Security Document

Contents

[Introduction 4](#_Toc188121219)

[Risk Analysis 4](#_Toc188121220)

[Assets 4](#_Toc188121221)

[Threats 4](#_Toc188121222)

[Risk Matrix 5](#_Toc188121223)

[OWASP TOP 10 compliant 7](#_Toc188121224)

[Security Implementations 8](#_Toc188121225)

[Secure Coding Practices 8](#_Toc188121226)

[Cloudflare 9](#_Toc188121227)

[Secret Manager 12](#_Toc188121228)

[Development environment 12](#_Toc188121229)

[Snyk scan 12](#_Toc188121230)

[Private network in Google Cloud 13](#_Toc188121231)

[Jenkins Credentials 13](#_Toc188121232)

[Users Authenticate via google OAuth2 14](#_Toc188121233)

[SAST 14](#_Toc188121234)

[GKE Security Posture 15](#_Toc188121235)

[Container Image scanning 20](#_Toc188121236)

[Cloud monitoring 21](#_Toc188121237)

[Communication 22](#_Toc188121238)

[Cloud Firewall 22](#_Toc188121239)

[Cloud IDS 23](#_Toc188121240)

[IAM Accounts 24](#_Toc188121241)

[Extras 25](#_Toc188121242)

[White source 25](#_Toc188121243)

[Aqua 25](#_Toc188121244)

[Contrast 26](#_Toc188121245)

[Conclusion 26](#_Toc188121246)

# Introduction

In this document I will go through the current security mechanisms I have in place to protect my application’s data and extra solutions that could be added but aren’t due to costs or time restrictions. I will also focus on a Risk Analysis for my application and its compliance with OWASP TOP 10.

# Risk Analysis

## Assets

List of components within my application that need protection:

* User Data (name, emails, messages, caves they belong, permissions)
* Application Infrastructure(API endpoints, WebSocket Servers, databases, microservices, Kubernetes)
* Communication(WebSocket Connections, API communication, interservice communication)
* Compliance-related data (GDPR related data in this case name, email. messages)

## Threats

For identifying threats, I will mainly combine the STRIDE framework with the listed assets.

* Altering users’ messages in transit
* Impersonate user to send messages
* Delete user messages
* Overload API Servers
* Overload WebSocket Servers
* General Privilege escalation
* Privilege escalation within a cave.
* Access databases directly
* Access of unauthorized user data
* Altering user data
* Unauthenticated and unauthorized access to API endpoints

## Risk Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Likelihood | Impact | Risk Level | Mitigation |
| Unauthorized message edit | Low | High | High | RBAC, permission verification, log and monitoring for this type of access |
| Unauthorized message view | Low | High | High | RBAC, permission verification, log and monitoring for this type of access |
| User impersonation when sending messages | Low | High | High | RBAC, permission verification, log and monitoring for this type of access |
| Unauthorized User message deletion | Low | High | High | RBAC, permission verification, log and monitoring for this type of access |
| DDoS/DoS attack on API servers | Medium | High | Critical | Use rate limiting and request analysis. Implement WAFs |
| DDoS/DoS attack on WebSocket servers | Medium | High | Critical | Use rate limiting and request analysis. Implement WAFs |
| Privilege escalation within a cave | Low | High | Critical | RBAC, permission verification, log and monitoring for this type of access |
| Unauthorized access to databases | Low | High | Critical | Ensure no database is exposed to the internet. Strong Authentication and Authorization. Encryption |
| Unauthorized access to user data | Low | High | Critical | RBAC, permission verification, log and monitoring for this type of access. |
| Cryptographic exploitation | Medium | High | High | Keep all cryptographic libraries up to date. Detect outdated cryptographic libraries by security analysis |
| Vulnerable library exploitation | Medium | High | High | Keep all libraries up to date. Library scans. Logging and monitoring in place |
| CI/CD tempering | Low | High | High | Ensure the CI/CD environment is secure. Store secrets in secret management tools. Use dedicated agents for tasks. Use signed commits |
| Cloud environment access | Medium | High | High | Enforce MFA, RBAC, conditional access policies, audit logs, and least privilege for admin roles. |

# OWASP TOP 10 compliant

Broken Access Control: I have Authentication and Authorization implemented throughout all services, always verifying identity whenever user data is accessed.

Cryptographic failures: I make use of the most recent encryption algorithms and cryptographic libraries. On top of that I check for vulnerabilities within these libraries so if any arises, I will know and will patch it.

Injection: All my inputs are sanitized, and I make use of ORM technologies so normal injection is impossible.

Insecure Design: I apply SDLC practices on my project minimizing possible security risks.

Security misconfiguration: Every component that I use and control I make user has the correct configurations applied. When it comes to cloud components, I configure them based on the provider recommendations.

Vulnerable Components: During the pipeline process all external libraries are checked for any known vulnerabilities, with this every library will always be updated to latest version minimizing the attack vectors.

Identification and authentication failures: For authentication I make use of Google SSO with OAuth2 and then generate proprietary JWT using the latest and recommend encryption signing algorithms.

Software and data integrity failures: My pipeline runs a Snyk scan on the code checking for any vulnerability. The application doesn’t interact with any external sources that are not verified.

Security logging and Monitoring failures: Currently I have in place Google Cloud monitoring, but this will still have improvements because I don’t have yet a fully built robust monitoring and logging system for the services. In addition, I have the monitoring provided by Cloudflare which is a great help for analyzing and filtering unwanted requests.

Server-side request forgery: My application doesn’t provide any functionality that would be vulnerable to this attack.

# Security Implementations

## Secure Coding Practices

Examples of secure coding practices throughout my system:

A screen shot of a computer program

Description automatically generated

*No hardcode of any sensible credentials*

*A screenshot of a computer program

Description automatically generated*

*Server Input Validation example*

*A screen shot of a computer code

Description automatically generated*

*Example of Access control checks*

These are just a few examples of secure coding practices that I have throughout some services. These practices spread throughout the whole system.

## Cloudflare

Cloudflare is my first layer of security being the interaction point users go through. I’m using Cloudflare’s free plan which comes with the following:

* Unmetered DDoS Protection
* Universal SSL certificate
* Managed Ruleset
* Monitoring Analytics
* WAF

All these services are managed by Cloudflare, but they also provide customization specially in the WAF since I can add more restrictions to the WAF if I think there is something very specific to my application that needs to be checked.

A screenshot of a computer

Description automatically generated

*Traffic analysis dashboard where I can see every request a filter IP’s out if necessary.*

A screenshot of a computer

Description automatically generated

*Cloudflare’s managed WAF rules.*

A screenshot of a computer

Description automatically generated

*Bot traffic protection.*

A close up of text

Description automatically generated

*Managed DDoS protection.*

## Secret Manager

A screenshot of a computer

Description automatically generated

I make use of google secret manager to store all the necessary secrets, this way they are never exposed and can only be accessed at runtime.

## Development environment

When it comes to development environment, I run everything on my local machine for the code development part.

When it comes to the CI/CD pipeline all servers are running on VM’s on an environment managed by Fontys that can only be accessed if access is given by the environment administrator.

Like the CI/CD environment, the code is all stored in Fontys secured GitLab servers that require Fontys credentials to be viewed.

## Snyk scan

As one of the steps of the pipeline a Snyk scan is ran on all services checking for vulnerable libraries, blocking the pipeline execution if any is found.

A screenshot of a computer program

Description automatically generated

*Snyk scan from one service on the pipeline*

Unfortunately, I didn’t take screenshots but, in the end of 2024, there were a lot of vulnerabilities published which made my previous running tests to have 10+ outdated libraries which required update on all services which was a very tiresome job.

## Private network in Google Cloud

All my resources within google cloud are in a private network with the only network exposed resource being the Load balancer that connects to the Kubernetes ingress controller.

## Jenkins Credentials

All necessary credentials to run the pipeline are stored in the Jenkins credential manager. This keeps the credentials secure and invisible to the pipeline execution.

A screenshot of a computer

Description automatically generated

*Jenkins credentials list*

Above are all kinds of credentials and secrets that are injected when the pipeline runs setting up any environment where they are required.

## Users Authenticate via google OAuth2

All users of my application authenticate via Google SSO with OAuth2, all the logic is also handled by the spring security framework which is an industry standard and is a secure option as of the day of writing.

## SAST

All my code through a SonarQube analysis to check for any security issues.

A screenshot of a computer

Description automatically generated

*Example from my SonarQube dashboard*

## GKE Security Posture

Security Posture is a set of practices google employes on their Kubernetes clusters.

Features:

* Hardened and regularly patched node software.
* Shielded GKE nodes for boot integrity and rootkit prevention
* Encrypted traffic
* Vulnerability detection and analysis for each pod
* Threat detection
* Misconfiguration detections
* Supply Chain security
* Network security
* Strict trust between components

A screenshot of a computer

Description automatically generated

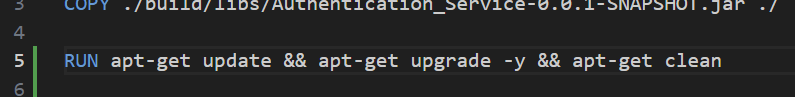
They provide this dashboard where I can see all details about their security and what have their scanners picked up.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

In this case I discovered that the docker images I was using had packages with vulnerabilities. To fix this I switched images to a simpler one and added a step on the image build to update any packages used.



*Update packages step*

A screen shot of a computer

Description automatically generated

*New image*

On top of outdated packages, I also had some misconfiguration issues.

A screenshot of a computer

Description automatically generated

*Misconfiguration issue*

This issue happens because my Kubernetes deployment files were deploying the pods as root user with full permissions which is bad practice. To fix this issue I had to add the following configuration on the deployment files of all the services

A screen shot of a computer

Description automatically generated

*New configuration*

With these fixes the dashboard values got reduced.

A screenshot of a computer

Description automatically generated

*New Results*

The reason why the values are not at zero is because of two component that I have that are kafka and zookeeper(kafka dependency). The images for these have vulnerabilities that I didn’t try to fix because in a real-world scenario most likely kafka wouldn’t be in the cluster like it is now. Even if in a real-world scenario this was the case nowadays there are more recent kafka versions that don’t rely on zookeeper so if I had one of those all these problems would be fixed.

## Container Image scanning

I have a container repository on my google cloud environment. This repository has image scanning active, so every time an image is published here it gets scanned.

A screenshot of a computer

Description automatically generated

*Docker registry with scanning enabled*

A screenshot of a computer

Description automatically generated

*Example of image scan*

The above is an image scan of the new images that I started using after the Kubernetes scans warned me. As you can see there are 6 total vulnerabilities where 5 have no fix available while one has. The one vulnerability that is fixable is from tomcat(spring native web server), however it cannot be fixed because at the time of writing there isn’t a spring version that is already using this patch. In any case is a medium case vulnerability that doesn’t allow RCE (remote code execution), so the outcomes don’t affect any important component.

## Cloud monitoring

All resources in the cloud have cloud monitoring enable allowing me to check their metrics and logs for any kind of unwanted events. Particularly helpful are the network logs that display every single incoming request where I can analyze it and trace it.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

*Network specific security logs*

## Communication

All my services communicate using TLS 1.3 for encrypted traffic.

## Cloud Firewall

Since I use google’s private network I also have a very strict firewall that only allows outbound traffic and then there is the exception for the load balancer. Even within the same network there is only traffic allowed for specific components like a database and nothing more.

A screenshot of a computer

Description automatically generated

*Firewall Rules for the network*

On top of that I also have logging enabled for each firewall rule.

A screenshot of a computer

Description automatically generated

*Logs for Firewall rules*

## Cloud IDS

Cloud IDS is an intrusion detection system that detects malware, spyware, command-and-control attacks and other network-based threats.

This IDS works by setting up and endpoint. Once this endpoint is setup we create a mirror policy. This is what mirrors the traffic that the IDS will analysis and flag.

A screenshot of a computer

Description automatically generated

*IDS Endpoint*

In my case I have created a policy to mirror all traffic from the subnetwork where my system components rely on which is the “europe-west1”.

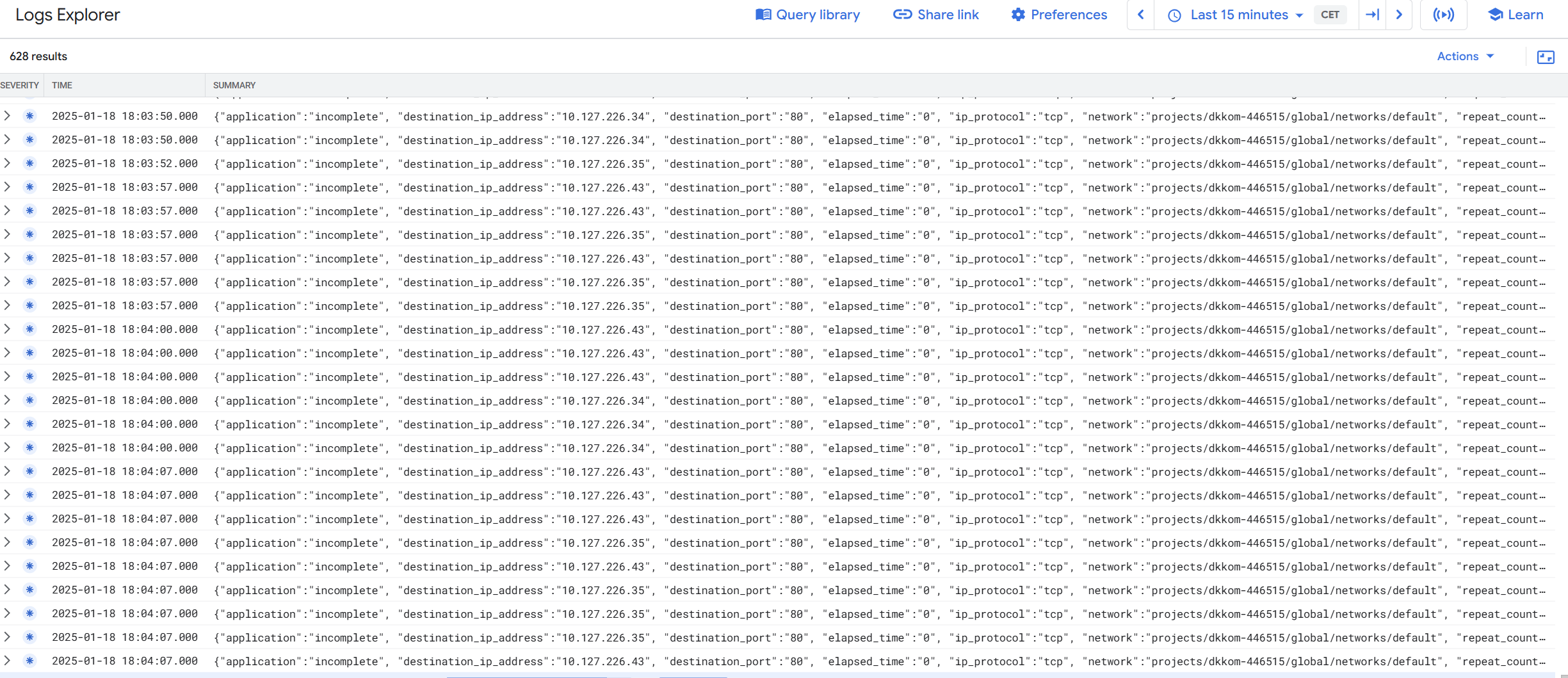
A screenshot of a computer

Description automatically generated

*Threat dashboard*

Any flagged threads are displayed in this dashboard where I can act.

Within the log explorer I also can see all logs from the IDS and the traffic it analyzed.



*IDS logs*

The google IDS is a very powerful specially because it’s built with industry standard technology. In addition, this setup would be perfect to have an HIDS like wazuh and then connect everything to a SIEM but unfortunately there wasn’t time for that.

## IAM Accounts

Some of my services need access to cloud resources at runtime, for this I have custom IAM accounts with only the necessary permissions following the principle of least privilege. This allows for reduces risks in the event that a service server gets hacked, and the credentials get exposed.

A screenshot of a computer

Description automatically generated

*Example of different accounts I have for different access.*

## Extras

On this sections I will mention some tools or practices that could also be added to my SDLC to provide the best possible security.

### White source

White Source is an open-source security management platform that helps organizations manage the use of open-source components within their applications. It provides visibility into security vulnerabilities, license compliance, and quality issues related to open-source libraries and dependencies.

Use cases:

* Vulnerability Management
* License Compliance
* Dependency Tracking

### Aqua

Aqua Security is a comprehensive container security platform designed to secure containerized applications, Kubernetes environments, and cloud-native infrastructure. It provides runtime protection, vulnerability scanning, and compliance monitoring for containers and microservices.

Use Cases:

* Container Security
* Kubernetes Security
* Vulnerability Scanning
* Compliance Monitoring

### Contrast

Contrast Security is a security platform that focuses on providing real-time, automated application security testing. It helps identify vulnerabilities in applications during runtime and provides deep insights into the security posture of the application without needing external testing. (IAST)

Use Cases:

* Runtime application security monitoring
* Code Scanning
* Security Monitoring

# Conclusion

My current security implementations cover all seven security layers more than enough with some security layers even repeating themselves to ensure extra fault tolerance. There are always improvements and best practices to be implemented. Since there are vulnerabilities found every single day 100% security is never guaranteed but with this setup even if something goes wrong it should be picked up quickly. This setup could be even further improved by creating special analysis metrics for the system. With his I mean seeing all data the system produces and filter out anything that is outside of that. This is a very hard job that requires an entire team but would be an extra that would for sure take this to a very complete solution. As an overall I think I did everything I could and made the system extremely secure.