Sjoerd Halmans

Productio

Productio Security report

Inhoud

[Preface 2](#_Toc72835964)

[SQL Injection 3](#_Toc72835965)

[Broken Authentication 4](#_Toc72835966)

[Sensitive Data Exposure 5](#_Toc72835967)

[XML External Entities (XXE) 6](#_Toc72835968)

[Broken Access Control 7](#_Toc72835969)

[Security Misconfiguration 8](#_Toc72835970)

[Cross-Site Scripting (XSS) 9](#_Toc72835971)

[Insecure deserialization 10](#_Toc72835972)

[Using components with known vulnerabilities 11](#_Toc72835973)

[Insufficient Logging & Monitoring 12](#_Toc72835974)

# Preface

In this document I will be documenting the security status of Productio, and explain what measures are taken at the moment to protect the application against attackers.

This document will mostly talk about how the application is protected against the top ten security risks as described by the OWASP top ten.

# SQL Injection

The OWASP top ten sees SQL injection has the biggest risk that plagues modern web apps, and for a good reason.

To protect against attacks of this kind, I have taken it to myself to use ORM systems on every service that is connected to a database, due to the nature of these ORM systems the chance of SQL attacks is severely diminished.

One thing that makes the application more secure, is that no queries are used in all of my API’s. All of the services use the premade functions offered by said ORM system, this means the only place where SQL injection could happen is in the create functions, these are cleaned to make sure only proper data comes through and no malicious SQL queries are put through to the database.

# Broken Authentication

Second on our list is Broken authentication, so let’s dive into it.

Not much was done on my part to protect the application against Broken authentication, this is because Auth0 does most of the work for me in this case. The application is protected against brute force attacks using the Auth0 API and multi-factor authentication is set in place so that employees need to accept login via their smartphone to login on a new device.

Apart from that, Productio also puts a high standard on which kind of passwords are allowed, this is also regulated by the Auth0 API and makes sure that all of the passwords used by users are up to standard.

# Sensitive Data Exposure

This third one is important, Productio is built on being a platform for a big company. This means there are many juicy company secrets in the databases.

To protect against this security risk a few measures were taken. To start off, if data does not need to be saved in Productio then it won’t, this removes any unnecessary data storage and decreases the chance of this data being accessed by a user with malicious intent.

Maybe the most important data that is being stored in the application is of course all of the users their passwords and other personal information like emails and phone numbers. These are protected in an Auth0 hosted database and encrypted using Bcrypt.

# XML External Entities (XXE)

To make sure this security risk is taken care off, care was taken in how data is serialized and send around the application and the different services. To start off only JSON is used to serialize data and all of the modern practices are used while doing this.

Furthermore, none of the services use any XML libraries, further decreasing the chance that Productio will be hit by an XXE attack.

# Broken Access Control

Broken access control can be very dangerous if for instance user ID’s are used in routes, this is never done in Productio. All pages load in data based on the user that is logged in, this is checked whenever a call is made, (for instance, if a HR employee somehow made it onto the production page, they would still not see any data due to the gateway refusing their request based on their security clearance.) this makes sure that all actions that a user made are only possible if their clearance level allows it.

# Security Misconfiguration

Perhaps one of the easiest but longest ones to fix, security misconfiguration can be a huge risk.

I have made sure that the configuration of my security system has had the time it needs to be operational and properly working, most of the configuration was done using the documentations of all the systems I used, this makes sure that no mistakes were made by me due to inexperience with any said system.

Furthermore, I have checked all of the configuration of my application multiple times to make sure I did not miss anything.

# Cross-Site Scripting (XSS)

Cross site scripting can be very dangerous if different users can upload data to a system, and have this data rendered by other users. In Productio this happens quite often, this means that XSS is a serious security risk and has been taken seriously as well.

To make sure that no XSS attack is made on Productio all of the data that users upload is checked and sanitized, though I will admit that this does not happen everywhere yet and maybe not on a satisfactory level.

This means that XSS is not possible on the most basic levels at the moment, but care should be taken in further development to increase the defence to also make sure that the more advanced XSS attacks are negated.

# Insecure deserialization

In short, insecure deserialization is handled in Productio by not accepting serialized objects from any unknown and untrusted sources.

# Using components with known vulnerabilities

In short, all components and libraries that I have used within Productio have been checked on any known security issues, and if they were there they have been handled with the recommended actions.

# Insufficient Logging & Monitoring

Logging within Productio is done at two different locations, Redis and Auth0.

Redis logs all of the requests that go through it, and Auth0 logs all of the login attempts made by any user of the application. I currently think that this is not enough logging to truly log all of the important information, this is why I am planning on making an additional logging service if time permits it later on in the project.

# Network

One of the main measures I have taken to protect my services from any incoming attacks is putting them on an internal network, thus disabling any attackers from directly reaching them. The way the network is set up as of now is that the gateway and the frontend are reachable over the internet, but the gateway only accepts requests coming from the frontend and needs a token generated by Auth0 for this specific application.

This is displayed in the diagram linked below :

[Network Diagram](https://lucid.app/lucidchart/invitations/accept/inv_d6ba7511-2635-49e9-a221-1be2cf9d9d33)