate this process. It quickly became clear that LaTeX was our best option for facilitating this. Furthermore, we needed a way to generate LaTeX code automatically from the service itself; for that we used the Python package PyLaTeX¹⁹. With the help of this library we could program the structure of the report in a declarative manner, and let PyLaTeX perform the work of actually generating the LaTeX code.

We wrapped the PyLaTeX Document class in our own LatexDocument class, which encapsulates it along with the report and previous reports (used for showing

¹⁹<https://jeltef.github.io/PyLaTeX/current/index.html>

trends). Listing 14 shows the attributes of the class.

```
class LatexDocument:
    filename: str
    plots: list[Path]
    doc: Document
    report: ReportType
    prev_reports: list[ReportData]
    # ...
```

Listing 14: Custom class wrapping a PyLaTeX document (abbreviated)

Furthermore, the `LatexDocument` class implements reusable utility methods that create sections, tables and plots with minimal boilerplate code, thereby making the code easier to both read and write. Listing 15 shows how minimal the `LatexDocument` methods that add tables and figures are.

```
class LatexDocument:
    # ...
    def add_table_exploitable_vulns(self) -> None:
        """Adds a table of exploitable vulnerabilities."""
        tabledata = exploitable_vulns(self.report)
        self._add_section_longtable(tabledata)
```

Listing 15: `LatexDocument` method that adds a section containing a table showing exploitable vulnerabilities

The `add_table_exploitable_vulns()` method adds a section containing a table of exploitable vulnerabilities for the current report. This is all facilitated by the methods `_add_section_longtable()` and `_add_longtable()` which allows passing a `TableData` data structure to automatically create a section containing a LaTeX longtable.

```
def _add_section_longtable(
    self,
    tabledata: TableData,
    numbering: bool = True,
    newpage: bool = True,
    **kwargs,
) -> Section:
    """Adds a section with a longtable of the given table data."""
    if newpage and not tabledata.empty: # only make new page if
        ↪ we have rows
        self.doc.append(NewPage())
    with self.doc.create(
        Section(tabledata.title, numbering=numbering)
    ) as section: # type: Section
        if tabledata.description:
            section.append(tabledata.description)
        section = cast(Section, section) # mypy
        # Only add table if we have rows
        if not tabledata.empty:
            self._add_longtable(tabledata, LongTabularx, **kwargs)
    return section
```

Listing 16: Utility method for adding a section containing a longtable

This method in turn uses the method `_add_longtable()` to add a longtable to the section it just created in the document.

```

def _add_longtable(
    self,
    tabledata: TableData,
    table_type: Type[LongTable] = LongTabularx,
    row_height: float = ROWHEIGHT_MULTIRROW,
    booktabs: bool = True,
) -> None:
    """Creates a LongTable wrapped in a Table environment."""
    # Wrap tabular in a table so we can add a caption
    if tabledata.caption:
        ctx = self.doc.create(Table(position="h"))
    else:
        ctx = contextlib.nullcontext()

    with ctx as table:
        # Create the tabular environment
        table_spec = " ".join(["l"] * len(tabledata.header))
        with self.doc.create(
            table_type(table_spec, row_height=row_height,
                ↪ booktabs=booktabs)
        ) as tabular:
            tabular = cast(LongTable, tabular) # mypy

            init_longtable(tabular, tabledata.header)
            for row in tabledata.rows:
                add_row(tabular, row)

        # Add caption to table if it exists
        if tabledata.caption:
            table = cast(Table, table) # mypy
            table.add_caption(tabledata.caption)

```

Listing 17: Utility method for adding a longtable to the document

`_add_longtable()` is responsible for the actual creation and configuration of the LaTeX Tabular environment. Further, it inserts the table rows, and finally adds a caption if one exists.

7.5.6 Creating Tables

What facilitates these reusable methods for creating tables is the data structure `TableData`, which encapsulates all the data required to render a table and its section. Listing 18 shows the relative simplicity of this class. Other helper functions and classes also exist to add text formatting such as hyperlinks.

```
@dataclass
class TableData:
    title: str
    header: list[str] = field(default_factory=list) # column
    ↪ names
    rows: list[list[Any]] = field(
        default_factory=list
    ) # each row is a list of len(header)
    caption: str = ""
    description: str = ""

    @property
    def empty(self) -> bool:
        return len(self.rows) == 0
```

Listing 18: The `TableData` data structure

Since the data is encapsulated in a data structure that is not coupled to PyLaTeX in any way, it can be used for any type of frontend, be that a LaTeX document, a website or a mobile application. Even though our application currently only produces PDF reports, it would be trivial to render this data in other formats, as long as those formats support creating tables, embedding images and adding text.

```
def image_info(report: ReportType, digest_limit: Optional[int] =
↳ 8) -> TableData:
    """Generates the table data used to display the info for an
    ↳ image."""
    columns = [
        "Image",
        "Created",
        "Tags",
        "Digest",
    ]

    rows = [] # type: list[list[str]]
    if isinstance(report, AggregateReport):
        for r in report.reports:
            rows.append(_get_image_info_row(r.image,
↳ digest_limit))
    else:
        rows.append(_get_image_info_row(report.image,
↳ digest_limit))

    if isinstance(report, AggregateReport):
        title = "Images in This Report"
    else:
        title = "Image Information"
    return TableData(
        title=title,
        header=columns,
        rows=rows,
        caption="",
        description="",
    )
```

Listing 19: Example of a function that returns TableData

Listing 19 shows a function that takes in an object that fulfills the ReportType interface and creates a table displaying the image info of the report(s). As this object can be either a single- or aggregate report, the function needs to determine how it should retrieve the image info from the ReportType object. If the object's actual type is AggregateReport, the function iterates through all the single reports it contains.

1 Images in This Report

Image	Created	Tags	Digest
auspex/reporter	2022-05-09 16:01:08	latest	e614e96b
auspex/scanner	2022-05-09 15:58:04	latest	0ba27ca7

Figure 7.4: The table data rendered as a LaTeX table

As long as tables are created via functions that return `TableData` objects, the actual data aggregation of the program is completely frontend-agnostic. Figure 7.4 shows the image info table rendered as a PyLaTeX table. See section 9.2 for a discussion around the possibility of expanding the presentation of the Auspex reports.

7.5.7 Creating Plots

The report features a wide range of different plots used to visualize the findings. To generate these plots we used the Python library Matplotlib. Matplotlib enables the creation of a wide range of plots with minimal boilerplate code.

Similarly to the way we create tables, we also have a `PlotData` data structure for holding data for plots, so that they can be rendered by any frontend, not just LaTeX.

```
@dataclass
class PlotData:
    title: str
    plot_type: PlotType
    description: str = ""
    caption: str = ""
    path: Optional[Path] = None
```

Listing 20: The `PlotData` data structure

Plots are generated by Matplotlib and stored on disk²⁰, and then the path of the generated file is added to the `PlotData` object's `path` field. In cases where a plot

²⁰In the case of a Google Cloud Run service, this is an in-memory file system: <https://cloud.google.com/run/docs/container-contract#filesystem>

cannot be created (such as when there is no data to present), rendering methods should fall back on the plot's description to display a message stating that the plot cannot be displayed. It is up to functions that create `PlotData` objects to add alternate descriptions in these cases.

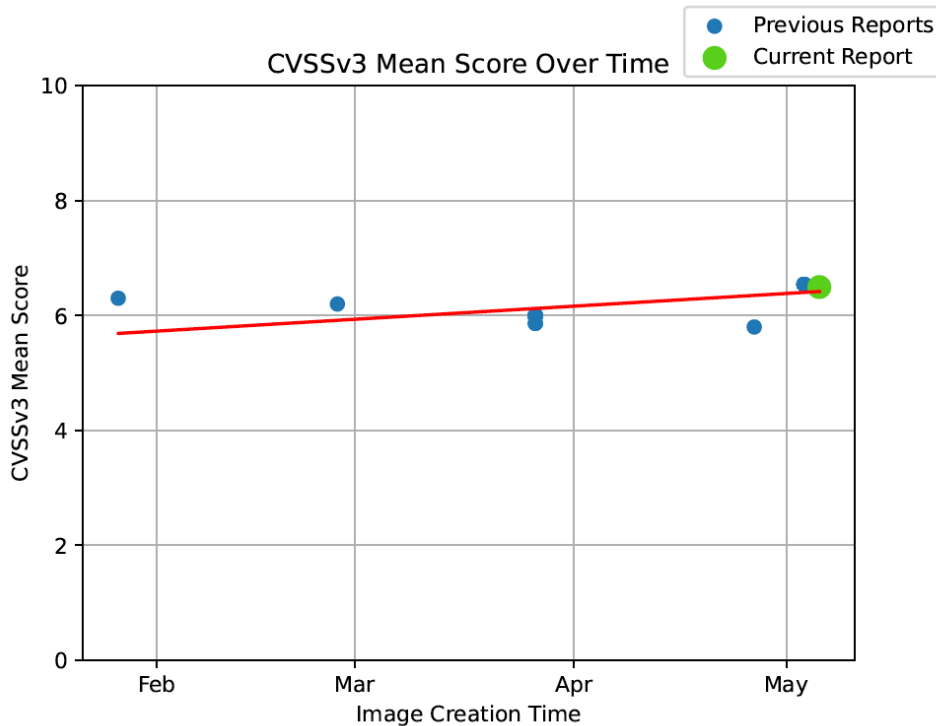


Figure 7.5: A scatter plot of CVSS mean score for current and previous reports of the same image, with a trend line

7.5.8 Final Output

The final output of the PDF report creation process is a PDF document spanning multiple pages, containing tables, figures and descriptions. This information includes:

- CVSSv3 scoring system overview
- Image information
- Vulnerability statistics
- Plot showing mean CVSSv3 score over time with trend line
- Table of most critical vulnerabilities
- Table of most critical vulnerabilities that can be upgraded
- Pie chart of vulnerability severity distribution



Figure 7.6: Structure of a report

- Scatter plot of vulnerability age and score
- Table of exploitable vulnerabilities
- Pie chart of severity of exploitable vulnerabilities
- Table of *all* critical vulnerabilities

Figure 7.6 shows an overview of the general structure of the generated report. The full report can be found in Appendix A.2.

7.6 Service: API Gateway

The service that end-users and outside applications will interact with is the API Gateway. The API Gateway service is a FastAPI application running on a Uvicorn web server inside a Docker container based on the python:3.10.4 image. The application handles incoming HTTP requests and initiates actions based on the contents of the request.

The API Gateway acts as an abstraction over the underlying Scanner and Reporter services, and is able to orchestrate these more efficiently than a user would otherwise be able to do manually. The API Gateway dispatches multiple scan requests in parallel to the Scanner service to ensure that the time it takes to get the results of a scan is constant regardless of how many image scans a user or application requests. As outlined in 7.4, each Scanner instance only takes in 1 connection at

the time, and each new connection spins up a new instance. The API Gateway orchestrates the dispatching of requests in parallel to Scanner.

API Gateway is then able to take the results of the parallel scans and send them as a request to the `POST /reports` endpoint of Reporter. After receiving a response from Reporter, it then forwards that response back to the caller. See section ?? for more information.

7.6.1 Endpoints

Most of the endpoints follow the same interface as the underlying services, and their descriptions here will refer to the description located in the relevant service's endpoints section. Figure 7.7 shows the endpoints the service exposes.

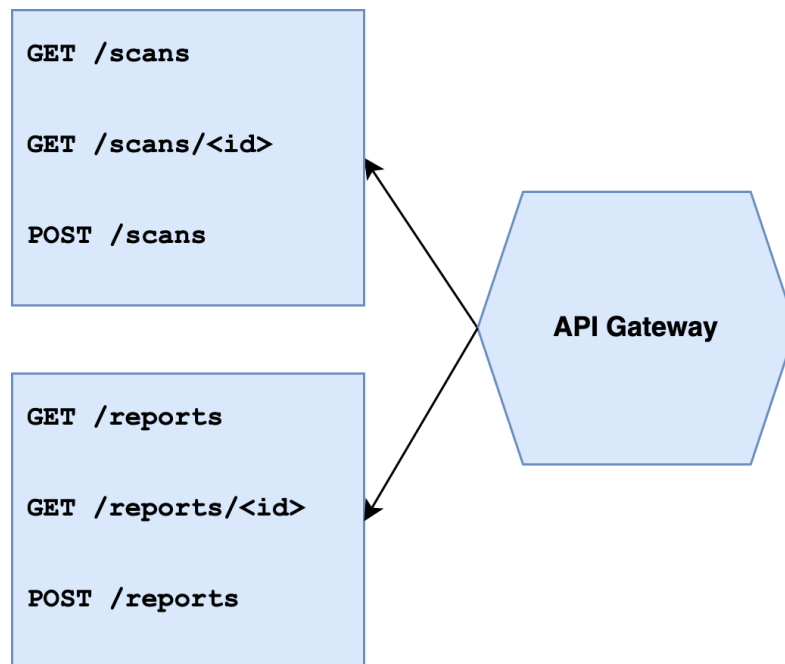


Figure 7.7: API endpoints exposed by the API gateway

`GET /scans`

This endpoint mirrors the `GET /scans` endpoint of Scanner.

See section 7.4.1.

GET /scans/<id>

This endpoint mirrors the GET /scans/<id> endpoint of Scanner.

See section 7.4.1.

POST /scans

This endpoint mirrors the POST /scans endpoint of Scanner.

See section 7.4.1.

GET /reports

This endpoint mirrors the GET /reports endpoint of Reporter.

See section 7.5.1.

GET /reports/<id>

This endpoint mirrors the GET /reports/<id> endpoint of Reporter.

See section 7.5.1.

POST /reports

As outlined in section 7.6, the service provides this endpoint as a way to automatically scan one or more images and create reports for those images, optionally also creating an aggregate report of all images.

API Gateway which uses an interface with the following schema for this endpoint:

```
{
  "aggregate": true,
  "individual": true,
  "ignore_failed": false,
  "format": "latex",
  "trend": true,
  "trend_weeks": 26,
  "images": [
    "string"
  ]
}
```

Listing 21: Schema for requests to the POST /reports endpoint

The schema shown in Listing 21 is almost identical to the schema of Reporter's POST /reports endpoint, only replacing its "scan_ids" field with "images". See Listing 7 for that schema.

The endpoint returns a data structure with a schema roughly like the one shown in listing 22.

```
{
  "reports": [
    {
      // ...
      "report_url": "string"
      // ...
    }
  ],
  "aggregate": {
    // ...
    "report_url": "string",
    // ...
  },
  "message": "",
  "failed": [
    {
      "scan_id": "string",
      "error": "string"
    }
  ]
}
```

Listing 22: Response schema for POST /reports endpoint (most keys omitted)

The data returned by the service contains information about each image and their reports, as well as the optional aggregate report. Furthermore, it contains a "failed" section with a list of errors for failed reports/scans if the option "ignore_failed" was set to true in the request, otherwise this value is empty.

See appendix B.1 for the full schema.

7.7 Testing

The concept of testing code can manifest itself in several different forms. As a developer, the first and foremost kind of testing one tends to engage with is *unit testing* [31]. Unit testing is the practice of isolating pieces of source code, called units, and testing that they behave as expected given a range of different test inputs. Certain unit testing *frameworks* exist in most languages to facilitate setup and teardown of tests, reduce the amount of boilerplate code developers have to write, and allow for more granularity and control over tests that otherwise

handwritten tests can achieve [32].

Unit testing gives developers confidence that changes they make to a system do not have a cascading effect leading to bugs and other unwanted behavior. As a project grows in size, having unit tests in place for existing code allows developers to expand and modify the system without fear of breaking things. A robust test suite gives developers more agility and confidence to work quickly.

For our project, we did not employ the development process known as Test Driven Development (TDD). TDD requires developers to write unit tests before writing implementation code, and is generally a useful approach when requirements are defined and the overall program architecture is planned out beforehand. We, however, did not have the luxury of having a fully planned out service architecture beforehand, and we had to "feel" out, that is to say work iteratively and periodically assess, the overall program structure as we went. In order to reduce the amount of time spent writing test code for implementations that might be scrapped a week later, we wrote implementation code first, then test code. This approach worked well for us, and the only thing we had to make certain of when writing the implementation code was that it was modularized to the extent that it could easily be tested. Having completed programming projects involving unit testing previously during our study program, the concept of writing testable code was not unknown to us.

7.7.1 Pytest

To test the code we used the Python testing framework Pytest²¹, which according to the JetBrains Python Developer Survey 2020 is the most popular testing framework for the language [33].

```
collected 52 items
tests/test_app.py sss [ 5%]
tests/test_cve.py ..... [ 15%]
tests/test_protocols.py ... [ 21%]
tests/test_reporter.py . [ 23%]
tests/test_utils.py ... [ 28%]
tests/frontends/test_frontends.py . [ 30%]
tests/frontends/shared/test_tables.py ..... [ 84%]
tests/snyk/test_aggregate.py .. [ 88%]
tests/snyk/test_model.py ..... [100%]
===== 49 passed, 3 skipped, 1 warning in 12.07s =====
(reporter-fgFK-q0-py3.10) bash-3.2$
```

Figure 7.8: Screenshot of Pytest collecting and running tests

²¹<https://docs.pytest.org/en/7.1.x/>

Pytest provides a comfortable test development experience with features such as the ability to use assert statements directly in the test code, fixtures to automate parameterization of test inputs, and a good command-line interface. Listing 23 shows the complete lack of boilerplate code required to write basic tests with Pytest.

```
def test_timestamp_ms_to_datetime():
    assert timestamp_ms_to_datetime("1588888888000") ==
        ↪ datetime(2020, 5, 8, 0, 1, 28)
```

Listing 23: Test using assert statement

7.7.2 Fuzzing with Hypothesis

The data structures used by the different services have numerous fields and has a wide range of permutations as a result. It is of course impossible to test every possible permutation, but spending some time testing different permutations can be highly beneficial. In order to automatically generate different permutations of the data structures when testing, we used the testing library Hypothesis²² which integrates natively with Pytest.

Hypothesis is a library that allows for fuzz testing (also known as property-based testing), where values are generated based on some property or specification. The rules for generating data are determined via "strategies", which can generate primitives such as strings and integers, but also more complex user-defined types. An example of a basic strategy is `strategies.text()`, which is a built-in strategy that creates different kinds of strings.

Pydantic also has a built-in Hypothesis plugin, and as such is able to automatically generate strategies for any Pydantic-derived class, making it trivial to integrate our large data structures with Hypothesis.

Using Hypothesis, we were able to perform unit tests with randomly generated data, which allowed us to find and uncover bugs and corner-cases much more efficiently than we could have otherwise done without it.

²²<https://hypothesis.readthedocs.io/>

```
@settings(max_examples=10,  
  ↳ suppress_health_check=[HealthCheck.too_slow])  
@given(CLASS_STRATEGIES[SnykContainerScan])  
def test_fuzz_SnykContainerScan(scan: SnykContainerScan) -> None:  
    # CVSS sanity tests  
    assert scan.cvss_max >= scan.cvss_min  
    # ...
```

Listing 24: Example of fuzzing with Hypothesis (abbreviated)

In the code snippet in Listing 24, we have limited the test to accepting 10 valid executions before being marked as successful, while the default is 100 valid executions. The reason we chose a number this low is because the program's data structures are large and take time to generate, so to avoid an extremely long execution time of our test suite, we chose a relatively low number. This optimization of runtime duration is doubly important when running tests on CI/CD machines that have less computing resources than our local machines.

Regardless of number of iterations per test, what's important is that we are able to test the program with a wide range of inputs to improve its resiliency and reliability. Hypothesis has helped us catch numerous edge-cases and bugs by simulating a far larger quantity of inputs, and more varied to boot, than we could ever write by hand. As such, Hypothesis has been an integral part of ensuring that we deliver a robust end-product.

7.7.3 Coverage

Test coverage is a metric that defines which percentage of the overall application code is executed by the program's test suite. This metric is not the be-all-end-all of testing, but is an indicator to which degree the program code that is covered by tests is able to run without errors given specific test inputs. As mentioned in section 7.7.2, given that we are running with a wide range range of inputs, we are able to fairly confidently assert that the program code we cover is robust.

However, in general, code coverage can be a misleading metric, as a block of code that runs successfully with a single known valid input is reported as having 100% coverage despite containing a bug waiting to manifest itself given the right (or indeed wrong) input. For this reason, we focused on thorough testing of important functionality over trying to achieve 100% coverage.

Coverage report: 66%				
Module	statements ↓	missing	excluded	coverage
reporter/backends/snyk/model.py	370	31	0	92%
reporter/frontends/latex/latex.py	177	121	0	32%
reporter/db.py	163	130	0	20%
reporter/backends/aggregate.py	152	3	0	98%
reporter/frontends/shared/tables.py	117	4	0	97%
reporter/frontends/shared/plots.py	109	90	0	17%
reporter/report.py	76	46	0	39%
reporter/types/protocols.py	75	0	1	100%

Figure 7.9: The coverage of the largest source files of the Reporter service

As a result, we defined comprehensive Hypothesis strategies and numerous tests for the report data structures in the program in order to maximize the coverage of their modules. Figure 7.9 shows that the module containing the Snyk data structures is the largest one in the program (800 lines²³, 370 statements), and has 92% test coverage. The lines that are lacking coverage are often fall-backs or failsafes that don't trigger due to our strategies, which can be seen as the lines marked in red in Figure 7.10.

```

attr = time_type.value # type: str
for vuln in self.vulnerabilities:
    # Put guards around our unsafe metaprogramming
    try:
        t = getattr(vuln, attr) # type: datetime
    except AttributeError:
        logger.exception(
            f"Vulnerability {vuln.identifiers} has no attribute {attr}"
        )
    continue

```

Figure 7.10: Example of code that is not covered by tests

7.8 Deployment

In order to deploy our services on Cloud Run, we needed to define build scripts that could upload the source files to Google Cloud Build²⁴, build the services' container images, and then push them to Cloud Run. To this end, we created Google Cloud Build `cloudbuild.yaml` files for each service that defined the build and deployment process for each image.

²³This is nothing to brag about, and some might see it as a symptom of a file that should be split up into smaller sub-modules

²⁴<https://cloud.google.com/build>

```
steps:
# Build the container image
- name: 'gcr.io/cloud-builders/docker'
  env:
    - DOCKER_BUILDKIT=1
  args: ['build', '-t', '${_IMAGE_NAME}', '-f',
    ↪ 'reporter/Dockerfile', '.']
# Push the container image to Container Registry
- name: 'gcr.io/cloud-builders/docker'
  args: ['push', '${_IMAGE_NAME}']
# Deploy container image to Cloud Run
- name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
  entrypoint: gcloud
  args: [
    'run',
    'deploy',
    '${_SERVICE_NAME}',
    '--image',
    '${_IMAGE_NAME}',
    # ...
  ]
# ...
```

Listing 25: Build and deployment script for the Reporter service (abbreviated)

Listing 25 shows the general structure of such a build script, and how it can be implemented. Omissions from the listing include the full extent of arguments to `gcloud run deploy`, the last step that redirects all traffic to the new version, as well as variable substitution rules. Important to note that the environment variable `DOCKER_BUILDKIT` is set to 1 to enable multi-stage builds, which we outlined in Section 7.1.3.

7.8.1 CI/CD

Continuous Integration / Continuous Delivery (CI/CD) is the practice of continuously integrating changes to the code base in a quick and agile manner while also deploying the selfsame code to production. To cover every aspect of CI/CD is outside of the scope of this thesis, but in essence it means that a great deal of the process of submitting new code, testing it, building it and pushing to production is automated. This is enabled through the use of hosted runners or dedicated servers. GitHub and GitLab both offer the ability for integrating CI/CD with source called

repositories with their services called Actions²⁵ and Pipelines²⁶ respectively.

By setting up an automated CI/CD pipeline, we were able to have our code tested, built and deployed automatically each time code is pushed to the master branch of our GitHub repository.

Testing

By using GitHub Actions, we were able to set up a workflow that ran each of the services' unit tests and reported back their completion status. Since we utilized Poetry as our virtual environment manager for Python, we needed to include a step in our workflow that installed the dependencies from the `pyproject.toml` file via Poetry. To that end, we used the action *Python Poetry Action*²⁷.

Building and Deploying

Being able to deploy the services by running the scripts manually from our local machines is well and good for the very first deployment, but subsequent deployments needed to be automated. To that end, we added the application's GitHub repository as a build trigger for Google Cloud Build. In the trigger's options, we configured it to run each time the master branch of the repository received updates, as shown in Figure 7.11.

²⁵<https://github.com/features/actions>

²⁶<https://docs.gitlab.com/ee/ci/pipelines/>

²⁷<https://github.com/marketplace/actions/python-poetry-action>

Repository event that invokes trigger

Push to a branch

Push new tag

Pull request
Not available for Cloud Source Repositories

Or in response to

Manual invocation

Pub/Sub message

Webhook event

Source

Repository *
auspex-ntnu/auspex (GitHub App) ▼

Select the repository to watch for events and clone when the trigger is invoked

Branch *
^master\$

Use a regular expression to match to a specific branch [Learn more](#)

Figure 7.11: Cloud Build repository trigger configuration

The Google Cloud Build hook runs a top-level `cloudbuild.yaml` file that contains shared environment variable definitions which are passed on to the build scripts for all 3 services (+ 1 setup script that ensures all necessary resources such as buckets and database indexes are created). All services are built in parallel and are completely independent of each other. The end result of the build process is that all services are updated, and their new images are uploaded to the project's container image registry.

7.8.2 Cloud Run Configuration

All services are mostly configured using standard Cloud Run parameters, however the Scanner service differs slightly with regards to its maximum requests per container, as detailed in section 7.4. Furthermore, it has been allocated 4 GiB of memory, as opposed to the default 512 MiB. This allows for automatic horizontal scaling when multiple images are scanned, and is one of the reasons why a managed container runtime such as Cloud Run is a great fit for our application.

7.9 Sending Emails

As a proof of concept for emailing the results, we set up a Google Cloud Workflows workflow that issues a request to Auspex for it to scan multiple images, create individual reports for each of them, as well as create an aggregate report of all

image scans. These results are then passed to the SendGrid²⁸ API which sends the results by mail to a destination address.

Listing 26 first retrieves the SendGrid API secret key from Google Cloud Secret Manager²⁹.

```
main:
  steps:
    - init:
      assign:
        - project_id: ${sys.get_env("GOOGLE_CLOUD_PROJECT_ID")}
        - secret_id: "SENDGRID_API_KEY"
        - version: "latest"
        - email_destination: "peder.andresen@gmail.com"
    - get_secret:
      call: googleapis.secretmanager.v1.projects.\
           secrets.versions.accessString
      args:
        secret_id: ${secret_id}
        version: ${version}
        project_id: ${project_id}
      result: SENDGRID_API_KEY
```

Listing 26: Retrieving a secret in a workflow.

Further, the workflow contacts the Auspex API gateway, wherein it requests the scanning and subsequent report creation of all of Auspex's images. Listing 27 shows that the workflow tries to complete this step up to 8 times before giving up.

²⁸<https://sendgrid.com/>

²⁹<https://cloud.google.com/secret-manager>

```
# Request a scan + create reports
- getReports:
  try:
    call: http.post
    args:
      url: https://restapi-qk6stf4ejq-lz.a.run.app/reports
      body:
        images: ["eu.gcr.io/ntnu-student-
          ↪ project/auspex/reporter",
          ↪ "eu.gcr.io/ntnu-student-
          ↪ project/auspex/scanner",
          ↪ "eu.gcr.io/ntnu-student-
          ↪ project/auspex/restapi",
          ↪ "eu.gcr.io/ntnu-student-project/auspex/setup"]
        aggregate: true
        individual: true
        ignore_failed: false
      timeout: 1200
    result: reportResult
  retry:
    max_retries: 8
    backoff:
      initial_delay: 1
      max_delay: 60
      multiplier: 2
```

Listing 27: Requesting scan and report creation for images in a workflow.

Thereafter, the URL of the generated aggregate report is sent by mail to a specific destination address. Listing 28 shows how this step uses the API key retrieved from secret manager initially and uses it to authenticate with the SendGrid API when issuing the /mail/send request.

```

- sendEmail:
  call: http.post
  args:
    url: https://api.sendgrid.com/v3/mail/send
    headers:
      Content-Type: "application/json"
      Authorization: ${"Bearer " + SENDGRID_API_KEY}
    body:
      personalizations:
        - to:
          - email: ${email_destination}
      from:
        email: auspexmailer@gmail.com
      subject: Daily Auspex Report
      content:
        - type: text/html
          value: ${"<a clicktracking=off href=\"\" +
            ↪ reportResult.body.aggregate.report_url +
            ↪ \">Report</a>"}
      result: email_result
- return_result:
  return: ${email_result.body}

```

Listing 28: Workflow step that sends an email.

The report is encoded as a HTML hyperlink using the `<a>` tag. Clicktracking is disabled to prevent SendGrid from rerouting the request through their own servers.

Finally, the full contents of the response from Auspex is returned to the workflow logger, which allows for inspection of the response after completion, shown in listing 29.

```

- return_result:
  return: ${email_result.body}

```

Listing 29: Workflow step that sends an email.

The recipient receives an email containing a hyperlink to the PDF report, shown in Figure 7.12

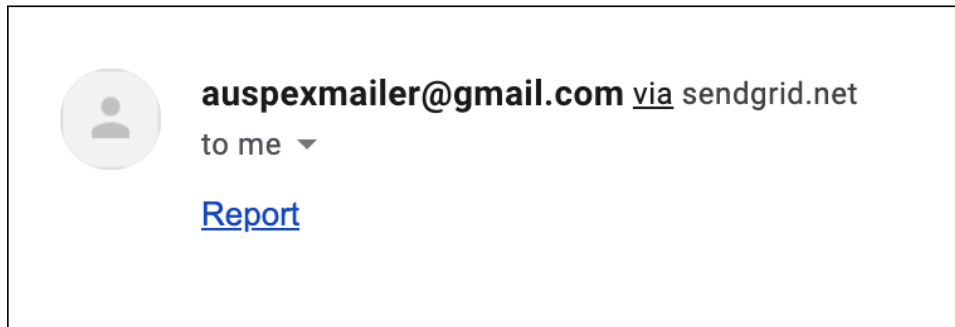


Figure 7.12: Visual representation of the workflow.

Figure 7.13 shows a visualization of the different steps in the workflow that ends with sending an email.

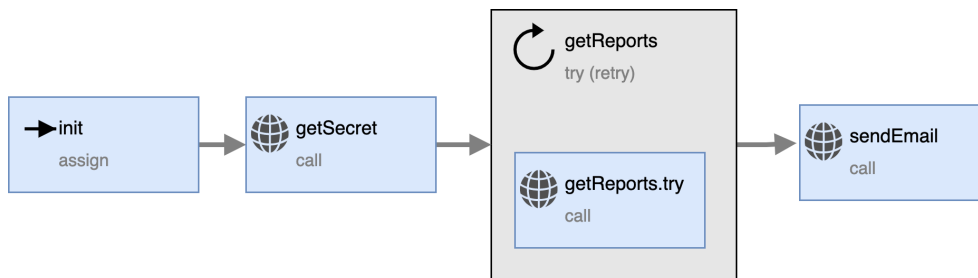


Figure 7.13: Visual representation of the workflow.

This is still a proof of concept, and has a lot of room for improvement, which we discuss in Chapter 9.

Chapter 8

Evaluation

In this chapter we discuss the survey and its results, as well as evaluate the requirements defined in Chapter 4

8.1 Survey

As part of this thesis and this project as a whole, we conducted a survey to assess perhaps the most important, and most visually prominent, part of our application; the Auspex report generated from a scan of a container image. The survey was distributed via our contact at Telenor, Eirik Stephansen, and he distributed the survey amongst his team. As the usefulness of our application will ultimately be determined by the relevancy and comprehensibility of its content, the decision on what to include and the our ability to respond to feedback is crucial. In this section, we will elaborate on how we chose to address this challenge.

The survey itself is a series of questions to assess our attempt at conveying useful and relevant information about the security state of the scanned container image. Our main goal is to evaluate whether the information displayed in the figures and tables of the Auspex report is helpful or not, and to what degree.

Note, that our assignment specification was fairly vague in stating the level of competency of the team our product is intended for, thus making it challenging to determine what to show them. For this reason, we decided to include a question for each section of our product's output that relates to whether or not the section is understandable to the reader. In addition, we added an introductory question where the participant states whether or not he or she is a developer, as a way to control for biased data. The value of this particular question in and of itself is arguably not particularly high. However it helps us understand why one respondent

might ask for more technical info or details to be presented, while another might ask for more descriptive explanations of the information presented.

In the final section of the survey we are asking the respondents if there is any information they feel is missing and would like to see, and for general feedback. Particularly, someone working with development or other task related might have an idea of information they would like to see that can be helpful in their role and would add information value to the Auspex report.

Accompanying the survey is a sample report, which will form the basis of the feedback we will receive. This sample report was generated by our software, scanning a known vulnerable image of PHP 5.4.1, found online in the Vulhub repository¹. This particular image contains many different vulnerabilities, meaning it was very useful for our purpose to test Auspex, and ensuring realistic input data. For the purpose of the survey however, the data for the trend diagram was mocked using random scores and dates. All other data was genuine. To ensure the report's content would be evaluated in a contextual manner, we decided to include a picture of each section from the example report, together with its corresponding question in our survey.

8.1.1 Survey Results

In chapter 4, we specified the requirements for the application as a whole. We also outlined some requirements relating to the Auspex report.

The survey was shared via Eirik on April 26th, with a deadline of May 6th, to his team at Telenor, and to an external team in Poland working on developing Mitt Spor. These two teams consist of 8 people in Oslo and 12 more in Poland at the time of writing, a total of 20 people. These are the people who we believe would have use for Auspex, and would directly benefit from its use. First lets preface that we did not get as many responses as we would have like to. That was the consequence of sharing the survey with only relevant people who undoubtedly had more pressing matters to attend to. Relevant people is in this context people working at or with Telenor, and working with container images. We did not have direct access to share the survey at Telenor internally and was working through Eirik to do so, and as such did not want to burden him with having to search all of Telenor to find suitable candidates. Thus our pool of possible respondents was quite limited.

Our final response rate was 15%. The amount of responses does then not provide much in the way of statistical value in and of itself. Therefore we had to evaluate the written feedback and responses we received. We are quite pleased that most

¹<https://github.com/vulhub/vulhub>

2.Statistics

2 Statistics

Median CVSS	Mean CVSS	CVSS Stdev	Max CVSS	L	M	H	C	# Vulns
6.50	6.53	1.84	9.80	88	659	469	159	1375

Where: L = Low (0.1 - 3.9), M = Medium, (4.0 - 6.9), H = High (7.0 - 8.9), C = Critical (9.0 - 10.0)

To what degree does the statistics highlight the important information of a scan?

1 2 3 4 5

Low High

Is there any other information or statistics you would like to see? Is there anything you feel is unnecessary?

Your answer _____

Figure 8.1: Example of a question from our survey

of the respondents provided written feedback to most questions.

8.1.2 Survey Feedback and Improvements

Through the survey we received feedback on the example Auspex report, and from this feedback we made corresponding improvements.

Improved Explanations

The biggest finding from the survey, and also the easiest to improve, was to provide better descriptions and explanations of the figures and tables in the Auspex report. The example report provided with the survey lacked sufficient explanations to help

the reader understand the purpose and details of a particular section.

An example of lacking descriptive and explanatory text is evident in the section for the trend diagram, as pictured in Figure 8.2. The explanation is minimal and does not provide the reader with much understanding. What is lacking is the explanation of what each blue dot represent, in addition to the what the red line represents. Making assumptions on what a random reader will understand, or not, is counterproductive. It is therefore much better to provide clear explanations from the get go to avoid misunderstanding of the information provided.

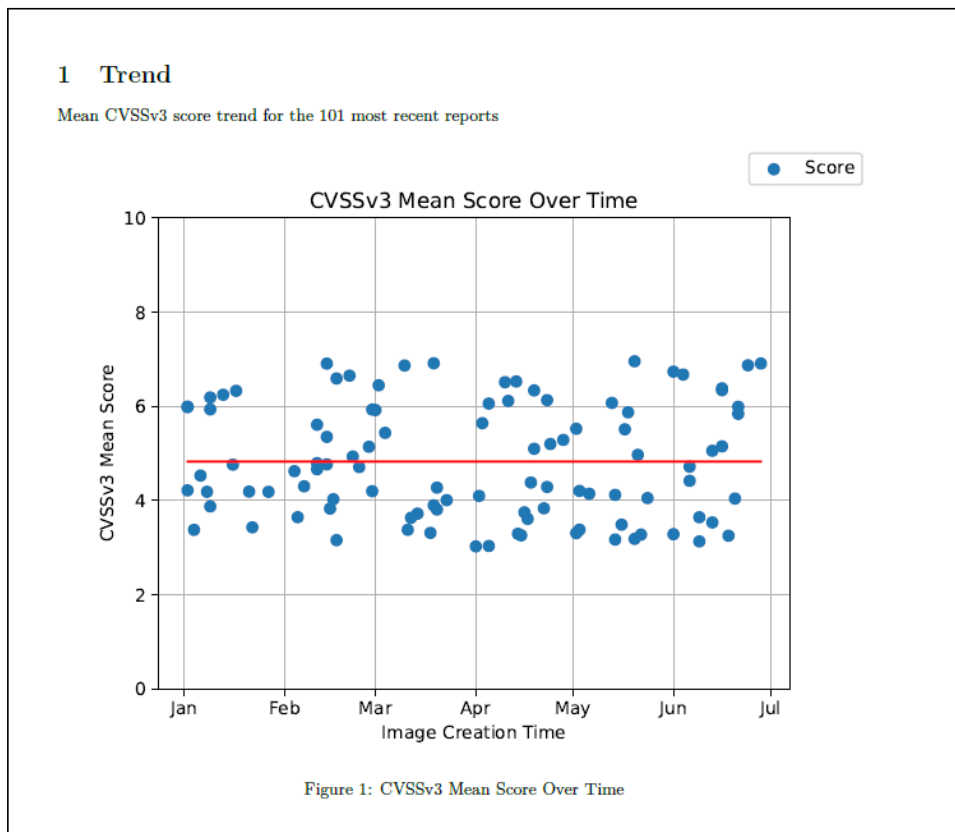


Figure 8.2: Trend section prior to feedback

In Figure 8.3 we added more explanatory text to the Trend section and gave the section a more descriptive name. In addition we added a green dot in the diagram to represent the current scan. Note that this version of the trend diagram has less data points compared to the trend diagram in Figure 8.2 due to being based on real data, and not mock data.

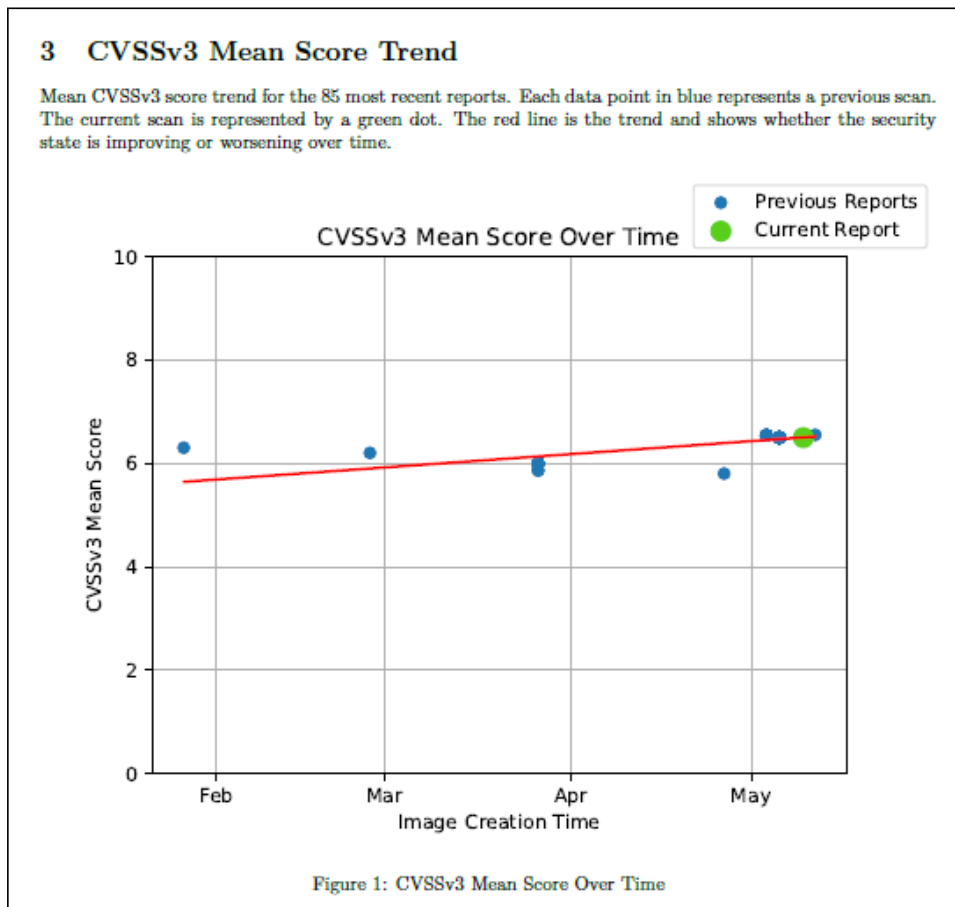


Figure 8.3: Trend section after receiving feedback.

Scoring Intervals

Throughout the Auspex report we are using the scoring intervals from CVSSv3, see Table 2.1. This was not clarified very well and led to some confusion from our respondents in terms of what the different scores meant, and their significance, in the tables and figures we presented to them. Figure 8.4 showcases our previous, minimal explanation of the scoring intervals for severities.

To provide the context for the scoring intervals used in the Auspex report, we added a new static table at the very beginning of the report, outlining the scoring intervals used throughout the report, together with a short explanation. This table, pictured in Figure 8.5, helps the reader to understand the context for the scoring values used throughout the Auspex report.

2 Statistics									
Median CVSS	Mean CVSS	CVSS Stdev	Max CVSS	L	M	H	C	# Vulns	
6.50	6.53	1.84	9.80	88	659	469	159	1375	

Where: L = Low (0.1 - 3.9), M = Medium, (4.0 - 6.9), H = High (7.0 - 8.9), C = Critical (9.0 - 10.0)

Figure 8.4: The scoring intervals, highlighted in the red box, as presented in the survey and prior to feedback

CVSSv3 Scoring System			
The following intervals are used to define the severity of a vulnerability. Scoring interval is based on the CVSSv3 scoring system, rating vulnerabilities from 0.0 to 10.0 and ranking them by severity, 'Low' to 'Critical' according to their score.			
Low	Medium	High	Critical
0.1 - 3.9	4.0 - 6.9	7.0 - 8.9	9.0 - 10.0

Figure 8.5: The scoring intervals presented at the top of an Auspex Report in the final version

Hyperlink to Vulnerability Information

In the survey feedback, and other feedback, it was suggested that the CVSS ID's should have a link to more information about that specific vulnerability. The information provided by the official sources is much more comprehensive than what could be presented in the Auspex report, and also outside its scope and purpose. Therefore we added hyperlinks for all CVSS ID's throughout the Auspex report that takes an interested reader to either the official CVE Program website², or the Snyk equivalent³, depending on which site is listed for that vulnerability in the output of Scanner. Both websites provides very similar information, and the Snyk website also provides a link to the CVE website for that specific vulnerability.

²<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-33574>

³<https://security.snyk.io/vuln/SNYK-DEBIAN10-GLIBC-1296899>

4 Most Critical Vulnerabilities

Lists the found vulnerabilities with highest CVSS scores. The CVSS ID is a hyperlink to official documentation for that vulnerability. 'Upgradeable' denotes whether the found vulnerability has a known fix ie. a new version of a package or library.

Vulnerability	CVSS ID	CVSS Score	Severity	Upgradeable
Use After Free	CVE-2021-33574	9.80	Critical	False
Buffer Overflow	CVE-2022-23219	9.80	Critical	False
Buffer Overflow	CVE-2022-23218	9.80	Critical	False
Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False
Use After Free	CVE-2021-33574	9.80	Critical	False

Figure 8.6: An example of clickable CVSS ID's in a table listing vulnerabilities

Pie Chart of Exploitable Vulnerabilities

One of the respondents asked for a second pie chart that refined the distribution of found vulnerabilities further, showing the distribution of found, *exploitable* vulnerabilities. Exploitable means that there are known exploits for a vulnerability. This can be helpful to understand the level of risk that the container images are exposed to. A container image with a high number of exploitable vulnerabilities of high severity is more exposed to risk than a container image with a low number of exploitable vulnerabilities of low severity.

Not Implemented Feedback

There was some feedback that we were unable to accommodate due to prioritizing having the basic features and implementations ready for the final delivery. Focusing on advanced tasks and implementing them well, unfortunately fell outside of the scope in the final few weeks of work. The features or changes that were suggested or asked for in the survey included clustering the vulnerabilities by type, refining upgradeable to patchable without breaking change and showing the scatter plot for age of vulnerabilities as years since current date. These can absolutely be implemented with the current codebase, but not by us within the deadline for the project.

8.1.3 Other Feedback

In addition to the survey we handed out, we also received feedback directly from Eirik, as well as Kelly, our supervisor. All the feedback was used to improve the final version of the Auspex report.

During our talks with Eirik, speaking about what information he wanted to see in a report, he was very clear that he wanted to see a trend line of the security state. The trend line provides information *at a glance*. With a quick look, one can be able to determine if the security is improving or worsening over time. This was implemented as a diagram, plotting the mean score of all vulnerabilities found for the specific image scanned. Scans of previous versions of that image is also plotted in the diagram, and a trend line is calculated and added. An example of the trend diagram can be seen in Figure 8.3.

Our supervisor had some suggestion for improvement of the Auspex report during our meeting in week 17, which can be seen in Appendix C. At the core of her suggestion was the need to clarify the scoring system and the corresponding intervals for severity. We are using the CVSS scoring system, and more specifically the CVSSv3 scoring system, in all parts of the Auspex report and in this thesis. The scoring system should be clear immediately upon reading the Auspex report. Additionally, we received feedback suggesting that the ID of each vulnerability should be clickable. CVSS uses a unique ID for all the vulnerabilities identified. This ID is non-descriptive however, and mostly used as a way of telling different vulnerabilities apart. From this feedback, and the feedback from the survey, we made the ID part of the vulnerabilities clickable hyperlinks, ensuring the reader can easily find more information as seen in Figure 8.6.

8.2 Does It Meet the Requirements?

By design, our survey only covers our application's output, the Auspex report, thus limiting the usefulness of any feedback received to just what the user sees. Therefore, we formulated a collection of both functional and non-functional requirements early on in the development process, that would ultimately determine whether or not our application functions to our, and our employer's satisfaction. In this section, we will review the requirements we set and whether or not our application meets them. We will begin by evaluating our application as it relates to our functional requirements, followed by the non-functional, before finally concluding with a discussion of our findings.

8.2.1 Evaluation of Functional Requirements

We set three functional requirements that we believe to be necessary for the application to be useful to our employer. We will go through each requirement in turn, and discuss whether or not our application actually met them, as well as the rationale behind why they were included.

F-1 "Auspex should scan container images in a connected Container Registry."

Description: Auspex needs to be connected to an existing Container Registry that contains all the container images Auspex can scan. Auspex does not have an internal Container Registry itself, but is connected to the Container Registry of the environment Auspex pulls the container images from.

Evaluation: Auspex has the capability to scan any image of whichever container registry it is authenticated with. In the current implementation, only one registry can be accessed per service deployment, but expanding this to allow authentication with multiple registries should be trivial.

Our application does indeed meet this requirement. We use Google Workflows to automate the process of pulling container images from a container registry and performing a scan.

F-2 "Auspex's Scanner microservice should perform a vulnerability scan on the given container images and output a JSON formatted response body."

Description: Given that we structure our application components as multiple microservices, one microservice should be able to pass information to another microservice, without interfering or knowing each other's internal functionality and inner workings. Passing the data in a common and predictable format allows the data to be used in different ways, but remain unchanged.

Evaluation: Our application is structured as a collection of microservices, and should thus adhere to a number of recommended practices and design patterns that unlock the benefits of this architecture type. In essence, the looser the coupling between two services, the less they know about the inner working of each other. This has several benefits regarding portability, security and reusability that are beyond the scope of this chapter, but it is important not only for the functionality of our application, but also for its continued use and development by our employer when they take over the application code. Our application meets this requirement as well since 'Scanner' does indeed pass its output in JSON format to 'Reporter' without the latter being aware of the inner workings of the former.

F-3 "Auspex's Reporter microservice should transform JSON formatted responses into human-readable PDF reports detailing vulnerabilities for a given container image."

Description: The report that Auspex generates is at the core of the application. This report is generated based on data from a scan of a container image and used

to highlight and convey key information about vulnerable container images in a human-readable presentation.

Evaluation: The core idea behind this assignment, was that the raw output from the existing container security scanning tools was too extensive and complicated to be useful for non-experts. Therefore, a simplified report highlighting the most important information within this output was needed, and specified as a requirement. We do however, somewhat rely on opinion on this point, as a clear objective assessment of the understandability of our application's output is difficult to establish, even with the survey we conducted. Our application does meet this requirement as it stands, since it can successfully transform JSON formatted responses into a PDF reports. Yet we rely on feedback from our survey and our employer to further improve report readability, and to properly assess how understandable and helpful the actual generated reports are to the intended users.

8.2.2 Evaluation of Non-Functional Requirements

In this section, we will evaluate whether or not our application met our non-functional requirements.

N-1 *"Auspex should be as available, as the guarantees given by the cloud platform provider."*

Description: Auspex will be deployed on Google Cloud Platform, and is thus tightly connected with the availability of GCP and its services. The availability is defined as a Service Level Objective (SLO) in the SLA, outlining the objectives for the service, such as uptime, latency and error rate [16]. The main services that is used by Auspex are Cloud Run and Cloud Storage, and both define their Monthly Uptime Percentage as 99.95%⁴⁵. To achieve this requirement, our application needs to be Cloud Native so that the aforementioned SLO terms are guaranteed.

Evaluation: From the outset, our application was developed with a Cloud Native design philosophy, meaning that all of the infrastructure surrounding the application's deployment is controlled by a third party cloud provider. The SLA of GCP guarantees a high degree of availability for the cloud services our application uses [16] [34] [35] [36]. Our application's expected availability thus matches the availability specified by GCP

⁴<https://cloud.google.com/run/sla>

⁵<https://cloud.google.com/storage/sla>

N-2 "Auspex must ensure confidentiality and privacy."

Description: Since Auspex deals with the security of our employer's container image environment, it is of vital importance that our application does not disclose any sensitive information to potential attackers. To this end, we strive to design the application to be Cloud Native, which in turn will allow our employer to utilize the many different security options provided by cloud platforms.

Evaluation: We decided early on that we wanted to make our project a cloud-native application. Partly for functionality reasons, but also because of the flexibility it would give our employer with regard to security. Unauthorized disclosure of Auspex reports would give an attacker a list of environment's most vulnerable points.

The application is deployed on Google Cloud Platform which provides a great degree of security and authentication configuration options for users and services [37] [38]. The application uses individual service accounts, thereby ensuring granular control over the permissions of each microservice. This allows the application to be deployed in a Google Cloud Project with the confidence that the individual microservices do not have privileges beyond the bare minimum. Moreover, since parts of Telenor's infrastructure is already hosted on GCP, the application should be easy to integrate with their existing network-based security and access control measures.

These are just some of the possibilities our employer can implement, and ultimately, we are satisfied with the confidentiality and privacy aspect of our non-functional requirements.

N-3 "Auspex reports must show all vulnerabilities detected in a scan. "

Description: The generated reports must show all vulnerabilities detected from a scan of container images. It is vitally important that our report does not omit any information, as this will undermine the credibility of Auspex's reports.

Evaluation: For our employer to be able to trust the data Auspex's reports provide, we needed to ensure that the report would not omit important information. For practical reasons however, we could not display every detail of every vulnerability in the report, as that would make the reports too long. Instead, we rank and prioritize the different vulnerabilities according to their severity, highlight those that have a known exploit, and show the total distribution grouped by severity.

Furthermore, we decided to display a complete list of all critical vulnerabilities as the very last chapter of the generated Auspex report, thereby ensuring no critical vulnerabilities could ever be omitted from the report. Should there be a need

to include a full list of every vulnerability, this would be trivial to implement, but would make the report less applicable for printed mediums due to the large number of pages this would require.

Ultimately we are satisfied with our solution in relation to this requirement. We give the reader enough information to find the most critical elements without omitting other crucial data. Furthermore, we give the readers information about the concrete vulnerabilities that can be upgraded, as well as any vulnerabilities that have working exploits.

N-4 "Auspex must produce readable and understandable reports."

Description: The Auspex report needs to be human-readable and understandable for personnel within Telenor with varying levels of expertise. Therefore, we need to ensure a minimal threshold for readability, by omitting unnecessary information, and highlighting more important elements.

Evaluation: The degree to which we are able to fulfill this requirement depends on the feedback from our employer and the results of our survey. Ultimately, the usefulness of our application will be determined by whether or not our employer finds the reports generated by Auspex helpful in their security work.

We believe we satisfied this requirement in the end by using feedback from the survey to make changes that users requested. These changes include, but are not limited to, modifying the presentation of certain figures, adding new figures, adding new columns to tables, and adding more descriptive text for each category in the report. Even if the number of survey responses was low, we made sure that the respondents were made up of people who would be using Auspex in their work.

Chapter 9

Discussion

In this chapter we discuss why we made Auspex, as well as some of the choices made throughout the project.

9.1 Why Auspex Was Developed

In the original task description presented by Telenor, see Appendix A, the project was pitched as an assignment mainly looking at different tools used to rate container image security, such as Docker Bench Security¹ and kube-bench². However, in our meeting with Eirik after the presentation, he expressed interest in being able to aggregate the information from these tools and present them in meetings where people didn't necessarily have the technological background required to understand the raw output of these container scanning tools.

In our initial talk with Eirik after being assigned this task, he told us about Snyk, and that Telenor was interested in using it, and indeed had given it the greenlight to be used for internal services. This inspired us to investigate the applicability of Snyk for our project

Looking at the capabilities of Snyk, it became obvious that it was possible to automate the scanning process, parse the results, and then aggregate it. The raw JSON output from Snyk, however, was far too complicated to read and parse manually. As such, we needed to look at the possibility of presenting that information in a concise and understandable manner. Auspex is our proof of concept that this is in fact possible.

¹<https://github.com/docker/docker-bench-security>

²<https://github.com/aquasecurity/kube-bench>

9.2 Website

As outlined in Sections 7.5.6 and 7.5.7, the data structures `TableData` and `PlotData` and the functions that return them are frontend-agnostic, and as such can theoretically be used to display the data in a range of different mediums and formats. Generating a static website using this data should be a rather trivial affair using a templating library such as Jinja³, along with a HTML/CSS framework such as Bootstrap⁴. Due to time constraints, we did not have the ability to do this before the thesis deadline.

We envision that a website could be implemented to present a user interface for scanning images, creating reports, and viewing previously generated reports. Reports could be presented natively on the website as HTML elements and embedded images, while also giving users the ability to download the reports in a PDF format. Since we already have a REST API in place to facilitate the scanning of images and creation of reports, the website itself would only act as a presentation layer over the actual business logic implemented in the individual services themselves. Having the ability to select images from a list of images found in a repository, queue up scans, schedule report creation, and more, would make the Auspex application itself usable to more than just administrators and power users.

9.3 Cloud Run

Cloud Run is an ideal platform for hosting our services, as we are not concerned about the latency associated with container cold starts⁵, and our services only run on-demand. This keeps the price of running the system to a minimum, and has no impact on the performance of the services beyond the initial request latency.

However, we discovered that due to the fact that Cloud Run containers only use an in-memory filesystem, the Reporter service could run out of memory when scanning very large images (>2GB) due to having to download the images before scanning them, thereby using a large portion of its memory just for storage. In order to rectify this, we experimented with different values the container memory limit. Even with 4GB allocated memory, Scanner was still not able to scan Reporter's image, which is based on a TexLive image exceeding 2GB in size. To that end, we had to increase the memory of the service to 8GB, for which Google stipulates that 2 virtual CPUs (vCPU) are required, up from the 1 vCPU we originally had configured. This essentially doubled the per-second runtime cost of Scanner,

³<https://jinja.palletsprojects.com/en/3.1.x/>

⁴<https://getbootstrap.com/docs/3.4/css/>

⁵A service that has no active containers running, and must spin up a container before it can process requests.

but was necessary to support scanning of large images.

If Auspex were to be scaled to support thousands of requests daily, and cost was an important metric to optimize, we would have to deploy multiple tiers of the scanning service with different specs, each designed to handle images in a specific size range. We would then need to implement a Scanner gateway service which would fetch information about each image it receives, decide the machine tier required to scan that image, and then forward the request to the appropriate scanning service. Such a scenario does not seem likely given our employer's current use case, but it is important to have a strategy in place if it does become relevant.

9.4 Email Workflow

The email workflow described in Section 7.9 is quite bare-bones, and does not allow for much manipulation and formatting of the data received from Auspex before it is sent in the email's body. Relegating this step to a service such as Google Cloud Workflows, where workflows are defined in a declarative manner gives us limited control over the email contents, as we are beholden to a limited YAML-based syntax.

The step that sends the email could be replaced by a standalone service, perhaps one hosted on a FaaS service such as Google Cloud Functions⁶. Performing this step of the email workflow in a service developed in a general-purpose programming language allowing for imperative programming, would give us more freedom to process the data received from Auspex, before sending it by email. This processing could include actions such as downloading the report and sending it as an attachment rather than as a hyperlink to a bucket, batch-sending emails, personalized emails for specific recipients, and more.

9.5 Choice of Database

When we started working on the project implementation we had two primary reasons in mind for selecting Google Firestore as the project's database solution. First and foremost, we wanted to learn more about NoSQL databases. Given that we were working with JSON data, which is composed of key-value pairs, we thought a key-value database would be the best fit. Secondly, we wanted to use a database hosted in the cloud with solid client libraries and a good pricing scheme, for which Firestore was two for two.

⁶<https://cloud.google.com/functions>

Chapter 10

Closing Remarks

In this closing chapter we summarize our learning outcome from this project, conclude the thesis and describe future work that can improve Auspex.

10.1 Learning Outcome

In this chapter we discuss the learning outcome and takeaways from this project.

10.1.1 Project

During the project as a whole we learned how to better work as a team and the importance of communication. We also learned the importance of prioritizing tasks. In the beginning we had a lot of ideas for features and implementations for Auspex, but we learned along the way that it is better to make one well-made feature rather than many badly made features. Of course, given enough time there is always the possibility to something better, but with a strict deadline we saw the need to limit ourselves.

10.1.2 Teamwork and Communication

During the project we had the unique advantage that all three team members lived together. This made communication within the team easy and lead to free-flowing discussions. It also made working together quite easy as we could look at each others computer screens and offer advice and solutions to each other with minimal delay. It was however challenging communicating externally with the

other stakeholders of the project due to being limited to using digital solutions for communication, but we adapted ourselves to this.

10.1.3 Writing the Thesis

Prior to this project we were already familiar with using Overleaf to write papers. Still, it is always a challenge to write large papers and ensuring that the content is of consistent quality, well written and correct.

10.2 Conclusion

The goal of this project was to develop an application that parses the overly extensive output of container security scanning tools, and produces a report in a more understandable and presentable format. This sounds straight forward, but when we started this project we knew very little about either of these topics. For example, we knew nothing of CVE, their scoring system, what were typical vulnerability sources, how they were ranked, or how scanning tools identified them. Additionally, for it be of any practical use to our employer, we had to learn how to design the application as cloud native. This naturally revealed another long list of topics we needed to research, including NoSQL databases, cloud platforms, and finally the crucial importance of caffeine when writing a thesis. In short, the learning outcomes from this project has been immense for us, and despite a few nitpicks, we achieved what we intended at the start of the project.

10.3 Further Work

In this section we go through some of the possibilities for further work and improvements for Auspex.

10.3.1 Website

As outlined in Chapter 9, the creation of a website would increase the accessibility of Auspex by a great degree. The code base is ready to accommodate the presentation of reports in multiple formats, and as such the creation of a website should be one of the highest priorities when expanding the number of available presentation formats. As we move forward with this project, this is the primary new feature we want to introduce.

10.3.2 Email Workflow

As discussed in Chapter 9, parts of the email workflow can be replaced by a standalone service that enables more comprehensive and more richly formatted emails, as well providing a greater degree of configurability. As we only came up with this idea at the tail-end of the project, we did not want to try to introduce new services so late in the development process. We therefore relegated this to future work, but it is definitely a high priority for us.

10.3.3 Optimizing Service Docker Images

The Reporter service's Docker image is based on a TeXLive image that exceeds 2 GB in size. It should be investigated whether or the image size can be reduced by making a custom Docker image with only the bare minimum of software and libraries required to produce the PDF report.

10.3.4 Other Scanning Tools

Our services have interfaces in place to easily accommodate other scanning tools. In the event that we discover a better alternative to Snyk, we should be able to integrate it with minimal additions to the source code.

10.3.5 Supporting Multiple Container Registries

Currently Auspex can only access one private registry per service deployment due to the way the authentication process is programmed. In the future, we envision that Scanner service deployments can include a configuration file with specific registry names and their corresponding authentication information. By allowing granular control over the authentication method for each registry, Scanner can support an ever growing number of registries.

10.3.6 Feedback

Some of the feedback we received on our survey we were unable to implement due to time constraints, as noted in Section 8.1.2. We also would have liked to conduct another survey on the final version on an Auspex report to receive more feedback and to further improve it, but there was not enough time to do so. Another survey could have helped to confirm, or disprove, that our implementation of features as suggested from the survey feedback, were good.

Bibliography

- [1] *Telenor at a glance*. [Online]. Available: <https://www.telenor.com/about/who-we-are/telenor-at-a-glance/>.
- [2] *Mitt Spor*. [Online]. Available: <https://www.mittspor.com/>.
- [3] *Snyk*. [Online]. Available: <https://snyk.io/>.
- [4] *SnykDocker*. [Online]. Available: <https://snyk.io/docker/>.
- [5] CVE® Program, *About*. [Online]. Available: <https://www.cve.org/About/Overview>.
- [6] Surbiryala, 'Cloud Computing: History and Overview,' *IEEE*, 2019.
- [7] *Containers and the Cloud: An Easier Way to Deploy Workloads*, Mar. 2022. [Online]. Available: <https://www.intel.com/content/www/us/en/cloud-computing/containers.html>.
- [8] Docker Inc, *Understanding Docker Registry*. [Online]. Available: <https://docs.docker.com/registry/introduction/>.
- [9] N. e. a. Dragoni, 'Microservices: yesterday, today, and tomorrow,' *Springer*, 2017.
- [10] Dimitriou et. al., 'Container Security: Issues, Challenges, and the Road Ahead,' *IEEE Access*, 2019.
- [11] *CVE Home*. [Online]. Available: <https://www.cve.org/About/Overview>.
- [12] FIRST, *Common Vulnerability Scoring System version 3.1: Specification Document*, 2015. [Online]. Available: <https://www.first.org/cvss/specification-document>.
- [13] *Functional Requirements*, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Functional_requirement&oldid=1066505482.
- [14] *Non-functional requirement*, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Non-functional_requirement&oldid=1073196491.
- [15] Cloud Native Computing Foundation, *CNCF Cloud Native Definition v1.0*, 2018. [Online]. Available: <https://github.com/cncf/toc/blob/main/DEFINITION.md>.

- [16] *Cloud Run Service Level Agreement (SLA)*, Dec. 2019. [Online]. Available: [Cloud%20Run%20Service%20Level%20Agreement%20\(SLA\)](#).
- [17] *Cloud Firestore*. [Online]. Available: <https://firebase.google.com/docs/firestore/>.
- [18] *Key Terms - Bucket*. [Online]. Available: <https://cloud.google.com/storage/docs/key-terms#buckets>.
- [19] *Snyk CLI*. [Online]. Available: <https://docs.snyk.io/snyk-cli>.
- [20] *TeX Live*, Oct. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=TeX_Live&oldid=1052664309.
- [21] *Matplotlib - Visualization with Python*. [Online]. Available: <https://matplotlib.org/>.
- [22] J. Fennema, *PyLaTeX*, 2015. [Online]. Available: <https://jeltef.github.io/PyLaTeX/current/#>.
- [23] Janani, *Agile Methodology*, Jun. 2021. [Online]. Available: <https://www.atatus.com/glossary/agile-methodology/>.
- [24] S. K. Pal, *Software Engineering | Classical Waterfall Model*, Mar. 2022. [Online]. Available: <https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/>.
- [25] S. Jena, *Lean Software Development (LSD)*, Oct. 2021. [Online]. Available: <https://www.geeksforgeeks.org/lean-software-development-bsd/>.
- [26] M. Rehkopf, *Kanban vs. scrum: which agile are you?* [Online]. Available: <https://www.atlassian.com/agile/kanban/kanban-vs-scrum>.
- [27] *MkDocs*. [Online]. Available: <https://www.mkdocs.org/>.
- [28] G. van Rossum, B. Warsaw and N. Coghlan, *PEP 8 – Style Guide for Python Code*, Jul. 2001. [Online]. Available: <https://peps.python.org/pep-0008/>.
- [29] The Chromium Projects, *Memory safety*. [Online]. Available: <https://www.chromium.org/Home/chromium-security/memory-safety/>.
- [30] G. Thomas, *A proactive approach to more secure code*, Jul. 2019. [Online]. Available: <https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/>.
- [31] I. Cunningham & Cunningham, ‘Ten Years Of Test Driven Development,’ Jun. 2014. [Online]. Available: <https://wiki.c2.com/?TenYearsOfTestDrivenDevelopment>.
- [32] Kent Beck, *A Brief History of Test Frameworks*, Aug. 2007. [Online]. Available: <https://shebanator.com/2007/08/21/a-brief-history-of-test-frameworks/>.
- [33] JetBrains s.r.o., *Python Developers Survey 2020 Results*, 2020. [Online]. Available: <https://www.jetbrains.com/lp/python-developers-survey-2020/>.
- [34] Google Cloud, *Firestore Service Level Agreement (SLA)*, Jan. 2020. [Online]. Available: <https://cloud.google.com/firestore/sla>.

- [35] Google Cloud, *Cloud Storage Service Level Agreement (SLA)*, Nov. 2021. [Online]. Available: <https://cloud.google.com/storage/sla>.
- [36] Google Cloud, *Workflows Service Level Agreement (SLA)*, Dec. 2021. [Online]. Available: <https://cloud.google.com/workflows/sla>.
- [37] Google Cloud, *Authentication overview*, May 2022. [Online]. Available: <https://cloud.google.com/docs/authentication>.
- [38] Google Cloud, *Identity and Access Management (IAM)*. [Online]. Available: <https://cloud.google.com/iam>.
- [39] MongoDB Inc., *MongoDB Limits and Thresholds*. [Online]. Available: <https://www.mongodb.com/docs/manual/reference/limits/#mongodb-limit-BSON-Document-Size>.

Appendix A

Additional Material



Sikkerhet i Containers

Bakgrunn

Stadig flere IoT enheter blir tilkoblet nettet for å kunne gi brukerne enkel tilgang til å styre og kontrollere disse. Disse enhetene lages av mange ulike leverandører, alt fra små til store multinasjonale selskaper. For mange forbrukere er det en lav terskel for å installere og bruke disse enhetene og muligheten for at persondata kommer på avveie er økende. For å kunne sikre persondata er en avhengig av at tilbyderne forholder seg til GDPR (General Data Protection Regulation) og også gir brukerne mulighet til å kontrollere sine egne data uten at det gir en risiko for at data kan gå tapt eller misbrukt på noen måte.

Brukerdata lagres ofte i skyen og i mange tilfeller brukes løsninger fra Amazon, Google, Microsoft og andre leverandører av skytjenester. I et tenkt oppsett vil kanskje løsningen være basert på Containers som er organisert av en Orchestration løsning. Dette kan f.eks. være Docker Containers og Kubernetes Orchestration løsning. Enheten vil da kommunisere med denne skyløsningen over internett, enten gjennom WiFi, kablet eller mobil tilkobling. I en del tilfeller krever disse enhetene en egen gateway som kommuniserer med skyløsningen.

For ulike enheter vil protokoller som brukes kunne være Zigbee, Z-Wave, WiFi, eller Bluetooth mellom enheter og gateway, MQTT eller proprietære protokoller mot skyløsningen. I tillegg vil apper kommunisere mot skyløsningen basert på http eller websocket-baserte protokoller.

Oppgave 2 (Sikkerhet i Containers):

Oppgaven blir å vurdere sikkerheten rundt Containers og Orchestration løsninger. Dette gjøres ved å sette opp en enkel løsning som bruker et minimum av 2 Containers. Det enkleste vil være å sette opp en web-front som bruker en database og at disse kjører i hver sin Container. Det finnes ulike eksempel på nettet av løsninger som gjør dette, f.eks.:

<https://wkrzywiec.medium.com/how-to-run-database-backend-and-frontend-in-a-single-click-with-docker-compose-4bcda66f6de>

Når denne løsningen er oppe og kjører blir oppgavens neste steg å vurdere sikkerheten på løsningen. Dette gjøres ved å ta utgangspunkt i Docker Bench Security og kube-bench. Etter å ha kjørt disse må en vurdere resultatet og komme med et forslag til hvordan en kan bruke dette for å:

1. Forbedre sikkerheten i den oppsatte løsningen
2. Gjøre kontinuerlige vurdering av løsningen etter hvert som en videreutvikler denne

I Telenor har vi fokus på personvern og sikkerhet, og mange av løsningene som finnes på markedet kan være tvilsomme med hensyn til dette. Målet med oppgaven er å se på **hvilke verktøy en kan bruke for å evaluere sikkerhet** for de gitte komponentene i et IoT system og komme fram til et forslag om **hvilke kriterier en bør sette for et IoT system** som en ønsker å tilby til Telenor sine kunder.



I dette arbeidet er det en fordel å sette opp en risikoanalyse og bruke dette som benchmark på hvor bra evalueringen fungerer.

Oppgavens **mål blir å sette seg inn i hvordan Docker Bench Security og kube-bench fungerer** og så bruke resultatet fra dette til **å komme med en anbefaling** hvordan disse kan brukes på en effektiv måte for å bedre sikkerheten i løsninger der en har flere Containers. Finnes det andre verktøy som kan bidra til å forbedre sikkerheten enda mer og som kan anbefales?

Kontaktperson: Eirik Stephansen, Tech Lead IoT, Telenor Mobil
92220222, eirik.stephansen@telenor.no

Project Plan - Bachelor Thesis

Peder Hovdan Andresen

Patrick Eilert Krosby

Anders Christoffer Westby

31. January 2022

Table of Contents

1 Background and Goals	4
1.1 Background	4
1.2 Project Goals	4
2 Scope	5
2.1 Subject Area	5
2.2 Task description	5
2.2.1 Requirements for Cloud Application	6
2.3 Limitations	7
3 Project Organization	7
3.1 Roles and responsibilities	7
3.2 Workflow and Group Rules	8
3.2.1 Communication channels	9
3.2.2 Group rules	10
4 Planning, execution and reporting	11
4.1 Main Project Sections	11
4.2 Development Model	11
4.3 Method and Approach	13
4.4 Status Meetings and Decision Points	14
5 Quality Control	14
5.1 Documentation, Standards and Source Code	14
5.1.1 Software Documentation	15
5.1.2 Standards	15
5.1.3 Version Control	16
5.2 Inspections and Testing	16
5.2.1 Formatting	17
5.2.2 Static analysis	17
5.2.3 Tests	17
5.2.4 Deployment	17
5.3 Risk Analysis	18
6 Project Activities	21
References	22

1 Background and Goals

1.1 Background

Telenor¹, one of the leading telecommunication companies in Norway, has a great focus on security. They deliver a wide range of security products and solutions to both private consumers and businesses. A large part of these solutions are cloud based solutions using providers such as AmazonWebServices (AWS)², Google Cloud Platform (GCP)³ or Microsoft Azure⁴, and organized in containers, typically created using Docker, and orchestrated using container orchestration solutions such as Kubernetes.

To evaluate and ensure satisfactory security levels for components running in these container solutions it is necessary to use tools such as Docker Bench Security⁵ and kube-bench.⁶ The purpose of these tools is to run and scan a provided container image and return the result. The information provided can often be difficult to parse and understand, especially for personnel without technical knowledge.

1.2 Project Goals

The goals of the project will be to deliver a bachelor thesis describing and reflecting on our work, the process and decisions, as well as our recommendations for which scanning tool to use for container security evaluation. As a part of this we will also develop our own product, a tool to parse logs from container security scanners and generate readable and understandable reports from these scans. The reports generated can be used internally to make security decisions across different levels of competency. The goal of the tool will be to make it available to Telenor so they can integrate it, or adapt it, for their own workflows.

¹ <https://www.telenor.no/om/>

² <https://aws.amazon.com/>

³ <https://cloud.google.com/>

⁴ <https://azure.microsoft.com/en-us/>

⁵ <https://github.com/docker/docker-bench-security>

⁶ <https://github.com/aquasecurity/kube-bench>

2 Scope

2.1 Subject Area

For this project we will branch across multiple subject areas, such as:

- **Software Development**
 - Developing an application for parsing information, deploying this on GCP and integrating it into a workflow.
- **DevOps**
 - Developing, delivering and operating an application, automating and optimizing processes.
- **Cloud Operations**
 - Integrating our application into a workflow and hosting it on a cloud service, including processing and storage of logs and reports.

This means we will need to acquaint ourselves with tools, software and processes, such as

- Docker and Kubernetes
- Security tools
- Container security
- GCP

2.2 Task description

In order to create a product that satisfies our employer's requirements, we have to create a cloud application consisting of several different microservices that can scan container images and produce PDF reports. These reports can be for either a single container image or a set of container images.

Firstly, we need a REST API service that exposes this functionality through HTTP endpoints. Furthermore, we need to create an application that can perform container scanning and produce formatted log files that can be parsed by other programs and stored in a database. Finally, we need an application that can use the aforementioned logs to produce human readable reports in a PDF format that contains the most relevant information from the scans, presented in a way that people without an IT background can understand.

To begin working on the aforementioned services, we will get acquainted with different container security scanning tools that can be used to evaluate security vulnerabilities in container images. We will then need to create a service that uses this scanning application and expose its functionality through an endpoint in our REST API.

Furthermore, we will need to create another service that can host the report generator itself. This will require the most work out of everything in the project, and will require the majority of our time to implement due to the considerations of the project requirements, both functional and non-functional. We will need to run tests such as A/B testing in order to evaluate the optimal report format for people without IT backgrounds, as this can provide us with valuable feedback to improve how we present the information in the report.

2.2.1 Requirements for Cloud Application

Functional Requirements

- Create a cloud application that can run vulnerability scanning on container images and output a JSON-formatted report.
- Create a cloud application that can transform JSON-formatted reports into human-readable PDF reports detailing vulnerabilities for a given container.
- Set up a cloud-based service exposed through a REST API using the two previously mentioned applications, that can be integrated in a CI/CD pipeline to automatically generate human-readable reports in a PDF format detailing vulnerabilities for a given container image or a set of container images.

Non functional requirements

- Streamline the process of scanning a given container image or images and sharing findings.
- Easier to retrieve log data from security scans.
- Reports that are understandable by people without any technical background.

Preliminary List of Scanner Candidates:

- Docker bench
- Kube-bench

- Snyk⁷
- GitHub Actions (Dependabot, etc.)
- GitLab CI/CD

Preliminary Metrics for Testing:

- Number of vulnerabilities found
- Scan duration
- Amount of metadata for each scan
- Amount of metadata for each vulnerability
- Number of languages / platforms supported

2.3 Limitations

The scope of this thesis will be limited to the aspects of container security that can be discerned from analyzing container images in a non-runtime environment. Thus, while our analysis is limited to the CVE program's list of identified vulnerabilities, this list is constantly updated and added to by a dedicated and professional community. However, there are certain vulnerabilities that our analysis will not include, such as vulnerabilities in the software itself, the system and -network, architecture, and how persistent data is stored. This means that our analysis will not be addressing the handling of personal data under GDPR, nor will it evaluate other privacy related issues.

3 Project Organization

3.1 Roles and responsibilities

Team members

- Team Lead - Patrick E. Krosby
- Lead Developer - Peder H. Andresen
- Secretary - Anders C. Westby

Other parties

- Project Advisor: Jia-Chun Lin, NTNU

⁷ <https://snyk.io/>

- Telenor Norge via: Eirik Stephansen, Telenor

3.2 Workflow and Group Rules

Time tracking

All members of the team will use the time tracking tool Clockify⁸ to log their work hours. All time spent on the project by each individual team member needs to be logged with a small description of what work has been done.

Kanban board

The team will utilize a Trello⁹ board to keep track of the tasks that are being worked on, what needs to be done and what tasks have been completed. The structure of our Trello board will serve to mimic a kanban board, which is highly relevant to our chosen development methodology (Agile). In order to efficiently organize sprints, we need a Kanban board to keep track of the various tasks that require our attention. New tasks will be added continuously and all members of the team will need to keep themselves updated on changes and updates.

Meetings and minutes of meeting

The team will have weekly meetings with the project advisor on Wednesdays from 13:00 to 13:30. Before each meeting with our advisor we will hand in a weekly status report via email outlining the work that has been done, issues encountered and plans for the following week using the predefined template located in the shared Google Drive folder.

We will have continuous contact and communication with our employer and set up meetings as necessary. The goal of these meetings will be to keep our employer updated on our progress as well as clarifying any question that may arise from our work.

Minutes of meetings will be kept for all meetings with both advisor and employer starting from week 4.

Notes

We will use the note taking and knowledge organization tool Obsidian¹⁰ to write notes and document our knowledge in a structured way. Obsidian automatically structures documents

⁸ <https://clockify.me/>

⁹ <https://trello.com/>

¹⁰ <http://obsidian.md>

in a graph (see Graph Theory), which enables us to see how they relate to each other. This allows us to write short notes and documents, which can then be sorted and structured based on their relationships, be it direct references or common tags.

Documentation

For documenting our work and writing our thesis we will use Overleaf, an online LaTeX editor,¹¹ to generate documents typeset with LaTeX. This allows us to write the thesis collaboratively and ensures everyone is working on the same version of the thesis. Furthermore, writing the thesis in LaTeX ensures consistent typesetting across the entire document, as well as providing numerous tools to produce high quality figures.

For writing and storing other documentation such as minutes of meeting and weekly status reports we will write these documents in Google Docs and use a shared Google Drive folder only accessible by the team members to store them in.

Expenses

We do not foresee any notable expenses related to the project. There may be costs associated with using cloud services but Telenor has offered to incorporate us into their GCP plan. As such, any cost for the team related to GCP will be minimal and mostly related to early testing and exploration for ourselves. Any cost incurred using other software or services will be minimal and if necessary the cost will be split equally among the team members. Any travel expenses incurred will be covered by the team members individually.

3.2.1 Communication channels

Slack

Our employer wanted to use the chat application Slack¹² as our primary channel of communication. To that end, we have set up a Slack workspace where we can conduct continual text-based communication with him. He has signaled to us that he will attempt to answer any questions we have as soon as possible, and that we should bring any inquiry to him.

Microsoft Teams

¹¹ <https://www.overleaf.com/>

¹² <https://slack.com/>

For the weekly meetings with our advisor, we came to a mutual agreement to use Microsoft Teams due to its world class video conferencing functionality. Furthermore, our Teams team will also serve as a place to send text messages and upload files that are relevant for our advisor.

Email

For other communication purposes, we will use email, as this is an ubiquitous line of communication that nearly everyone can be reached through. Formal inquiries to other parties will be conducted via email whenever possible.

In-person meetings

We are in a unique situation where all 3 members of the team live together, and as such we can hold in-person meetings whenever the need arises at a short notice. We will attempt to work collaboratively in person to the maximum extent possible.

3.2.2 Group rules

Meetings

Attendance for all meetings is mandatory. All team members are expected to show up for the meetings prepared and ready.

Workload

As outlined in the course description, each team member is expected to work 25- 30 hours each week.

Group Work

All team members will utilize the team's Kanban board to inform themselves of which tasks need to be worked on at any time. Most work will be done collaboratively.

All team members are expected to contribute equally and to log their time using clockify during or after each work session with a short description.

Absence

If a team member falls ill or is otherwise unable to attend meetings or work activities the rest of the team shall be notified. Work tasks and activities will be distributed among other team members as necessary.

Violation of rules

Violation of any rules will result in a strike against the offending team member.

All team members should strive to not receive any strikes against them and contribute to a good working environment.

4 Planning, execution and reporting

4.1 Development Model

There were multiple development models that we discussed using for this project. Among these were the Waterfall model, and the Agile methodologies of Lean, Scrum and Kanban. Choosing the right development model for us and this project boiled down to the question; What are our goals and how do we achieve those? We wanted to use a development model that allowed us to iterate frequently and fast, as well as quickly and continuously respond to adjustments or changes in scope or requirements. This immediately rules out the very rigid Waterfall methodology as it makes it difficult to readjust to scope or requirements changing during the project.

The Agile methodology and approach focuses on continuous development, iteration and testing when developing software [1], which suits our needs. Now our choice is between Lean, Scrum and Kanban.

Lean focuses on delivering only what is needed and eliminating waste by employing a strategy of producing a minimum viable prototype (MVP) and reiterating on feedback.[2] We could use the Lean model because we are intending to deliver a product to the customer in the form of an application. However, we don't believe it takes into account the whole process of the project. Lean is very focused on only the necessities of development, eliminating meetings and documentation demands.[3] This aspect does not work well with producing a well-reflected bachelor thesis.

Scrum is organized in 2-3 week long sprints, where each sprint defines a set of features that must be produced.[1] Sprints then result in a delivery of the product to the customer. Scrum does however fall into the trap of the Waterfall model and being rigid in terms that the sprints

require the features and tasks to be defined before the start of the sprint. This does not work well with the frequent changes of requirements and scope that we expect.

Kanban on the other hand acknowledges that the 2-3 week sprints are too long and doesn't allow for reprioritization and changing of tasks during the sprints. The Kanban method approaches the agile workflow in a different way, utilizing a Kanban board for organizing tasks and priorities. The process is continuous and allows for changes to happen at any time.

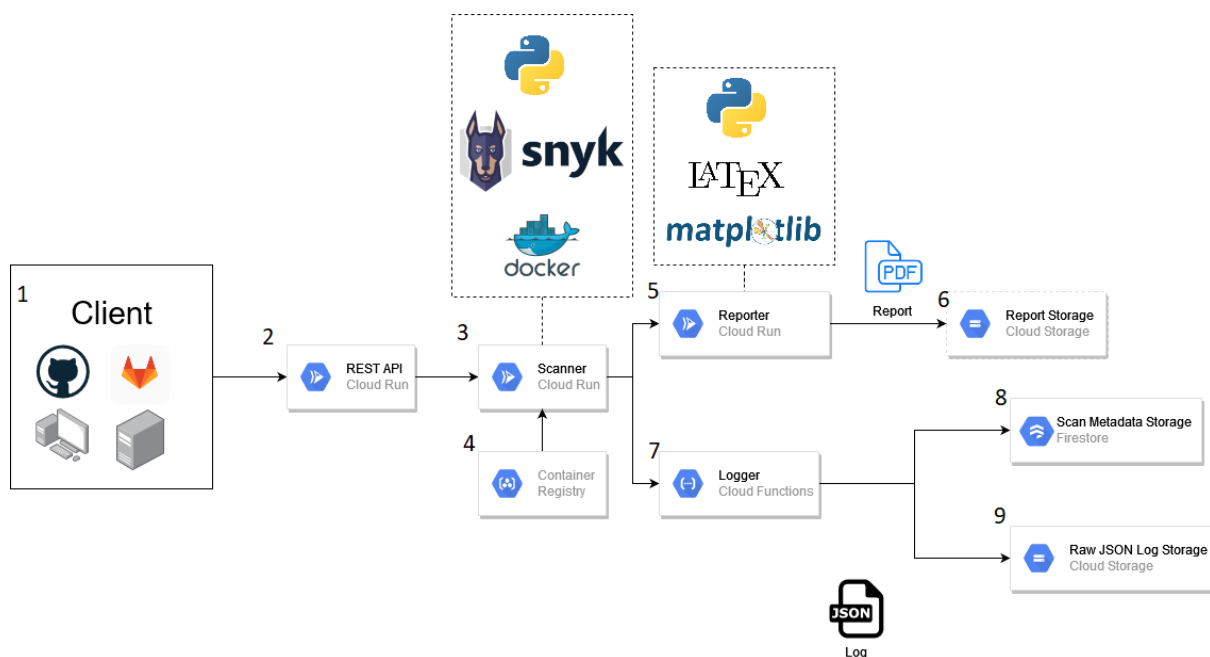
For our purposes, we have found that an agile development model suits what we are trying to achieve, and more specifically the Kanban methodology. This allows for a continuous workflow where we can respond to changes quickly without being locked in to a predefined process, and ability to go back and reiterate on previous iterations based on feedback. This methodology also allows for exploring different subjects and technologies that may change the scope or requirements. We will use Trello as our Kanban board to track our work tasks, and a Gantt chart to monitor progress, keeping track of milestones and making sure we are on schedule. We will not apply a pure Kanban methodology, also utilizing informal sprints.

At the center of Kanban, is the Kanban board. We will use Trello for our purposes. Tasks that are created will start in the "To do" category, moved to the "Doing" category when being actively worked on, and to the "Done" category when completed. Tasks will be assigned a deadline date if applicable. Tasks that are deemed to be of a lower priority will be moved to a separate category named "Backlog". We will only begin working on those tasks once the higher priority tasks have been completed. Usually the Kanban methodology requires that you limit the number of tasks that are in the "Doing"-category. We will not enforce any strict limit for this as there are multiple tasks that may need to be done in parallel, but strive to keep this at a sensible limit.

A valuable tool we will use in parallel is our Gantt chart. We will consult this chart weekly to make sure we are meeting our deadlines, achieving milestones and are on track.

We will utilize sprints from the agile methodology, but not as strict as the Scrum methodology. We will organize our work in 2 week sprints, where we allow for changes in tasks and approach to accommodate changes in scope and requirements. After each sprint period we will conduct our sprint review with our advisor as part of our weekly meetings. Every second meeting is then a sprint review. A new sprint will then start after this meeting.

4.2 Method and Approach



1. A client that contacts the REST API and requests it to scan a container image or retrieve a specific scan report.
2. A REST API hosted on Google Cloud Functions that provides a limited number of endpoints for interacting with the microservices within the architecture.
3. Scanner is a Docker container based on the Python3.10 image, and it has Snyk and Docker installed. The container runs an application that can pull Docker images and scan them using Snyk, which is a vulnerability scanning tool. We will use Snyk to scan container images for vulnerabilities. The scan results are returned as a JSON encoded string in the response payload.
4. Container Registry that is used to store all container images used by Telenor.
5. Report Generator is a Docker container based on the texlive/texlive image. It runs a Python application to transform one or more scan logs produced by Scanner into a human-readable PDF report.
6. Report storage is a Cloud Storage bucket where all PDF reports will be stored.
7. Logger is a simple cloud function that takes a JSON-encoded scan log generated by the Scanner, and stores it in a NoSQL database (Firestore in this case).
8. Scan Metadata Storage stores metadata for each scan log, such as image name, time of scan vulnerability score and id of the scan. The metadata is stored in a Firestore database.
9. Raw Jsn Log Storage is a Cloud Storage bucket where the raw logs from scans are stored.

4.3 Status Meetings and Decision Points

- Weekly meetings with the advisor every Wednesday at 1pm. Prior to these meetings we will hand in a weekly status report outlining our work, plans and any problems we may have encountered. These meetings will be used to get feedback.
- Meetings with the employer will be scheduled as needed to show progress. We will keep continuous contact via Slack to ask questions and share progress.
- As a part of our agile approach and ability to work together physically in most work sessions, we will make continuous decisions based on inter-team discussions, as well as discussions with our advisor and employer.

5 Quality Control

5.1 Documentation, Standards and Source Code

All work and relevant findings shall be documented and sources regularly added and updated. All code written should follow common coding standards and comment should be added for all code.

The main documents that we expect to write are:

- Project plan
- Technical documentation (mkdocs hosted on gitlab)
- User manual for the product
- Tutorial
- Bachelor thesis
- Minutes of meeting
- Weekly Status Reports for advisor

5.1.1 Software Documentation

In order to provide a user manual, as well as documentation for future developers, we will use the software MkDocs¹³ to create a static website hosted on GitLab serving documentation that is both written by us and automatically generated from our source code.

The user manual portion of the documentation will be hand-written by the team, and will feature guides and tutorials on how the application works and how to use it.

Furthermore, developer documentation will include hand-written sections as well as auto-generated API documentation for the various parts of the project source code.

5.1.2 Standards

Citations

Our citation style of choice for the bachelor thesis itself will be the IEEE citation standard¹⁴ due to the technical nature of our product, as well as being recommended by our supervisor.

Python

For docstrings in our source code we will follow the Numpydoc standard. We have chosen this standard due to its readability, structure and compatibility with type hints. Other types of source code documentation in Python files will automatically be formatted by the Python code formatter Black, which has a single canonical style that cannot be modified.

Additionally, we will strive to have close to 100% type coverage of our Python code through the use of type annotations. Good type annotations will help us uncover errors stemming from Python's dynamic type system. Furthermore, it helps new developers more quickly understand our code, as each function and variable will have clearly defined types rather than being opaque and untyped. An additional benefit of this is that we can compile performance sensitive modules to C extensions using Mypyc¹⁵ if we have full type coverage

Go

All source code and documentation written in Go will be automatically formatted with Gofmt. We will use the coding practices outlined in Effective Go¹⁶ as a guideline for writing code in

¹³ <https://www.mkdocs.org>

¹⁴ https://www.ieee.org/content/dam/ieee-org/ieee/web/org/conferences/style_references_manual.pdf

¹⁵ <https://github.com/mypyc/mypyc>

¹⁶ https://go.dev/doc/effective_go

Go. Due to Go's static type system, we will not have to worry about providing extra type annotations for our code like with Python.

5.1.3 Version Control

The full extent of our source code will continually be published in an online Git repository hosted on NTNU's GitLab solution in a group shared by all members of the bachelor project¹⁷.

Git will serve as both a version control tool for working collaboratively, as well as serving as an online backup of our source code. Git has become the de facto standard version control system for developers over the years, and has an adoption rate of upwards of 90% in this group¹⁸. Its relevance to our future jobs, its adoption rate, and the existence of a GitLab solution hosted by NTNU are all reasons we picked Git as our version control system.

When implementing features, we will make use of Git branches. A team member creates a branch from the master branch, implements their change, then submits a pull request that is accepted by one of the other group members. Should the need arise, we can enable multiple sign-off to accept pull requests, but in usual circumstances we will only require a single group member to accept the pull request.

5.2 Inspections and Testing

We will continually run our code through a CI/CD pipeline that performs formatting, static analysis, and testing. Pull requests must pass the checks implemented by our pipeline before they can be merged into the master branch. We will use GitLab CI/CD to set up the pipeline.

5.2.1 Formatting

As outlined in section 5.1.2 we will make use of automated code formatting tools, which includes the formatters Black for Python and gofmt for Go. These formatters ensure our code is formatted with a consistent style regardless of the author that contributed it. Having a consistent style is important both for code readability and for producing informative diffs. If two authors format their code differently, a diff message will not produce an output that

¹⁷ <https://gitlab.stud.idi.ntnu.no/containers-bachelor>

¹⁸ <https://en.wikipedia.org/wiki/Git#Adoption>

clearly shows the difference between two versions of the same file. Running a formatter post-commit will help mitigate this issue.

5.2.2 Static analysis

As part of the Continuous Integration part of our pipeline, we will run static analysis tools like Pylint to find common logical errors produced by developers, such as unbound variables and missing return statements.

Furthermore, we will make use of mypy to lint the soundness of the types in our code. As stated in section 5.1.2, we will strive to have 100% type coverage, which will improve the analysis mypy is able to provide, as type hints are the main heuristic used by mypy.

5.2.3 Tests

We will strive to have 100% test coverage of all code running in containers (this excludes code running in Cloud Functions). For Python code, we will write our tests with the testing framework Pytest in mind, while for Go we will write tests using the builtin `testing` package. In order to achieve maximum test coverage, we will make use of the Pytest plugin pytest-cov to map out the modules in most urgent need of more robust tests.

For a commit to pass the testing phase of the CI/CD pipeline, no tests are allowed to fail. In the initial development phase, we will deem it acceptable to mark certain tests to be skipped for convenience. However for final deployment, all tests must pass before code can be merged with the master branch.

5.2.4 Deployment

Once our software architecture is ready to be deployed on Google Cloud Platform, we will have to integrate a deployment phase to our CI/CD pipeline. After all checks have passed, new versions of our software should be automatically deployed to Google Cloud Platform by the CI/CD pipeline.

5.3 Risk Analysis

This is a simple risk analysis of the project and what we deem are the most likely risks to occur throughout the project and development period. For each risk there is a short

description followed by an evaluation of the likelihood of the risk occurring and the impact of the risk on the project. Likelihood and impact will be assigned with a score of 1 through 5, with 1-2 being low, insignificant or trivial, 3-4 being a medium risk and 5 being high, meaning critical or severe. Each risk should have an action to counteract the likelihood and reduce the impact of the risk.

No.	Description	Likelihood	Impact	Action
1	Losing data or access to data that is necessary to complete the project.	1	5	Use multiple backup solutions. Host local Git repository in a Google Drive folder.
2	Team member falls ill and is unable to work.	2	3	Ensure work is done in a timely manner, so that the impact of the temporary loss of a team member is minimized.
3	Significant changes in the scope of the project (scope creep)	2	3	React to changes in the next sprint.
4	Feature not working as intended.	2	3	Create Proof of Concept (POC) as soon as possible during a sprint.
5	Payment for Google Cloud Platform not completed in time for deployment	1	3	Consistent communication with employer to ensure inclusion in Telenor's GCP deal.
6	Intermediate deadlines are not met	1	3	Use sprints to work towards intermediary deadlines.

7	Thesis is not ready for final delivery	1	5	Meet intermediary deadlines in order to stay on schedule.
8	Final product not ready for final deadline.	2	5	Verify progress weekly with Gantt chart to stay on schedule.

All identified project risks are plotted in the heat diagram below, identified by the risk number. For each risk the values for likelihood and impact are multiplied and the resulting value signifies the total risk.

- Red (18-25): Critical risk
 - The impact of the risk will cause severe damage to the progress of the project. Will require significant counteraction to mitigate and reduce likelihood and impact.
- Yellow (8-17): Medium risk
 - The risk is notable and may cause damage, slow or alter progress. Requires counteractions to mitigate.
- Green (1-7): Trivial or low risk
 - The risk is insignificant or trivial and can easily be mitigated. The likelihood is low and the impact will only be a slight hindrance to progress.

We used an ordinal ranking system to calculate the risk for each of the aforementioned scenarios. The assigned values for each scenarios' likelihood and impact are based on our own experiences after 3 years of study and teamwork. The calculation is done by multiplying likelihood and impact, resulting in a final risk score. The distribution of which can be found in the below matrix.

As we can see in the heatmap when all our risks are plotted, there are no risks that are in the category critical and only one that is medium. Risk number 8 is related to not meeting our deadlines. To counteract this risk we will utilize our Gantt chart to make sure we are on schedule and that we are meeting intermediate milestones and deadlines. Risks related to loss of data are counteracted by hosting information and documents in online, private repositories.

LIKELIHOOD	5					
	4					
	3					
	2			2, 3, 4		8
	1			5, 6		1, 7
		1	2	3	4	5
	IMPACT					

References

- [1] Atatus, "Agile Methodology: Definition, Stages, Types, and Benefits.", June. 1, 2021. [Online]. Available: <https://www.atatus.com/glossary/agile-methodology/> . [Accessed Jan. 30, 2022.]
- [2] GeeksforGeeks, "Lean Software Development (LSD)". Oct. 13, 2021. [Online]. Available: <https://www.geeksforgeeks.org/lean-software-development-lsd/> . [Accessed Jan. 30, 2022]
- [3] GeeksforGeeks, "Difference Between Lean Development Model and Agile Development Model". Aug. 19, 2021. [Online]. Available: <https://www.geeksforgeeks.org/difference-between-lean-development-model-and-agile-development-model/?ref=rp> . [Accessed Jan. 30, 2022]

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett

Er den rett som den som skaper et åndsverk har til å fremstille eksemplarer av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater

Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater

Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn

Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring

Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: Institutt for informasjonsteknologi og elektronikk (IE)
Veileder ved NTNU: Jia-Chun Lin e-post og tlf.: jia-chun.lin@ntnu.no / 45849287
Ekstern virksomhet: Telenor Ekstern virksomhet sin kontaktperson, e-post og tlf.: Eirik Stephansen, eirik.stephansen@telenor.no , 92220222
Student: Peder Hovdan Andresen Fødselsdato: 26.02.1994
Ev. flere studenter ¹ Student: Patrick Eilert Krosby Fødselsdato: 14.04.1992
Student: Anders Christoffer Westby Fødselsdato: 25.05.1995

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato: 10.01.2022
Sluttdato: 20.05.2022

Oppgavens arbeidstitel er: Sikkerhet i Containers

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

¹ Dersom flere studenter skriver oppgave i fellesskap, kan alle føres opp her. Rettigheter ligger da i fellesskap mellom studentene. Dersom ekstern virksomhet i stedet ønsker at det skal inngås egen avtale med hver enkelt student, gjøres dette.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven². Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

<input checked="" type="checkbox"/>	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven
-------------------------------------	--

² Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

<input type="checkbox"/>	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt
--------------------------	---

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

<input checked="" type="checkbox"/>	Oppgaven skal være offentlig
-------------------------------------	------------------------------

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss	Sett dato
<input type="checkbox"/>	ett år
<input type="checkbox"/>	to år
<input type="checkbox"/>	tre år

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

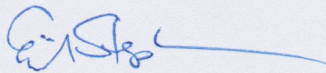
Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder:	
Dato:	
Veileder ved NTNU:	
Dato:	
Ekstern virksomhet:	Telenor Norge AS, Tech Lead IoT, Mobile Eirik Stephansen, 25.01.22 
Dato:	
Student:	Peder Ancher
Dato:	25.01.2022
Student:	Andreas Westby
Dato:	25.01.2022
Student:	Pavel Kny
Dato:	25.01.2022

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDMAL ved avtale om konfidensialitet mellom student og ekstern virksomhet i forbindelse med studentens utførelse av oppgave (master-, bachelor- eller annen oppgave) i samarbeid med ekstern virksomhet, jf. punkt 9 i standardavtale om utføring av oppgave i samarbeid med ekstern virksomhet.

Student ved NTNU: Patrick Eilert Krosby
Fødselsdato: 14.04.1992
Student ved NTNU: Peder Hovdan Andresen
Fødselsdato: 26.02.1994
Student ved NTNU: Anders Christoffer Westby
Fødselsdato: 25.05.1995
Ekstern virksomhet: Telenor Norge AS, Snarøyveien 30, 1331 Fornebu, org. nr. 976 967 631

1. Studenten skal utføre oppgave i samarbeid med ekstern virksomhet som ledd i sitt studium ved NTNU.

2. Studenten forplikter seg til å bevare taushet om det han/hun får vite om tekniske innretninger og fremgangsmåter samt drifts- og forretningsforhold som det vil være av konkurransemessig betydning å hemmeligholde for den eksterne virksomheten. Det er den eksterne sitt ansvar å sørge for å synliggjøre og tydeliggjøre hvilken informasjon dette omfatter.

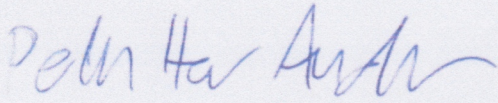
3. Studenten er forpliktet til å bevare taushet om dette i 5 år regnet fra sluttdato.

4. Kravet om konfidensialitet gjelder ikke informasjon som:

- a) var allment tilgjengelig da den ble mottatt
- b) ble mottatt lovlig fra tredjeperson uten avtale om taushetsplikt
- c) ble utviklet av studenten uavhengig av mottatt informasjon
- d) partene er forpliktet til å gi opplysninger om i samsvar med lov eller forskrift eller etter pålegg fra offentlig myndighet.

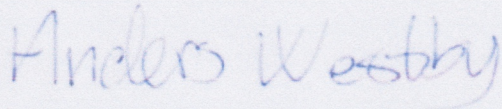
Signaturer

Student:



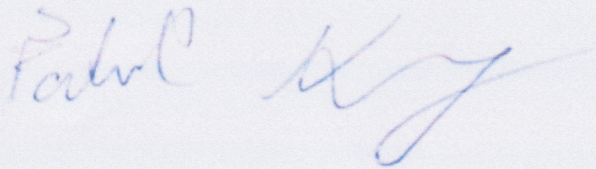
Dato: 07.02.2022

Student:



Dato: 07.02.2022

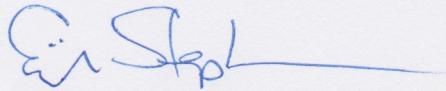
Student:



Dato: 07.02.2022

Ekstern virksomhet: Telenor Norge, Eirik Stephansen, Tech Lead IoT, Mobil

Dato: 07.02.2022

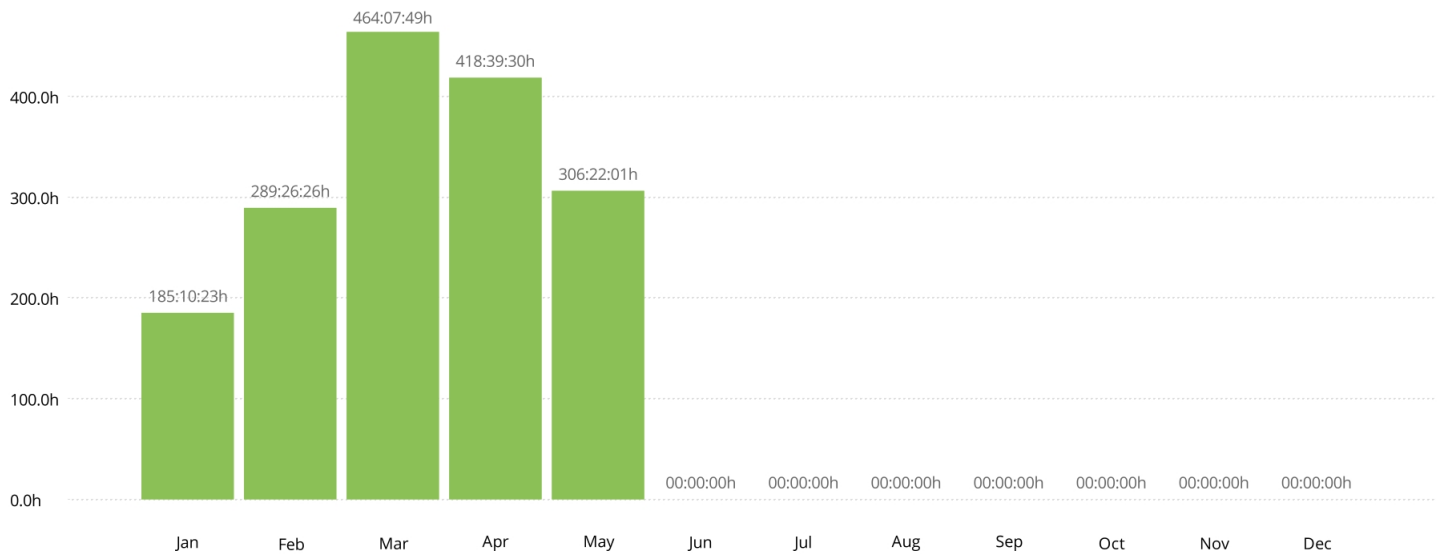


Summary report

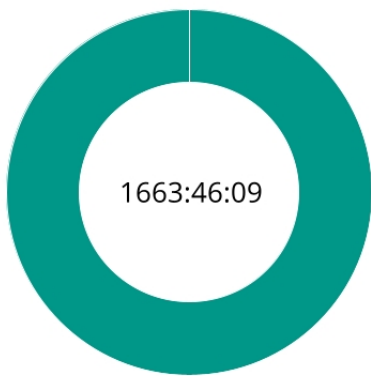
01/01/2022 - 31/12/2022



Total: 1663:46:09



Project



● Bachelor Thesis	1663:46:09	100.00%
-------------------	------------	---------

Project	Duration
Bachelor Thesis	1663:46:09

A.1 Early Report Example

docker.io/vulhub/php

Auspex

April 25, 2022

1 Trend

Mean CVSSv3 score trend for the 101 most recent reports

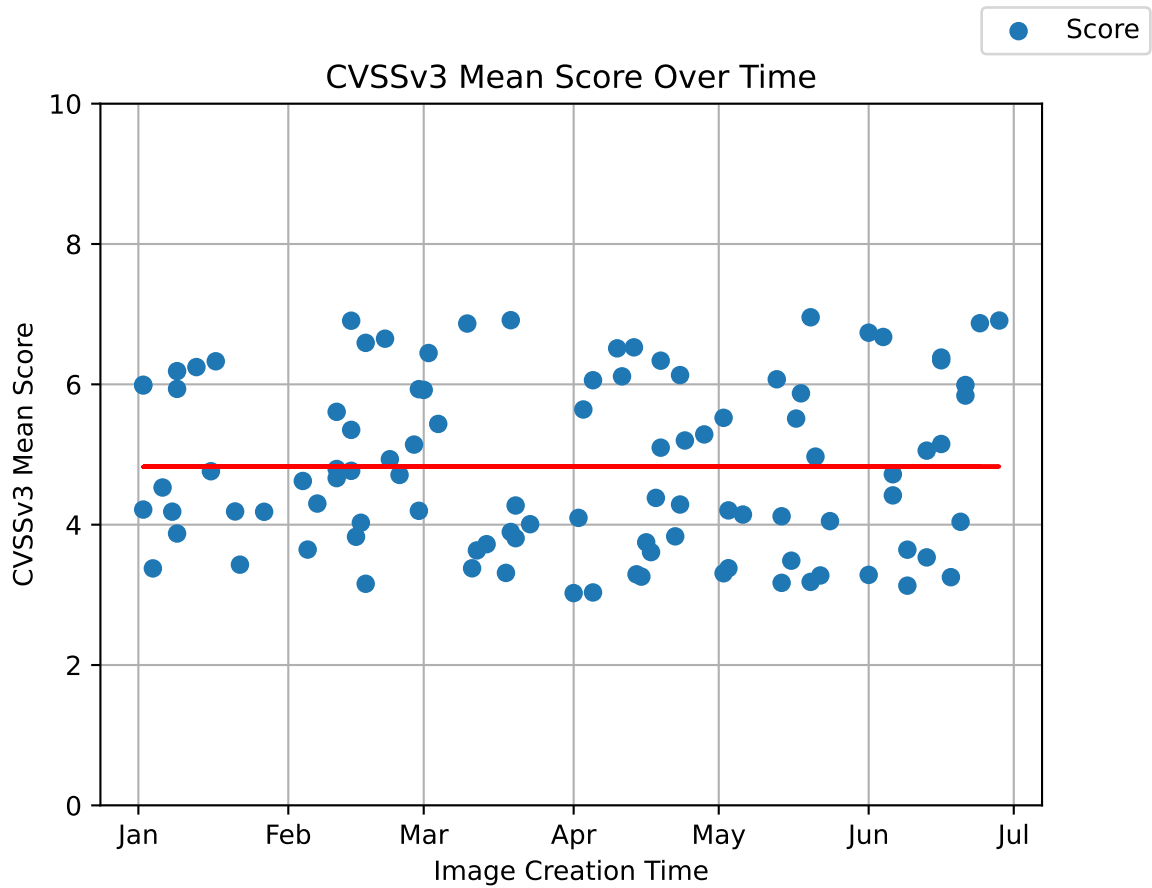


Figure 1: CVSSv3 Mean Score Over Time

2 Statistics

Median CVSS	Mean CVSS	CVSS Stdev	Max CVSS	L	M	H	C	# Vulns
6.50	6.53	1.84	9.80	88	659	469	159	1375

Where: *L* = Low (0.1 - 3.9), *M* = Medium, (4.0 - 6.9), *H* = High (7.0 - 8.9), *C* = Critical (9.0 - 10.0)

3 Top 5 Most Critical Vulnerabilities

Vulnerability	CVSS ID	CVSS Score	Severity	Upgradable
NULL Pointer Dereference	CVE-2017-7614	9.80	Critical	False
Out-of-Bounds	CVE-2014-9939	9.80	Critical	False
Out-of-bounds Write	CVE-2018-12699	9.80	Critical	False
Out-of-bounds Write	CVE-2019-12900	9.80	Critical	False
Out-of-bounds Write	CVE-2019-3822	9.80	Critical	False

4 Top 5 Most Critical Upgradable Vulnerabilities

Vulnerability	CVSS ID	CVSS Score	Severity	Upgradable
Out-of-bounds Write	CVE-2019-3822	9.80	Critical	True
Integer Overflow or Wraparound	CVE-2016-7167	9.80	Critical	True
Out-of-Bounds	CVE-2018-16839	9.80	Critical	True
Out-of-bounds Write	CVE-2019-5482	9.80	Critical	True
Out-of-bounds Write	CVE-2019-18218	9.80	Critical	True

5 Distribution of Vulnerabilities by Severity

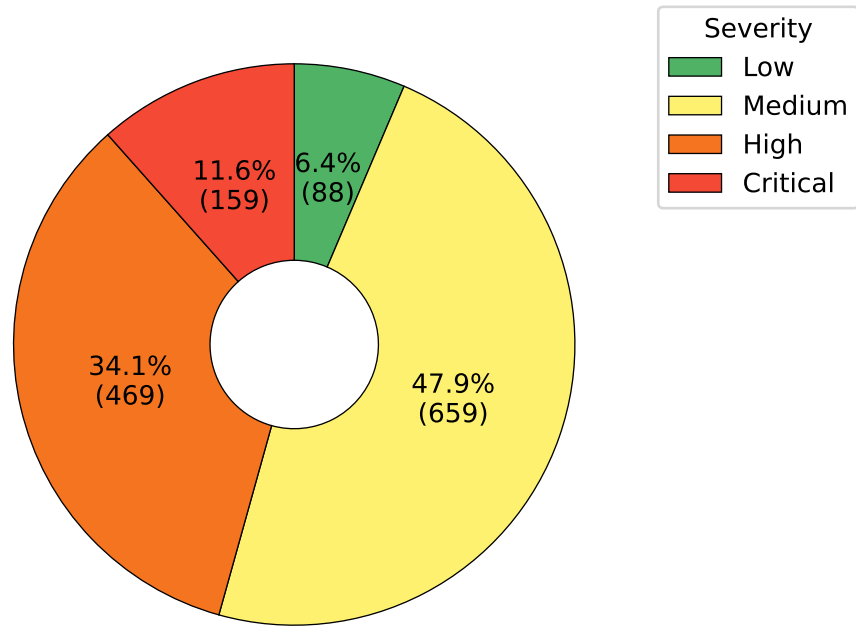


Figure 2: Distribution of Vulnerabilities by Severity

6 Age of Unpatched Vulnerabilities

The age of unpatched vulnerabilities found and their corresponding CVSSv3 scores.

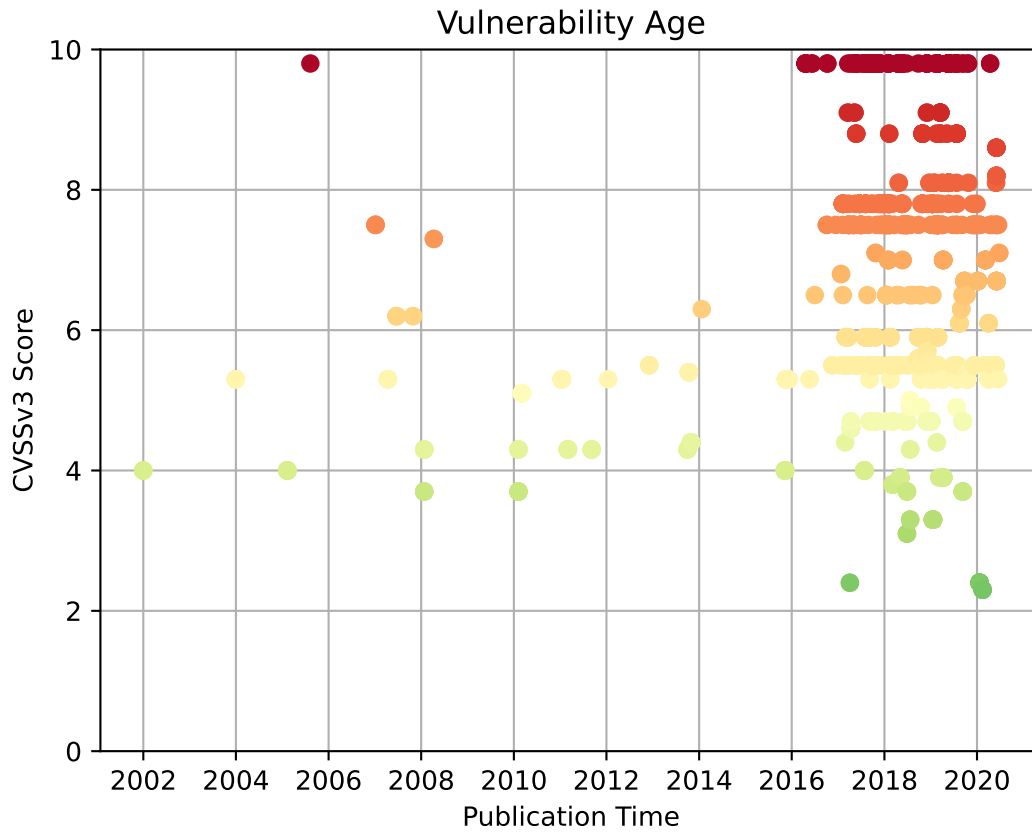


Figure 3: Age of Unpatched Vulnerabilities

A.2 Example of a Single Report

CVSSv3 Scoring System

The following intervals are used to define the severity of a vulnerability. Scoring interval is based on the CVSSv3 scoring system, rating vulnerabilities from 0.0 to 10.0 and ranking them by severity, 'Low' to 'Critical' according to their score.

Low	Medium	High	Critical
0.1 - 3.9	4.0 - 6.9	7.0 - 8.9	9.0 - 10.0

Table 2: CVSSv3 Severity Intervals

1 Image Information

Image	Created	Tags	Digest
auspex/scanner	2022-05-09 15:58:04	latest	0ba27ca7

2 Statistics

The statistics is based on the scanned image(s) and denotes the Median, Mean and Standard deviation (Stdev) score of all vulnerabilities found. Additionally it showcases the single highest score of a vulnerability for this scan. 'L', 'M', 'H' and 'C' denote the severity categories, with the corresponding number of vulnerabilities for each category. '#Vulns' denotes the total number of vulnerabilities found.

Where: L = Low (0.1 - 3.9), M = Medium, (4.0 - 6.9), H = High (7.0 - 8.9), C = Critical (9.0 - 10.0)

Median CVSS	Mean CVSS	CVSS Stdev	Max CVSS	L	M	H	C	# Vulns
6.50	6.49	1.73	9.80	58	211	200	37	506

3 CVSSv3 Mean Score Trend

Mean CVSSv3 score trend for the 85 most recent reports. Each data point in blue represents a previous scan. The current scan is represented by a green dot. The red line is the trend and shows whether the security state is improving or worsening over time.

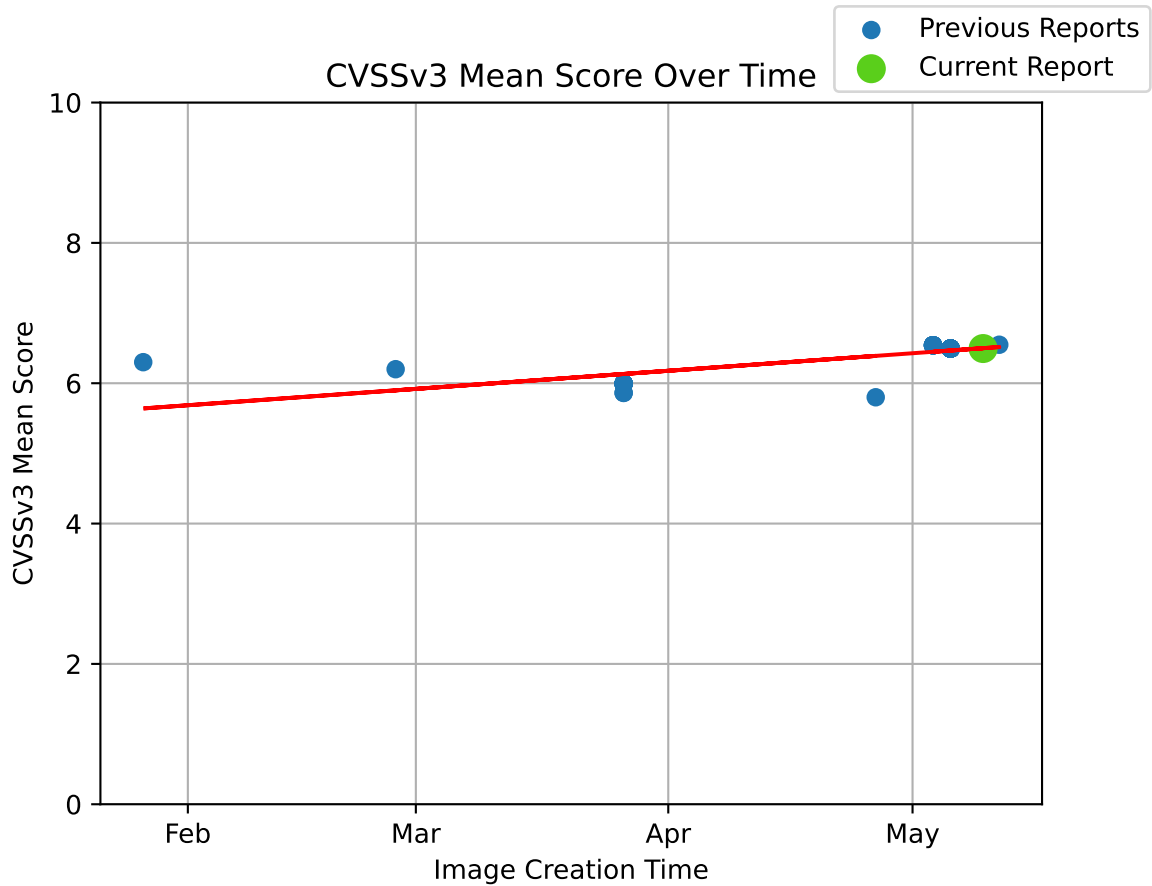


Figure 1: CVSSv3 Mean Score Over Time

4 Most Critical Vulnerabilities

Lists the found vulnerabilities with highest CVSS scores. The CVSS ID is a hyperlink to official documentation for that vulnerability. 'Upgradeable' denotes whether the found vulnerability has a known fix ie. a new version of a package or library.

Vulnerability	CVSS ID	CVSS Score	Severity	Upgradeable
Use After Free	CVE-2021-33574	9.80	Critical	False
Buffer Overflow	CVE-2022-23219	9.80	Critical	False
Buffer Overflow	CVE-2022-23218	9.80	Critical	False
Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False
Use After Free	CVE-2021-33574	9.80	Critical	False

5 Most Critical Upgradable Vulnerabilities

Lists the found vulnerabilities with highest CVSS scores. The CVSS ID is a hyperlink to official documentation for that vulnerability. 'Upgradeable' denotes whether the found vulnerability has a known fix ie. a new version of a package or library. Only vulnerabilities that are upgradable are listed.

Vulnerability	CVSS ID	CVSS Score	Severity	Upgradeable
CVE-2022-1271	CVE-2022-1271	3.90	Low	True

6 Distribution of Vulnerabilities by Severity

The pie chart shows the distribution of vulnerabilities by severity. Severities are grouped by colour, as described by the legend. Each slice of the pie denotes the percentage of the total, and sum of vulnerabilities for each severity.

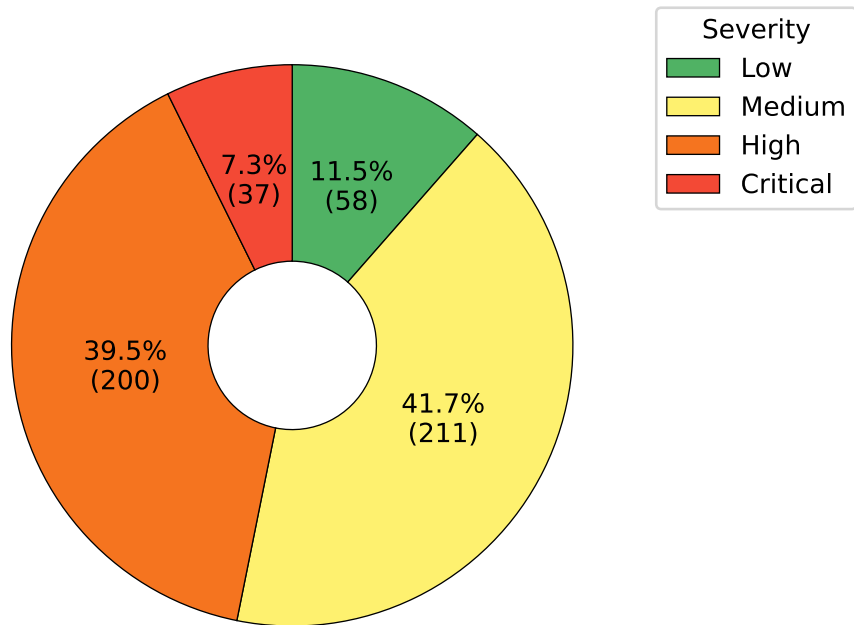


Figure 2: Distribution of Vulnerabilities by Severity

10 All Critical Vulnerabilities

Lists all discovered critical vulnerabilities. 'Upgradeable' denotes whether the found vulnerability has a known fix ie. a new version of a package or library. Year represents the publication year of the vulnerability.

Vulnerability	CVSS ID	CVSS Score	Severity	Upgradeable	Year
Use After Free	CVE-2021-33574	9.80	Critical	False	2021
Buffer Overflow	CVE-2022-23219	9.80	Critical	False	2022
Buffer Overflow	CVE-2022-23218	9.80	Critical	False	2022
Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False	2019
Use After Free	CVE-2021-33574	9.80	Critical	False	2021
Buffer Overflow	CVE-2022-23219	9.80	Critical	False	2022
Buffer Overflow	CVE-2022-23218	9.80	Critical	False	2022
Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False	2019
CVE-2019-9893	CVE-2019-9893	9.80	Critical	False	2019
CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Use After Free	CVE-2020-11656	9.80	Critical	False	2020
Use After Free	CVE-2020-11656	9.80	Critical	False	2020
Use After Free	CVE-2020-11656	9.80	Critical	False	2020
CVE-2005-2541	CVE-2005-2541	9.80	Critical	False	2005
Integer Overflow or Wraparound	CVE-2021-35942	9.10	Critical	False	2021
Integer Overflow or Wraparound	CVE-2021-35942	9.10	Critical	False	2021

A.3 Example of a Aggregate Report

CVSSv3 Scoring System

The following intervals are used to define the severity of a vulnerability. Scoring interval is based on the CVSSv3 scoring system, rating vulnerabilities from 0.0 to 10.0 and ranking them by severity, 'Low' to 'Critical' according to their score.

Low	Medium	High	Critical
0.1 - 3.9	4.0 - 6.9	7.0 - 8.9	9.0 - 10.0

Table 2: CVSSv3 Severity Intervals

1 Images in This Report

Image	Created	Tags	Digest
auspex/reporter	2022-05-09 16:01:08	latest	e614e96b
auspex/scanner	2022-05-09 15:58:04	latest	0ba27ca7

2 Statistics

The statistics is based on the scanned image(s) and denotes the Median, Mean and Standard deviation (Stdev) score of all vulnerabilities found. Additionally it showcases the single highest score of a vulnerability for this scan. 'L', 'M', 'H' and 'C' denote the severity categories, with the corresponding number of vulnerabilities for each category. '#Vulns' denotes the total number of vulnerabilities found.

Where: L = Low (0.1 - 3.9), M = Medium, (4.0 - 6.9), H = High (7.0 - 8.9), C = Critical (9.0 - 10.0)

Image	Median CVSS	Mean CVSS	CVSS Stdev	Max CVSS	L	M	H	C	# Vulns
auspex/reporter	6.50	6.56	1.63	9.80	26	161	146	15	348
auspex/scanner	6.50	6.49	1.73	9.80	58	211	200	37	506

3 CVSSv3 Mean Score Trend

Mean CVSSv3 score trend for the 41 most recent reports. Each data point in blue represents a previous scan. The current scan is represented by a green dot. The red line is the trend and shows whether the security state is improving or worsening over time.

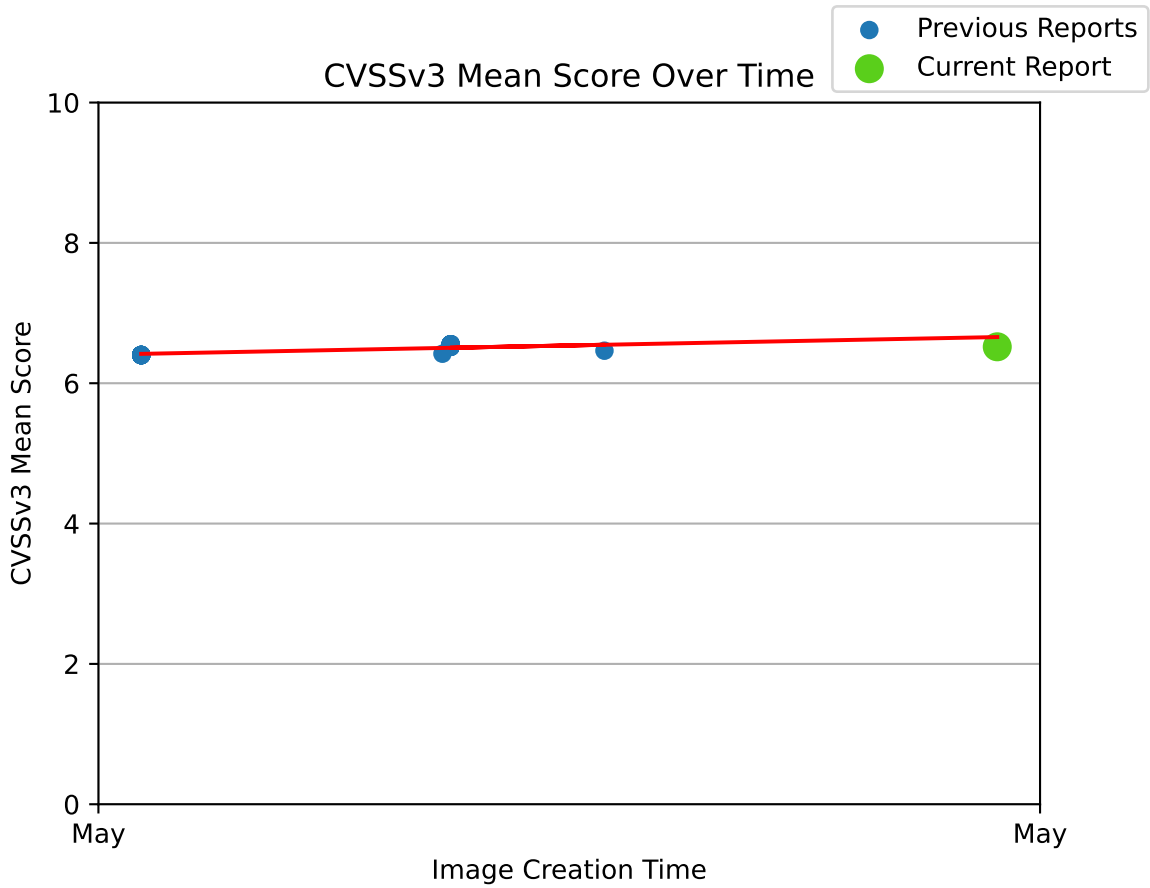


Figure 1: CVSSv3 Mean Score Over Time

4 Most Critical Vulnerabilities by Image

Lists the found vulnerabilities with highest CVSS scores. The CVSS ID is a hyperlink to official documentation for that vulnerability. 'Upgradeable' denotes whether the found vulnerability has a known fix ie. a new version of a package or library.

Image	Vulnerability	CVSS ID	CVSS Score	Severity	Upgradeable
auspex/reporter	Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False
auspex/reporter	Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False
auspex/reporter	Out-of-bounds Write	CVE-2020-25412	9.80	Critical	False
auspex/reporter	Out-of-bounds Write	CVE-2020-25412	9.80	Critical	False
auspex/reporter	Out-of-bounds Write	CVE-2017-17479	9.80	Critical	False
auspex/scanner	Use After Free	CVE-2021-33574	9.80	Critical	False
auspex/scanner	Buffer Overflow	CVE-2022-23219	9.80	Critical	False
auspex/scanner	Buffer Overflow	CVE-2022-23218	9.80	Critical	False
auspex/scanner	Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False
auspex/scanner	Use After Free	CVE-2021-33574	9.80	Critical	False

5 Most Critical Upgradeable Vulnerabilities by Image

Lists the found vulnerabilities with highest CVSS scores. The CVSS ID is a hyperlink to official documentation for that vulnerability. 'Upgradeable' denotes whether the found vulnerability has a known fix ie. a new version of a package or library. Only vulnerabilities that are upgradeable are listed.

Image	Vulnerability	CVSS ID	CVSS Score	Severity	Upgradeable
auspex/reporter	Uncontrolled Search Path Element	CVE-2022-24765	7.80	High	True
auspex/scanner	CVE-2022-1271	CVE-2022-1271	3.90	Low	True

6 Distribution of Vulnerabilities by Severity

The pie chart shows the distribution of vulnerabilities by severity. Severities are grouped by colour, as described by the legend. Each slice of the pie denotes the percentage of the total, and sum of vulnerabilities for each severity.

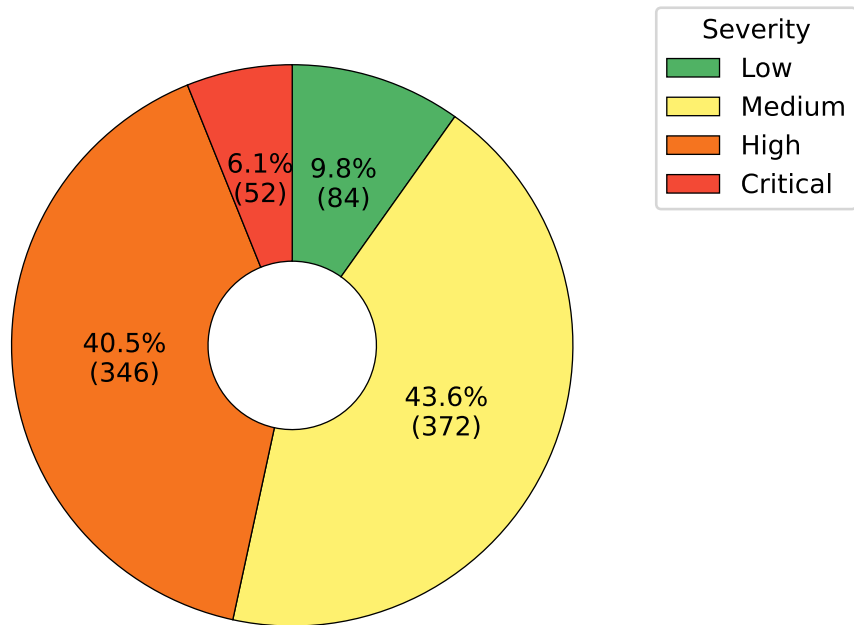


Figure 2: Distribution of Vulnerabilities by Severity

7 Age of Unpatched Vulnerabilities

The age of unpatched vulnerabilities found and their corresponding CVSS scores. Each dot represents a vulnerability and is color coded, following the same style as the pie chart. The age of a vulnerability is based on its publication time.

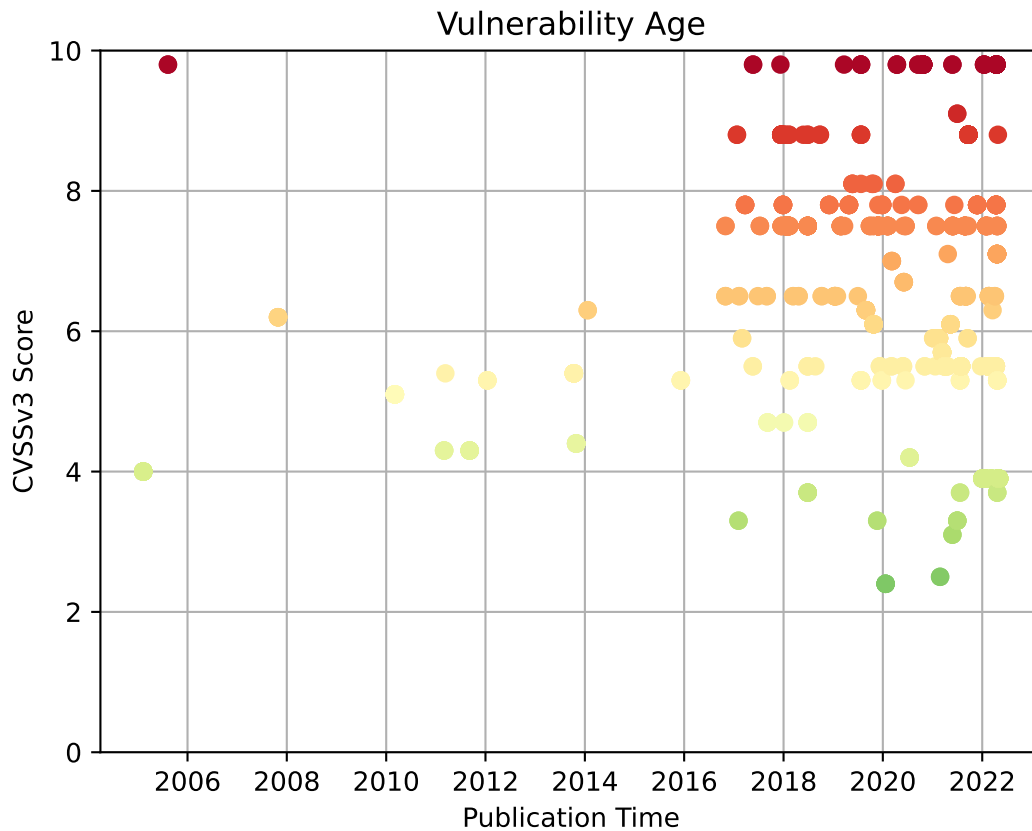


Figure 3: Age of Unpatched Vulnerabilities

8 Exploitable Vulnerabilities

No exploitable vulnerabilities found.

9 Distribution of Exploitable Vulnerabilities by Severity

No vulnerabilities found.

10 All Critical Vulnerabilities

Lists all discovered critical vulnerabilities. 'Upgradeable' denotes whether the found vulnerability has a known fix ie. a new version of a package or library. Year represents the publication year of the vulnerability.

Image	Vulnerability	CVSS ID	CVSS Score	Severity	Upgradeable	Year
Image	Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False	2019
Image	Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False	2019
Image	Out-of-bounds Write	CVE-2020-25412	9.80	Critical	False	2020
Image	Out-of-bounds Write	CVE-2020-25412	9.80	Critical	False	2020
Image	Out-of-bounds Write	CVE-2017-17479	9.80	Critical	False	2017
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	CVE-2005-2541	CVE-2005-2541	9.80	Critical	False	2005
Image	Out-of-bounds Read	CVE-2017-9117	9.80	Critical	False	2017
Image	Out-of-bounds Read	CVE-2017-9117	9.80	Critical	False	2017
Image	Use After Free	CVE-2021-33574	9.80	Critical	False	2021
Image	Buffer Overflow	CVE-2022-23219	9.80	Critical	False	2022
Image	Buffer Overflow	CVE-2022-23218	9.80	Critical	False	2022
Image	Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False	2019
Image	Use After Free	CVE-2021-33574	9.80	Critical	False	2021
Image	Buffer Overflow	CVE-2022-23219	9.80	Critical	False	2022
Image	Buffer Overflow	CVE-2022-23218	9.80	Critical	False	2022
Image	Out-of-Bounds	CVE-2019-1010022	9.80	Critical	False	2019
Image	CVE-2019-9893	CVE-2019-9893	9.80	Critical	False	2019
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022

Continued on Next Page

Image	Vulnerability	CVSS ID	CVSS Score	Severity	Upgradable	Year
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	CVE-2020-27619	CVE-2020-27619	9.80	Critical	False	2020
Image	Arbitrary Command Injection	CVE-2015-20107	9.80	Critical	False	2022
Image	Use After Free	CVE-2020-11656	9.80	Critical	False	2020
Image	Use After Free	CVE-2020-11656	9.80	Critical	False	2020
Image	Use After Free	CVE-2020-11656	9.80	Critical	False	2020
Image	CVE-2005-2541	CVE-2005-2541	9.80	Critical	False	2005
Image	Integer Overflow or Wraparound	CVE-2021-35942	9.10	Critical	False	2021
Image	Integer Overflow or Wraparound	CVE-2021-35942	9.10	Critical	False	2021

A.4 Auspex Survey

Auspex survey

This survey is conducted as part of a bachelor thesis at NTNU Gjøvik in the course DCSG2900.

The task is given by Telenor.

The purpose of this survey is to gather feedback on a report generated from a container vulnerability scan. The scan is performed using Snyk, a container image scan tool, and the output data is used to generate a human-readable document with key information about the findings of the scan.

Description of the report sections:

1. The trend diagram shows a trend line for CVSS mean score of previous scans to give an insight of whether the security is improving or worsening over time.
2. Statistics gives additional important information about the scan and the score denoting how vulnerable the scanned container image is.
3. The table for top 5 most critical vulnerabilities shows the most critical vulnerabilities.
4. The table for top 5 most critical upgradeable vulnerabilities shows the most critical vulnerabilities that can be upgraded, and thus remove that specific vulnerability.
5. The pie chart shows the distribution and number of found vulnerabilities, and their corresponding severity level.
6. The scatter plot show the age distribution of vulnerabilities, how long ago they were found/published and their CVSS score.

If you have any question, please contact: anderscw@stud.ntnu.no

Information about the bachelor course: <https://www.ntnu.no/studier/emner/DCSG2900/2021#tab=omEmnet>

*Required

1. Are you a developer? *

Mark only one oval.

Yes

No

Full
Example
Report

The full report can be viewed at: https://drive.google.com/file/d/1YNKD9XNRN88ZGUyQe-vKP6iRGJltWGw_/view?usp=sharing

However, each relevant section of the document is shown above their corresponding question, so there is no need to keep the document open during the survey.

1. Trend

1 Trend

Mean CVSSv3 score trend for the 101 most recent reports

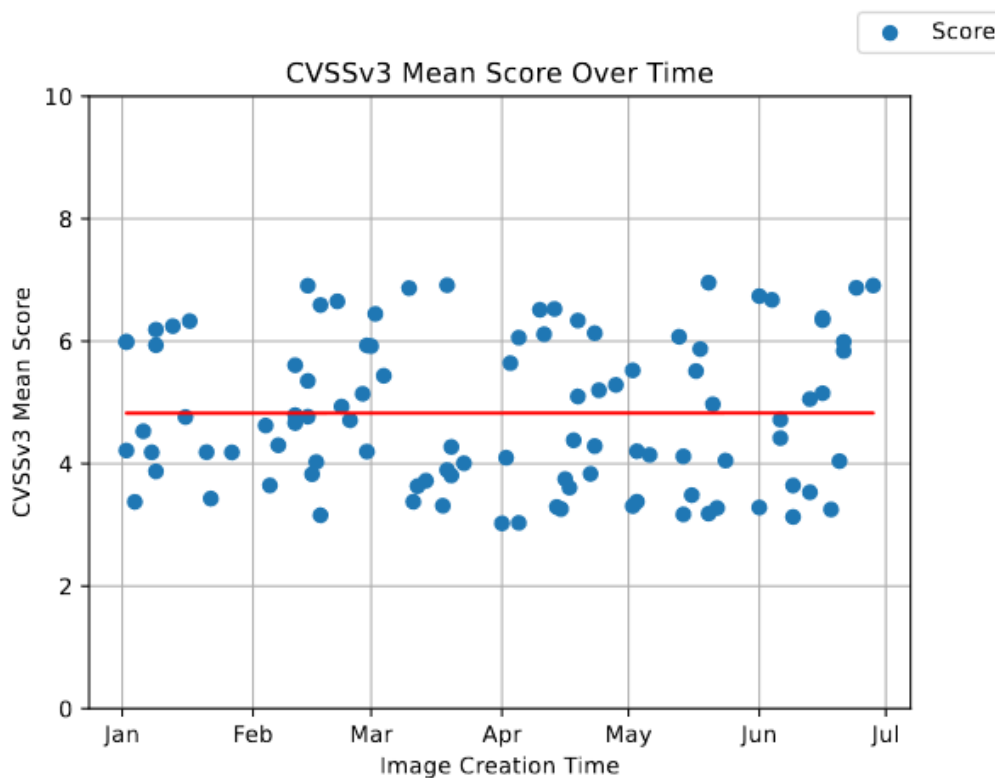


Figure 1: CVSSv3 Mean Score Over Time

2. To what degree does the trend diagram give you an overview of the security trend over time?

Mark only one oval.

	1	2	3	4	5	
Low	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	High

3. Does the trend line help you better understand the security trend over time?

2. Statistics

2 Statistics

Median CVSS	Mean CVSS	CVSS Stdev	Max CVSS	L	M	H	C	# Vulns
6.50	6.53	1.84	9.80	88	659	469	159	1375

Where: *L* = Low (0.1 - 3.9), *M* = Medium, (4.0 - 6.9), *H* = High (7.0 - 8.9), *C* = Critical (9.0 - 10.0)

4. To what degree does the statistics highlight the important information of a scan?

Mark only one oval.

	1	2	3	4	5	
Low	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	High

5. Is there any other information or statistics you would like to see? Is there anything you feel is unnecessary?

4. Top 5 Most Critical Vulnerabilities

3 Top 5 Most Critical Vulnerabilities

Vulnerability	CVE ID	CVSS Score	Severity	Upgradable
NULL Pointer Dereference	CVE-2017-7614	9.80	Critical	False
Out-of-Bounds	CVE-2014-9939	9.80	Critical	False
Out-of-bounds Write	CVE-2018-12699	9.80	Critical	False
Out-of-bounds Write	CVE-2019-12900	9.80	Critical	False
Out-of-bounds Write	CVE-2019-3822	9.80	Critical	False

6. To what degree does the "Top 5 Most Critical Vulnerabilities"-table help you understand what the 5 most critical vulnerabilities are?

Mark only one oval.

1	2	3	4	5	
Low	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	High

7. Is there anything you feel is missing or unclear?

5. Top 5 Most Critical Upgradeable Vulnerabilities

4 Top 5 Most Critical Upgradeable Vulnerabilities

Vulnerability	CVSS ID	CVSS Score	Severity	Upgradeable
Out-of-bounds Write	CVE-2019-3822	9.80	Critical	True
Integer Overflow or Wraparound	CVE-2016-7167	9.80	Critical	True
Out-of-Bounds	CVE-2018-16839	9.80	Critical	True
Out-of-bounds Write	CVE-2019-5482	9.80	Critical	True
Out-of-bounds Write	CVE-2019-18218	9.80	Critical	True

8. To what degree does the "Top 5 Most Critical Upgradeable Vulnerabilities" -table help you understand what the 5 most critical upgradeable vulnerabilities are?

Mark only one oval.

	1	2	3	4	5	
Low	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	High

9. Is there anything you feel is missing or unclear?

5. Pie Chart

5 Distribution of Vulnerabilities by Severity

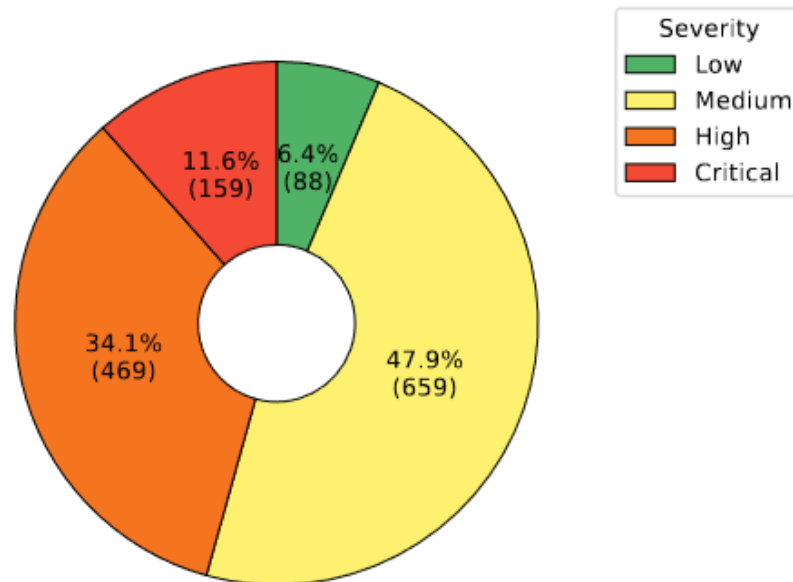


Figure 2: Distribution of Vulnerabilities by Severity

10. To what degree does the pie chart give you an overview of the found vulnerabilities and their distribution?

Mark only one oval.

	1	2	3	4	5	
Low	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	High

11. Is there anything you feel is missing or unclear?

6. Scatter Plot

6 Age of Unpatched Vulnerabilities

The age of unpatched vulnerabilities found and their corresponding CVSSv3 scores.

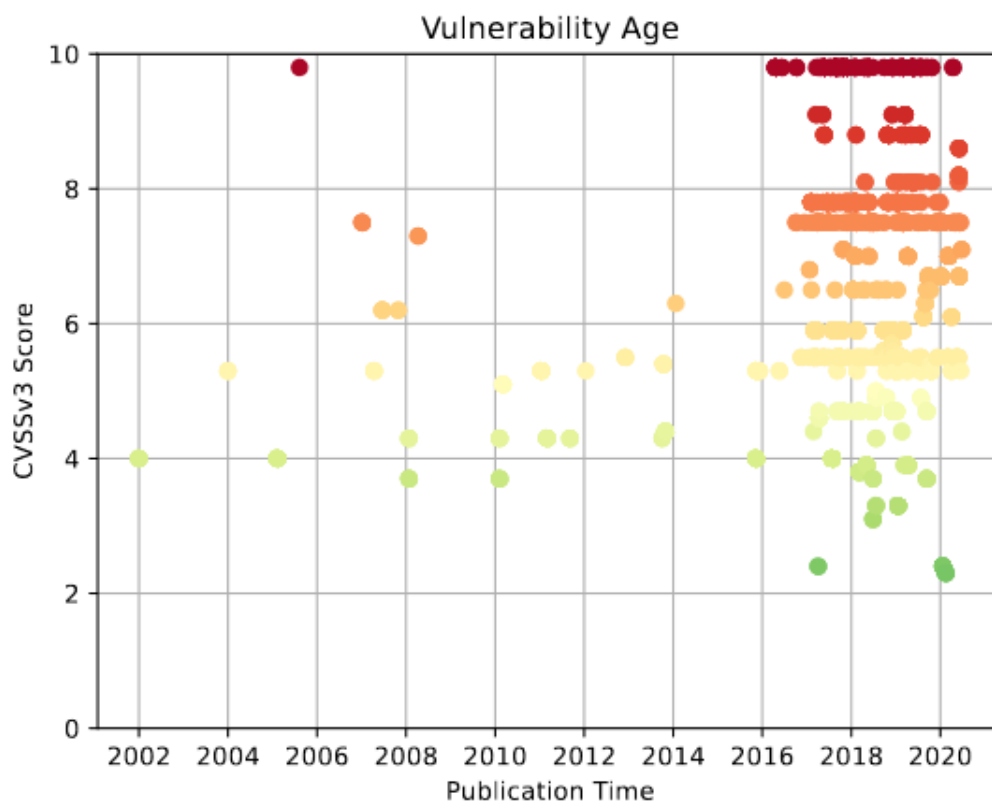


Figure 3: Age of Unpatched Vulnerabilities

12. To what degree does the scatter plot help you understand the age of detected vulnerabilities?

Mark only one oval.

	1	2	3	4	5	
Low	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	High

13. Any other thoughts on the scatter plot?

Feedback

14. Is there any information or specific sections you feel is missing?

15. Do you have any other feedback or suggestions?

This content is neither created nor endorsed by Google.

Google Forms

A.5 Auspex Survey Result

Auspex survey

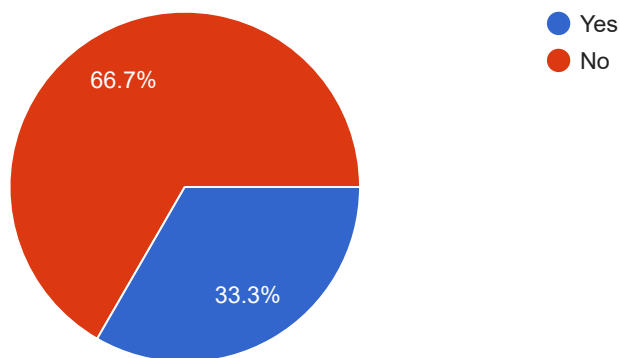
3 responses

[Publish analytics](#)

Are you a developer?

 Copy

3 responses



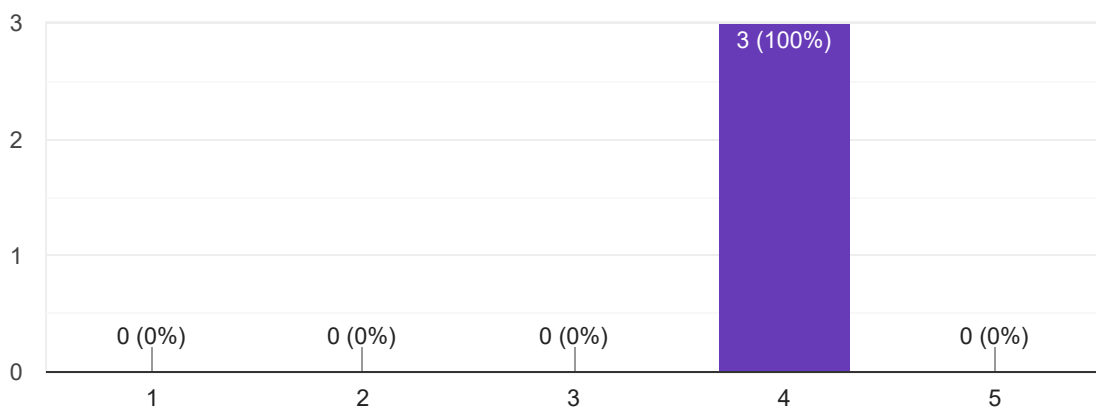
Full Example Report

1. Trend

To what degree does the trend diagram give you an overview of the security trend over time?

 Copy

3 responses



Does the trend line help you better understand the security trend over time?

3 responses

yes

somewhat. the jitter/scattered nature of the data is more interesting from an analysis PoV than the trend though

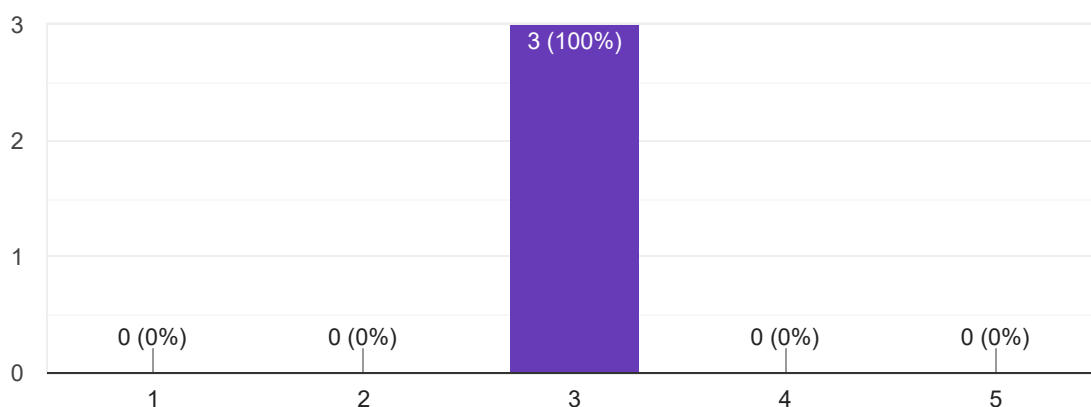
Yes

2.Statistics

To what degree does the statistics highlight the important information of a scan?



3 responses



Is there any other information or statistics you would like to see? Is there anything you feel is unnecessary?

3 responses

Maybe show percentage clearer, and values /diff from previous period

recency/cvss area chart

Short explanation of CVSS

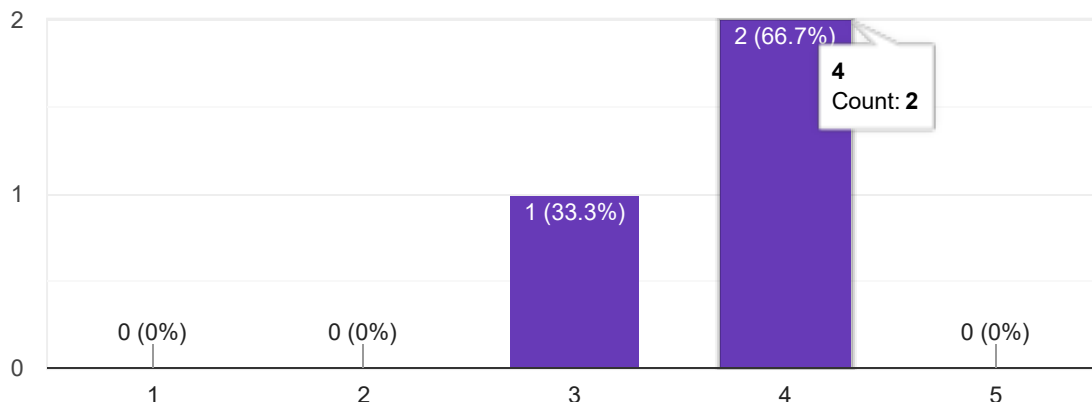
4. Top 5 Most Critical Vulnerabilities



To what degree does the "Top 5 Most Critical Vulnerabilities"-table help you understand what the 5 most critical vulnerabilities are?



3 responses



Is there anything you feel is missing or unclear?

3 responses


better legends/descriptions

cluster the data by type. in most codebases the top vulnerabilities will have the same title/main type

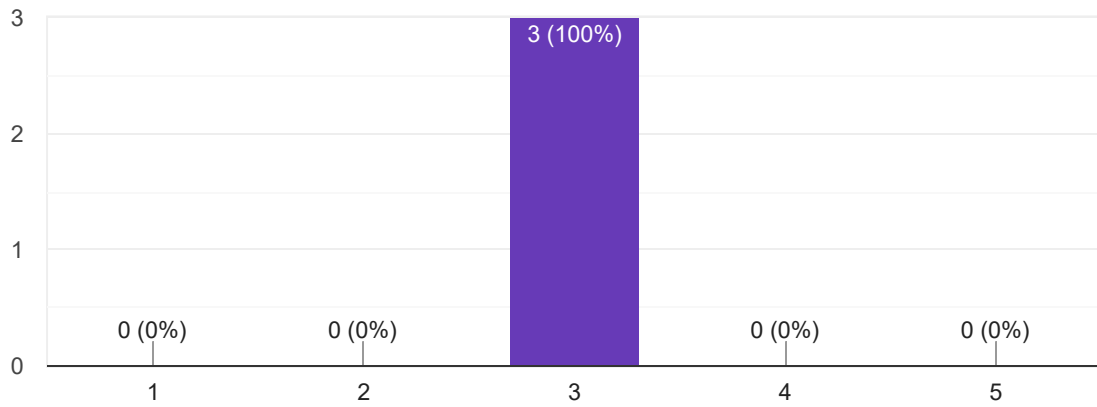
Maybe a link to each vulnerability so you can click to read more

5. Top 5 Most Critical Upgradeable Vulnerabilities



To what degree does the "Top 5 Most Critical Upgradeable Vulnerabilities" -table help you understand what the 5 most critical upgradeable vulnerabilities are?  Copy

3 responses



Is there anything you feel is missing or unclear?

3 responses

need better explanations

refine upgradeable to patchable without breaking change

CVSS ID gives little value

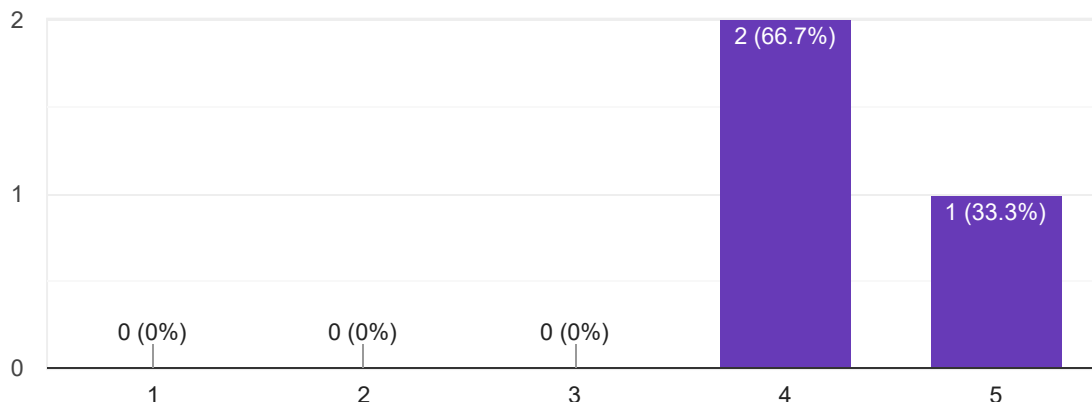
5. Pie Chart



To what degree does the pie chart give you an overview of the found vulnerabilities and their distribution?



3 responses



Is there anything you feel is missing or unclear?

1 response

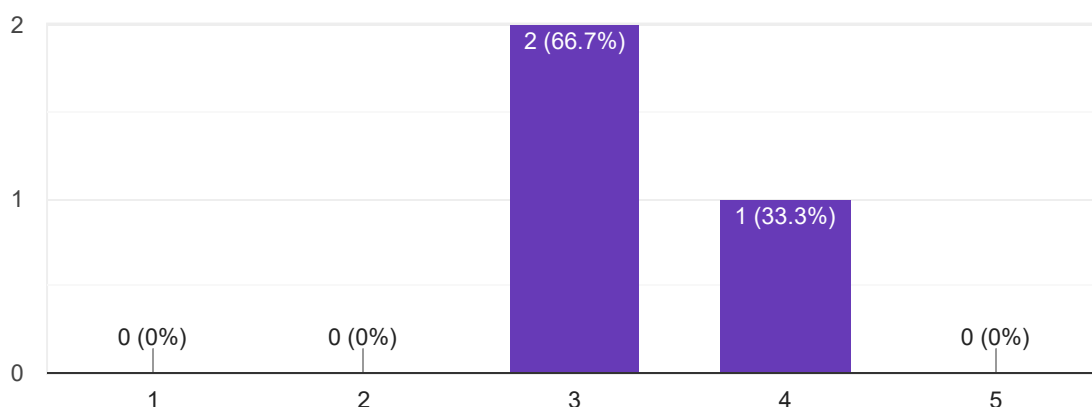
second pie with only vulns with known exploits please

6. Scatter Plot

To what degree does the scatter plot help you understand the age of detected vulnerabilities?



3 responses



Any other thoughts on the scatter plot?

3 responses

need description on what timing means

at the risk of making the report time-based, making the graph be years since current date will make it more digestable

Maybe to much data?

Feedback

Is there any information or specific sections you feel is missing?

0 responses

No responses yet for this question.

Do you have any other feedback or suggestions?

1 response

write descriptions in a "for dummies" level so very easy to understand

This content is neither created nor endorsed by Google. [Report Abuse](#) - [Terms of Service](#) - [Privacy Policy](#)

Google Forms



Appendix B

Schemas

B.1 POST /reports Response Schema

This is the schema for responses from the POST /reports endpoint. Its "reports" value is an array of multiple reports, and "aggregate" is an instance of a report representing the aggregate report for all reports contained in the "reports" array.

See Section B.3 for just the report schema.

```
{
  "reports": [
    {
      "id": "string",
      "image": {
        "imageSizeBytes": "string",
        "layerId": "string",
        "mediaType": "string",
        "tag": [
          "string"
        ],
        "timeCreatedMs": "2022-05-18T10:20:43.895Z",
        "timeUploadedMs": "2022-05-18T10:20:43.895Z",
        "digest": "string",
        "image": ""
      },
      "timestamp": "2022-05-18T10:20:43.895Z",
      "cvss": {
        "mean": 0,
        "median": 0,
        "stdev": 0,
        "min": 0,
        "max": 0
      },
      "vulnerabilities": {
        "critical": 0,
        "high": 0,
        "medium": 0,
        "low": 0
      },
      "report_url": "string",
      "aggregate": false,
      "schema_version": "1",
      "historical": false,
      "updated": "2022-05-18T10:20:43.895Z",
      "upgrade_paths": [
        "string"
      ],
      "dockerfile_instructions": [
        "string"
      ]
    }
  ],
  "aggregate": {
    "id": "string",
```

```
    "image": {
      "imageSizeBytes": "string",
      "layerId": "string",
      "mediaType": "string",
      "tag": [
        "string"
      ],
      "timeCreatedMs": "2022-05-18T10:20:43.895Z",
      "timeUploadedMs": "2022-05-18T10:20:43.895Z",
      "digest": "string",
      "image": ""
    },
    "timestamp": "2022-05-18T10:20:43.895Z",
    "cvss": {
      "mean": 0,
      "median": 0,
      "stdev": 0,
      "min": 0,
      "max": 0
    },
    "vulnerabilities": {
      "critical": 0,
      "high": 0,
      "medium": 0,
      "low": 0
    },
    "report_url": "string",
    "aggregate": true,
    "schema_version": "1",
    "historical": false,
    "updated": "2022-05-18T10:20:43.895Z",
    "upgrade_paths": [
      "string"
    ],
    "dockerfile_instructions": [
      "string"
    ]
  },
  "message": "",
  "failed": [
    {
      "scan_id": "string",
      "error": "string"
    }
  ]
}
```


B.2 Scan Metadata Schema

This is the schema for the JSON representation of scan metadata that is passed between and also returned by the different microservices.

```
{
  "image": {
    "imageSizeBytes": "string",
    "layerId": "string",
    "mediaType": "string",
    "tag": [
      "string"
    ],
    "timeCreatedMs": "2022-05-20T08:20:40.777Z",
    "timeUploadedMs": "2022-05-20T08:20:40.777Z",
    "digest": "string",
    "image": ""
  },
  "backend": "string",
  "id": "string",
  "timestamp": "2022-05-20T08:20:40.777Z",
  "url": "string",
  "blob": "string",
  "bucket": "string"
}
```

B.3 Report Metadata Schema

This is the schema for the JSON representation of report metadata returned by the Reporter service.

```
{
  "id": "string",
  "image": {
    "imageSizeBytes": "string",
    "layerId": "string",
    "mediaType": "string",
    "tag": [
      "string"
    ],
    "timeCreatedMs": "2022-05-20T08:23:32.225Z",
    "timeUploadedMs": "2022-05-20T08:23:32.225Z",
    "digest": "string",
    "image": ""
  },
  "timestamp": "2022-05-20T08:23:32.225Z",
  "cvss": {
    "mean": 0,
    "median": 0,
    "stdev": 0,
    "min": 0,
    "max": 0
  },
  "vulnerabilities": {
    "critical": 0,
    "high": 0,
    "medium": 0,
    "low": 0
  },
  "report_url": "string",
  "aggregate": false,
  "schema_version": "1",
  "historical": false,
  "updated": "2022-05-20T08:23:32.225Z",
  "upgrade_paths": [
    "string"
  ],
  "dockerfile_instructions": [
    "string"
  ]
}
```

B.4 Reporter Query Parameter Schema

The following is the OpenAPI definition for the query parameters available for the GET /reports endpoint of Reporter and API Gateway.

```
{
  "/reports":{
    "get":{
      "summary":"Get Reports",
      "description":"Fetch reports from the database.",
      "operationId":"get_reports_reports_get",
      "parameters":[
        {
          "required":false,
          "schema":{
            "title":"Image",
            "type":"string",
            "default":""
          },
          "name":"image",
          "in":"query"
        },
        {
          "required":false,
          "schema":{
            "title":"Aggregate",
            "type":"boolean",
            "default":false
          },
          "name":"aggregate",
          "in":"query"
        },
        {
          "required":false,
          "schema":{
            "title":"Ge",
            "anyOf":[
              {
                "maximum":10,
                "minimum":0,
                "type":"number"
              },
              {
                "maximum":10,
                "minimum":0,
                "type":"integer"
              }
            ]
          },
          "name":"ge",
          "in":"query"
        }
      ]
    }
  }
}
```

```
    "required": false,
    "schema": {
      "title": "Le",
      "anyOf": [
        {
          "maximum": 10,
          "minimum": 0,
          "type": "number"
        },
        {
          "maximum": 10,
          "minimum": 0,
          "type": "integer"
        }
      ]
    },
    "name": "le",
    "in": "query"
  },
  {
    "required": false,
    "schema": {
      "allOf": [
        {
          "$ref": "#/components/schemas/CVSSField"
        }
      ],
      "default": "mean"
    },
    "name": "field",
    "in": "query"
  },
  {
    "required": false,
    "schema": {
      "title": "Limit",
      "exclusiveMinimum": 0,
      "type": "integer"
    },
    "name": "limit",
    "in": "query"
  },
  {
    "required": false,
    "schema": {
      "allOf": [
        {
          "$ref": "#/components/schemas/OrderOption"
        }
      ],
      "default": "newest"
    },
    "name": "order",
    "in": "query"
  }
}
```

```
}  
  }  
    }  
  }  
}
```

Appendix C

Minutes of Meeting

Date(25.02.2022) - Title

Place: Online - Teams

Participants

- Kelly
- All team members

Agenda (short)

- Information
- Clarification of task description
- Project plan
- References and time keeping

Summary

Firstly we were informed that we should upload our future weekly status report on teams in the designated folder instead of sending them by email.

The next discussion was about clarifying our understanding about the project task and how our task is not about GDPR as one might be led to believe by the background section in the project description document, but rather about the security aspect. Our team and another both have Telenor as our employer and both tasks share the same background description. Their task is rather about concerns around GDPR laws and customer information while ours is about evaluating security in container applications. We are not supposed to deal with live system or user data regarding GDPR.

Kelly would like us to make a Gantt chart to show our projected schedule for the project period and to use this to continually evaluate our progress. Our next deadline is January 31st when we need to deliver our finished project plan. Kelly would like us to send it to her as soon as possible, uploading it on Teams.

Regarding our question about references, Kelly explained how we should use references and footnotes. We can use footnotes with a link to the source for example gitlab.com. If we use or quote external content or information we should use references. References should follow IEEE format, using reference numbers.

Regarding questions on timekeeping we only need to log the time as a whole. It is not necessary to separate the tasks with tags. Kelly was unsure if the timekeeping/ logging is a mandatory task of the project, but this was clarified by us that it is needed and expected. Information about these requirements can be found on Blackboard under 'Dokumenter' as well as the information page for the bachelor course.

As we stand now we are currently on track for the project plan and should be able to complete it soon. We will go through the finished project plan in the next meeting together. Next meeting is wednesday february 2nd at 1pm

Date: 02.02.2022

Place: Online -Teams

Participants

- Kelly
- All team members

Agenda (short)

- Feedback on project plan
- Look at requirement, functional and non-functional
- Section 4.2 in project plan , suggestions for diagram
- Wordlist
- Project plan approved

Summary

Project plan is well written according to Kelly and includes the necessary sections and explanations. To view her comments we need to download the document as a PDF. Reiterates that the project plan is more for us to use as a guideline during the project. The project plan should be included as part of the thesis, in the appendix.

Going through the project plan and comments, what we should improve, as well as other suggestions that Kelly has for us.

We should have more non-functional requirements. Kelly suggests checking out rqtest.com on the subject for inspiration for non- functional requirements. The exact link for this is in Kelly's comments on the project plan. We should focus on requirements that can be met and evaluated by us.

Section 4.2 in the project plan could use some clarification for the diagram outlining the structure of our application. We had numbered each element and listed the explanation for these. Kellys suggestion was that we don't need to number and explain simple things in the diagram such as the pdf and json icon. We should however improve how we represent the output from the analyzer, not as 2 separate objects, one for the raw json and one for the pdf generation. We should rather represent this in another way to show that both the raw json log and the pdf generation come from the same information. Some of our explanations also need some improvements to clarify how our pdf reports are generated and what tools are used

We also agreed that we should have a wordlist in our thesis for certain terms to make sure that these terms are unambiguous and that the reader will understand them and our use of these terms

Project plan is approved as it stands and we just need to update it based on Kelly's comments to improve the quality.

Date: 16.02.2022

Place: Online (Teams)

Participants

- Kelly
- All team members

Agenda (short)

- POC
- Changing cloud platform?
- Make a sketch for scan report
- Scheduling reports

Summary

Going through what we have done with Kelly and showcasing our POC and some functionality.

During a presentation with Telenor on the 15th we learned that the different teams use mostly Azure and some use GCP, and that there is an overarching goal of migrating mostly to Azure with a sub-goal of having a multicloud. This might mean we need to change from GCP to Azure due to Teleor migrating to Azure, but this will be further discussed on Thursday with Eirik. Kelly thinks we should stick with GCP due to us having started using it already and have made some cloud functions and code implementations.

May consider providing an adaptation that can target both Azure and GCP but this is currently out of scope.

Going through how we modelled the output of a Snyk scan in the application with further implementation details.

Something we lack a clear picture of is how the reports that will be made should look and how we convey the information found. We came to an agreement that we should make a sketch of how a scan report should look, both a single scan and summary report of multiple scans. The sketch can be made in multiple iterations until we are happy with it, and then we can start the actual modelling and coding using the sketch as a template.

One of the thoughts we have for the application is that the scans could be scheduled, for example daily, to scan and generate reports automatically without user input so that every day can be started with a fresh report of all container instances. We need to decide on how frequently this should be done so as to not generate reports with no new information.

Caching previous scans if no new vulnerabilities are found may be the answer.

Date:23.02.2022

Place: Teams (Online)

Participants

- Kelly
- All team members

Agenda (short)

- Decision regarding cloud platform
- Report function
- Application

Summary

Explained some of the results and decisions from our meeting with Eirik last thursday. We will continue with GCP and look into Azure if we are able to in the given timeframe.

Talked about our plan for the report functionality and how we present this information. Some additional features are required for the report function to work properly.

Talking about how we structure the report, one report per container and a larger summary for many containers. Additionally we also have made a quick sketch on how we think a report should look.

As it stands now the application works. We can send data and store it in firestore. A POST request sends the entire payload as a json document and stores this in a collection. We do however need a new collection that parses and stores the output of logs that can then be used for the pdf report so some additional functionality is required.

Thesis writing course is next week and Kelly will also help us with this subject.

Date 02.03.2022

Place: Teams (online)

Participants

- Kelly
- All team members

Agenda (short)

- Learnings from Telenor
- Working on matplotlib
- Thesis writing course

Summary

Talked about the last meeting with telenor (24th February) showcasing their backend for Mitt Spor and what we learned. Was very insightful and gave us some insight into how we should look at structuring our services on GCP. We also received access to the Mitt Spor developments container registry so we can use this to test our report generator later on.

Worked on learning about matplotlib, creating and generating pdf documents in the latex format automatically from a POST request to the service.

We need to make a finished product of the report, call it version 1, that we can send for evaluation to Telenor and receive some feedback on formatting and information that is presented

Sebastien, backend expert at Telenor and developer on Mitt Spor,, offered to look at some of the outputs that we generate from a container scan and tell us what information is relevant or not as he has experience using kube-bench and snyk.

Plan next is to make an example report that we can give to Eirik and he can give us feedback on what is needed or not which will be very helpful for us.

Showcasing that we are able to produce a simple pdf document for now. On another note Kelly thinks it would be a good idea to make our database query-able so it's possible to search for specific documents. We'll keep that in mind.

Kelly wants to offer a course for thesis writing and thus the next meeting will be a course on thesis writing.

Date 23.03.22

Place: Online (Teams)

Participants

- Kelly
- All team members

Agenda (short)

- Had meeting with Eirik
- Have implemented a queryable database
- Questionnaire

Summary

We had a meeting earlier in the day with Eirik and received some feedback on our report layout which we shared with Kelly.

The thesis writing is going okay and we are making progress, but it is difficult.

We have a queryable database in place as Kelly asked for previously, but there are some limitations in Firestore in regards to composite queries that we need to work around. It is not feasible to make composite indexes for every single field and combo of fields that could possibly be queried. What we think is a better solution is to limit the query to search for a specific image and then query it client-side.

What we have working now is querying using http queries. This method is not necessarily very user friendly, but the reports themselves will be. We don't think this method will be used often, but that the main purpose of these reports is to generate them automatically on a schedule.

We should try to get some feedback from Telenor on what they think about the querying method for reports and the database.

Lastly Kelly suggested we should prepare a questionnaire that we can present to Telenor to assess the quality of the report and receive feedback on the layout and information presented. We will aim to have a draft of this ready before next Wednesday's meeting so Kelly can give us feedback on the questions.

Date 06.04.2022

Place: Online (Teams)

Participants

- Kelly
- All team members

Agenda (short)

- Survey
- Nicer frontpage, update the glossary, small fixes
- Additions to chapter 2
- Non-functional requirements

Summary

First of all, regarding the survey, Kelly has a good point when she says we should try to look at reducing the amount of text before each section, and put it at the start instead, so that the taker of the survey can read everything first and then answer the questions. We will update this.

Kelly thinks we should try to make a nicer frontpage, which we agree on. We will make an effort for this before we hand in the final version.

We should go through and add all abbreviations to the glossary, and describe the abbreviation in the text to make sure the reader can understand most things.

Just some small things that we should fix. Amongst these are: Delete citation for the origin of the auspex name. We must cite Snyk when we first introduce it, and give a better explanation for matplotlib and pylatex.

Some things we should add to chapter 2.

We should explain what GCP is, either as a separate section, or as part of another sections where that would make sense. Perhaps there are some tools that we use that need to be introduced and explained if they are relevant to understanding everything. If so, they should be added to chapter 2. Otherwise they can be added to either implementation or the development process chapter.

Figure 3.1, application overview, should be given a better name and otherwise be updated to show the entrypoint better. Naming of the components should also be updated to be more consistent with other figures.

The introduction to chapter 3.1 should be rewritten to be more precise.

The nonfunctional requirements we have written down so far don't need to be so specific. We should rather change the title to something non specific, focusing on what the

requirement will achieve instead, such as availability, reliability , etc. Then we can provide a more specific description or explanation below. The reason for this, as Kelly rightly suggest, is that for example our time requirements may change when deployed on GCP and depending on how much data is scanned.

Remove the survey requirement or change it to something related to readability or user-friendliness instead. Something rather related to the readability of a generated report.

Figure 4.1 needs a better caption, and some of the names should be more concrete to make both them and the figure easier to understand.

No meeting next week due to easter.

Date 20.04.22

Place: Online (Teams)

Participants

- Kelly
- Peder, Anders
- Absent: Patrick

Agenda (short)

- Querying scans and reports
- Trend diagram and suggestions
- Email workflow for reports
- Handing in thesis for more feedback in may
- Get the survey out

Summary

Worked on querying the report database, but explained that this is not a main use case. We have added some basic querying on mean score and timestamp.

We have worked on the trend diagram requested by both Eirik and Sebastien at Telenor which showcases the mean score over time for an image, or alternatively for an aggregate scan showcasing the mean score of an aggregate scan ie. the mean score of all the images scanned. For this demonstration we showcased the diagram with mock data as a proof of concept, where each blue dot represents a mean score from a scan for an image throughout its different versions. The red trend line is added to showcase if the score is increasing or decreasing over time.

Kelly had some input to make the descriptions of the diagram better. The X-axis caption should be changed and be explanatory, perhaps to "images evolve over time (2022)".

In addition a description to emphasise what the diagram is showing and the significance of the scores on the y-axis should be added

As for the dataset we will be using to populate our database with scan data and reports, we are able to access Telenors MittSpor container registry and scan these containers, about 11. We can then scan single images, and the whole container registry to create single and aggregate reports.

Kelly suggests that we use some of the snippets of actual reports to showcase and explain in our thesis, as well as add an example of a complete report in the appendix..

Kelly raised a question about how we want to deliver a report. The solution we are leaning towards is to have a scheduled workflow on GCP to deliver reports via email, preferably on a weekly, or perhaps daily, schedule.

We should be able to tie everything together very soon, all the services to create a generated report.

Kelly would expect a more complete thesis to be handed to her a week before delivery, 13th may , so she can give feedback and we are able to revise before final delivery at noon May 20th.

Everything should be complete by the 20th and after this we should focus on the presentation in early June, including how we want to present the application. The presentation might be a good use case for a website.

The scheduling of scans is the primary use case for the application and manual scans is a secondary use case, meaning we are not facilitating a nice user experience with GUI or a website as of now. Google cloud scheduler can be used to schedule the scans and generation of reports. The scheduler basically sends an HTTP request to our API.

The survey and an example report needs to be handed out very soon. The missing part is the example report which we should be able to complete within a week. We will not be able to facilitate technical testing of the application as there is simply not enough time. The focus is primarily on the generated reports as that is in fact the most important end product and the inner workings of the application is not necessarily that important.

We will contact Eirik to help facilitate distribution of the survey and give them a full week to fill out the survey. We hope for a minimum of 5 results. We should however ask Eirik how many people are involved in his department and other personnel to get an idea of how many would read the reports and look at them in a real world scenario.

Date 27.04.22

Place: Online (Teams)

Participants

- Kelly
- All team members

Agenda (short)

- Example report
 - Trend diagram
 - Statistics
 - Top 5
 - CVSS id
 - Scatter plot (vuln age)
- Survey

Summary

The survey as it stands is okay and we will get some decent information from the questions.

The example report could use more extensive explanations for the figures and tables in general. In general could have a better explanation for the scoring and intervals we are using, which is the CVSS scoring system. This could perhaps be added to the top of the document.

The trend diagram should highlight that this is the score of the same image over time. Additionally, the current scan should be highlighted on the diagram, perhaps as a dot in a different colour to the other data points.

Kelly suggested it could be useful to have a section with some information about the image being scanned, such as name, date of current version, versionnumber etc.

It would be better to make it clear that statistics and all other sections following are for the current scan. For the statistics section we should evaluate if the mean, median and standard deviation fields are necessary. We should add an explanation of each field anyways.

When it comes to the top 5 tables we need to make it clear that its sorted by CVSS score, and the vulnerabilities are ranked as critical. Top 5 in itself is quite arbitrary in terms of why we are choosing to show only 5, but we don't think in a real world environment there will be many more critical vulnerabilities than 5. We also have a version with all critical vulnerabilities listed at the end of the document.

Kelly had a very good suggestion to try to add a hyperlink for the CVSS id in all tables so that one can read more about that particular vulnerability should one want to.

As stated before, we should make it clear if a figure or table is for the current scanned image or not. This also goes for the pie chart showing the vulnerability distribution, make it clear that it is for this current scanned image.

The scatter plot itself is fine, but could use some extra information. We could perhaps have a table critical and high vulnerabilities sorted by oldest to newest. The purpose of the scatter plot is to show that there are long standing vulnerabilities.

We should add an explanation of the colour gradient used in the scatter plot and what significance the colour has.

We will try to update the example report according to feedback from Kelly and update the survey and example, and send a new version to Telenor. We can probably omit the first question "are you a dev?" as there is not much value to it probably. Finally focus on getting as many respondents as possible to the survey as the deadline is may 6th.

Date 04.05.22

Place: Online (Teams)

Participants

- Kelly
- All team members

Agenda (short)

- Planning
- Deployment
- Survey shared
- Plan going forward

Summary

Had an internal meeting within the team about a plan for what needs to be done. Should be able to deliver chapter 1,2,3 for review by the end of this week.

We have deployed the project fully on gcloud and set up a CI/CD pipeline. Apart from some minor issues, everything seems to work and deploy correctly.

Survey sent out and shared with Telenor

Not expecting too many respondents due to only being shared with relevant teams. This is Eirik's team at Telenor and the team for Mitt Spor in Poland. We should ask exactly how many are involved.

Once the survey results are in we will look at them and improve, We think it's best to look at everything together, in addition to Kelly's feedback from last week and make the corresponding changes and improvements.

Kelly is available at campus Gjøvik if needed.

We should have a few chapters for Kelly to review at the end of the week.

Date: 17.02.2022 - Telenor

Place: Online(Teams)

Participants

- Eirik Stephansen - Telenor
- All team members

Agenda (short)

- Skyplattform
- Mangler noen rettigheter på GCP
- Demonstrasjon

Summary

Det store spørsmålet som dukket opp under presentasjonen på tirsdag var valg av skyplattform. Telenor har en multicloud tilnærming som betyr at de baserer seg på flere forskjellige skyplattformer. Blant annet bruker Infrastruktur teamet Azure, mens andre bruker GCP mm. Det kan tenkes at de helhetlig velger å flytte til Azure. For oss ble vi enige om at vi i fellesskap skal fortsette å bruke GCP i og med at vi har startet å bruke det allerede. Eirik foreslo at han kunne sette opp et møte med sjef for backend for å forklare litt hvordan Telenor jobber på den fronten, og kan antagelig komme med noen innspill til oss rundt oppgaven og arbeidet.

Vi kan legge til støtte for Azure og GCP. Noe av det vi har hittil er GCP spesifikt, men kan tilpasses. Kan være positivt for læring å bli kjent med flere plattformer og deres funksjonalitet.

Vi mangler tilgang til lagring i GCP gjennom Telenor sin plan da vi manglet noen rettigheter. Eirik fikset det med en gang etter litt troubleshooting og vi har nå det vi behøver av rettigheter for å bruke Firestore og Firebase.

En liten gjennomgang med Eirik over hva vi har gjort hvor vi blant annet viser frem hvordan vi har tenkt den helhetlige strukturen på applikasjonen skal se ut. Vi demonstrere også lagringstrukturen, hvordan vi lagrer output fra en scan med metadata og rå json. Videre gjennomgår vi modelleringen av en Snyk scan og noe av informasjonen vi kan hente fra en slik scan, blant annet utregning av score, hvilke sårbarheter som er mest kritiske og som går igjen oftest. Vi demonstrerer en POST av en scan til cloud function som lagrer json resultatet.

Planen er å sette opp en nytt møte neste uke for flere spørsmål

Date: 23.03.22 - Telenor

Place: Online (Google Meet)

Participants

- Eirik Stephansen
- All team members

Agenda (short)

- Showing the example report sketch
- Feedback
- New meeting monday

Summary

The meeting started with us showing two examples of how we think a report should look like, using a sketch made in diagrams.net and a pdf-document generated using pylatex. We were not able to generate the actual pie chart and scatter plot that we wanted to show, but instead used the sketch to show our intention.

Eirik remarked that he would like to see a diagram showing the trend over time to compare previous scans with the current. This will be helpful to gauge whether security is improving or not over time. We agree that it should show trends over time for score.

Showing firestore and what we are currently storing. Of the data that we are storing we could use cvss mean score to show trend over time, comparing previous scans to a new scan.

Eirik suggest we could ask Sebastien for some feedback as well and was happy to set up a meeting on monday the 28th at 1030

Date 30.03.22 - Telenor

Place: Google Meets

Participants

- Sebastien Renauld (Telenor)
- All team members

Agenda (short)

- Metrics and trends
- Missing params
- UpgradePath
- Exploitability and old vulns

Summary

We currently have a working pipeline for the application where we can trigger scans.

What metrics are we storing and planning to show?

Eirik mentioned in the last meeting that he would like to have trends to show if score is increasing or decreasing over time.

Sebastien would like to see trends on both the severity of vulnerabilities and the exploitability, meaning whether or not a vulnerability has been exploited or not.

One issue we have is that we are not logging the age of an image. Sebastien suggests using the container build date to keep track of this.

Is there anything missing?

Sebastien thinks we already have the basics completed in regards to what data we are storing. We are already storing all the scoring parameters and grouping them.

He would like a filter for whether a severity is exploitable or not as this is useful information to keep track of and be aware of.

'UpgradePath' is a must have as this helps showing how a vulnerability can be fixed, and how to fix it. He also suggested that we should use the param 'upgradeable' which is more reliable, rather than 'isPatchable' which doesn't work

'References' is also a useful param that we should store. This was omitted to save space but we will reintroduce this as by Sebastien's suggestion as he thinks it's useful to have.

Other useful information is 'packageFixedInVersion', showing in which version a package was fixed, and 'DockerFileInstruction', showing the exact command to upgrade a vulnerable package. The instructions only use the apt package manager, which is not relevant if you are using another distribution that doesn't have apt.

The key thing is to find a reliable way to know if something can be patched or if there are any immediate fixes.

Something else that we should be aware of according to Sebastien is theoretical versus actual exploitability, meaning whether or not there is an actual exploit for a given vulnerability.

Another suggestion is to find for how long an old high or critical vulnerability has been out. If it's more than ~ 3 months old, the exploit usually has been exploited and there might very well be ready made scripts that can be used by ill meaning individuals. We should try to sort this out and be able to look at a glance.

Seb will see if he has more containers that we can scan that might have more vulnerabilities to give us more data and understanding.

To summarise the capabilities. We can trigger a scan automatically and we can possibly trigger through a workflow with some work. To trigger on events such as updated image or new images would be useful for a pipeline, and on a schedule. This can be achieved using webhooks