**Dumb Dumber Dumbest**



# Web Application Penetration Test

Product Name: Runtime\_Terror Web Application

Product Version: 1.0

Test Completion:

Penetration Testers:

Kealan McCormack 18312236

Lukasz Filanowski 18414616

Gerard Colman 18327576

## Project Information

Application Name: Runtime\_Terror Web Application

Application Version: 1.0

## Overview

Dumb Dumber Dumbest conducted a penetration test on Runtine\_Terrors Web Application. This report presents the findings of a comprehensive vulnerability assessment based on penetration tests of the web application. Each reported vulnerability contains a detailed description along with a proposed solution.

## Total Vulnerabilities

## Tools Used

OWASP ZAP Version 2.11.1

Burp Suite Community Edition v2022.1.1

## Source Code Alterations

Text

Description automatically generated

Changed the datasource URL to point to a docker container on port 4000

Changed the password to match with the mysql db running in the container

Changed server port to 8070 to allow burp suite to function

# **Findings**

## Vulnerabilities Due to Application Components

Outdated Libraries – CWE 829 - Medium

Bootstrap v3.3.7 - <https://snyk.io/test/npm/bootstrap/3.3.7>

Possibly don’t use any of these

JQuery v3.2.1 - <https://snyk.io/test/npm/jquery/3.2.1>

Possibly don’t use any of these

# OWASP Top 10 – 2021

1. **Broken Access Control**

LUKAS – Opinion on this on? Could go in design or here

**Deny by Default**

One of the main Clauses of Access control is to deny access to any resource that shouldn’t be publicly available. This concept is broken by allowing access to vaccine bookings from the homepage without being logged in.

**No rate Limiting on Apis**

Not sure where this should go

1. **Cryptographic Failures**

**HTTPS**

Data is protected in transit using HTTPS to ensure the data is encrypted. This prevents man in the middle attacks and is a key part of cryptographic functionality for a web app.

**Encoding (Passwords)**

Passwords are encoded using BCryptPasswordEncoder in order to store the password in a secure form.

1. **Injection**

**JPA stops SQL injection**

The use of Spring JPA repositories means that by default parametrised queries are being used. This effectively prevents SQL injection attacks.

**Thymeleaf stops XSS**

This web app uses Thymeleaf to display dynamic user generated content. By default thymeleaf (when set use text) will escape the text being printed. This effectively stops XSS attacks.

1. **Insecure Design**

**Allow admin access through registration form**

During the Account Creation stage, the user is asked if they wish to register as an Admin or Standard User via a check box. If the Admin check box is pressed, the account is promoted to have Admin privileges.

**Allowance of weak passwords**

When creating an account, the password field only enforces a password with 6 characters. When it should be 8 characters. The password doesn’t feature character requirements.

1. **Security Misconfiguration**

**Explain secure headers being used, etc**

**Use secure headers**

1. **Vulnerable and Outdated Components**

**Outdated libraries**

Outdated libraries were used in different aspects in this application, because of this there are security vulnerabilities known to attackers making the application vulnerabilities.

1. **Identification and Authentication Failures**

**Unlimited requests**

The application does not limit the number of requests being sent to it in a certain amount of time making it vulnerable to DDos attacks.

**Multi factor Authentication**

Currently the web app has no form of multi factor authentication, the only authentication is via password. This is a major vulnerability as it makes brute force attacks much more viable.

1. **Software and Data Integrity Failures**

**Use of Calls to external library (Vega)**

1. **Security Logging and Monitoring Failures**

**Lack of logging**

This web app has mismatched logging functionality. In several controllers, there is logging for any form of update or write to the database. For other controllers, there is no logging of any information. This mismatch leaves the web app open to potential exploitation without any warnings being sent or any logs generated to find the issue.

1. **Server Side Request Forgery**

Ask Liliana what we do

## App-Specific Exploits and Vulnerabilities

**Gain Admin Access with no checks – Critical (4 – Insecure Design)**

During the Account Creation stage, the user is asked if they wish to register as an Admin or Standard User via a check box. If the Admin check box is pressed, the account is promoted to have Admin privileges. This is a critical security and design issue as any user can create an account with Admin level access and thus access all Admin functionality. This vulnerability makes some of the below exploits redundant as Admin access is available with no effort.

**Location in Code**

**Solution**

The first step is to remove the ability for any user to create an Admin account. That sort of power should be given to a root Admin which should have an account on the web application by default. Make sure to use a strong password for this account. That Admin should have the ability to create other admin users through a screen/form they only have access to.

**No Strong Password Enforcement – High (4 – Insecure Design)**

When creating an account, the password field only enforces a password with 6 characters. The [recommended shortest length](https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/minimum-password-length#:~:text=Best%20practices,for%20users%20to%20easily%20remember.) for a password is 8. These characters can be any characters meaning that the password doesn’t have to be complex or contain strong password characteristics like symbols, capital letters or numbers.

**Location in Code**

**Solution**

The best solution is to enforce a strong password policy. Start by requiring a password of length greater than 8 at least. Moreover, strong passwords have at least 3 of the following traits: Uppercase letters, Lowercase Letters, Base 10 Digits, Non-alphanumeric Characters etc. For more information on strong password requirement look [here](https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/password-must-meet-complexity-requirements).

**No Limit on Requests – Medium (7 - Identification and Authentication Failures)**

The application doesn’t limit the number of requests that can be sent to the server. This means that the web application is susceptible to Brute Forcing Password or Denial of Service (DoS) attacks.

**Location in Code**

**Solution**

Set a limit on the number of requests that can be accepted at a time or in a short span of time to mitigate this issue.

**Brute Force Passwords – High (4 – Insecure Design)**

The Web Application is susceptible to Brute Forcing the password during the user login stage. This is possible as there is no limitation on the number of Requests that can be sent when attempting to Login. This can be done using a Cluster Bomb Attack for example. This process can take a long time, although when it is completed, it can be detrimental to the security of the web application as access to an admin or user account can be gained.

**Location in Code**

**Solution**

Set a maximum number of attempts that a user can attempt to log in. After this number is exceeded, suspend logging in for some time or prompt the user with a “Forgot password” page.

**Forum Slowdown Due to Excess of Posts – Low (4 – Insecure Design)**

Currently the web app doesn’t have any form of request limiting built in. This allows any registered user to send post requests to the forum without limit. This can result in the forum being flooded with posts, which causes a hit to the performance. Additionally depending on the database set up, a user could send enough requests to fill the database causing the disk full error in MySQL. This would prevent further posts to the forum.

**Location in Code**

**Solution**

Impose limits on the speed at which users can make requests to the forum.

**No Anti CSRF tokens – Medium (Server Side Request Forgery** **)**

The requests being sent in the application do not have an anti CSRF token. This leaves the application open to cross site request forgery attacks. These attacks leverage the victim’s browser to send requests from a compromised site to this site, which will then have the user’s cookies. If the user is logged in to the site, these malicious requests can use the victim’s identity to make requests to the application.

**Location in Code**

**Solution**

Most of this is already done, as the application generates a per session token.

However, as the session doesn’t expire, that could cause an issue.

Spring security also has a built-in protection against CSRF attacks in the form of a CSRF token.