APRIL 16 2020

# SAFETY REVIEW

## ROCKET ELEVATOR WEBSITE

MAXIME PARENT

**A vulnerability review of the Rocket Elevators information system implementation**

- **Describe and identify the location where the information system is likely to expose a vulnerability of the nature described in one of the 10 listed vulnerabilities.**
- **Explain the potential risk to your information system if the vulnerability is not corrected.**
- **Describe the modification needed to resolve the vulnerability**

# Sensitive Data Exposure

## SSL (Secure Sockets Layer) provide by Cloudflare is not active on the website.

**SSL** is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browser remain private.

**Attacker** can monitors network traffic and steals user's session cookie and then use the same cookie and steals user's session and access sensitive data. It compromises all the data that should have been protected. Usually, attackers steal credential, personal data, username and password.

An SSL solution to decrypt traffic and send to inspection devices is a good step to mitigating the effects of malware.

**The solution is simpl**e: encrypt all data in transit with secure protocols SSL with Cloudflare.

## Username, password and keys for Zendesk API are not hidden.

**Attacker** can access our Zendesk API account and steal valuable data like forms, customers informations and employee informations.

**The solution** is to hide the key, password, username in the file application.yml which is link in gitignore and configure in the interventions controller the informations by example : ENV["ZENDESK_URL"]

# Cross Site Request Forgery

**CSRF is an attack that tricks the victim into submitting a malicious request. It inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf.**

Our web application primarily uses GET requests to transfer parameters and execute actions and GET requests are not protected since they do not have side effects like writing to the database and don't leak sensitive information.

**Cross Site Request Forgery** attacks target functionality that causes a state change on the server, such as changing the victim's email address or password, or purchasing something.

**An attacker** may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

**To protect ourselves**, we should turn on request forgery protection in every controllers of our web application.

As an example:

```
class ApplicationController < ActionController::Base

 protect_from_forgery

end

class LeadsController < ApplicationController

 protect_from_forgery except: :index

end
```

## Broken Authentication:

**Attackers have access to hundreds of millions of valid usernames and password combinations for credential stuffing and sometimes to the employee or user email address.**

Our Employees/admins don't have strong passwords on the login page of our website and the permit default password when creating a new user is too weak and an attacker can get thru too easily.

**Attackers** can take advantage of this weakness and steal informations, change data, broke the entire website etc.

We definitely should change the password of the employees of Rocket Elevator by stronger password and change the policies password length and complexity when a new user sign up for the first time.

## Using Components with known vulnerabilities:

Rocket Elevator use a lot of APIs and components on the web application and we often didn't take the time to look at the versions that we implement in our code. Obviously it's a bad habit and we should take the time to look if the components are supported, vulnerable or out of date but we should also scan for vulnerabilities regularly and subscribe to security bulletins related to the components we use. If we do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion it's a vulnerability for our web application.

While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components. Depending on the assets you are protecting, perhaps this risk should be at the top of the list.

The solutions are to be up to date with all APIs we use in our web app, use scanners such as retire.js help in detection and check if the composents are still supported or out of date.

### Our login does not block bots to enter our website as a user.

reCAPTCHA is a provider of human verification systems owned by Google.

The main purpose of a CAPTCHA system is to block spambots while allowing human users. reCAPTCHA is a free service can protects our website from spam and abuse. reCAPTCHA uses an advanced risk analysis engine and adaptive challenges to keep automated software from engaging in abusive activities on your site. It does this while letting your valid users pass through with ease.

The solution is to install and setup reCAPTCHA in our web app.