# Software Requirements Specification:

for

# Vaultron

Version $< 0.0.1 >$

Prepared by

**Cryptomaniacs**

| | | |
|---|---|---|
| Colton King | 11245746 | colton.king@wsu.edu |
| Grant Wade | 11435949 | grant.wade@wsu.edu |
| Robby Boney | 11453444 | robby.boney@wsu.edu |
| Rob Wooner | 11496643 | robert.wooner@wsu.edu |

**Date:** Sunday, October 15th, 2017

# Contents

# 1 Introduction

The goal of this project is to create a cryptographically secure cross platform password manager. The cross platform compatibility will be achieved using electron and node.js. This section will describe who the intended audience for the password manager will be and describe the purpose of the project in depth.

## 1.1 Document Purpose

The product we are writing this SRS document for is the cryptographically secure cross platform password manager Version 0.0.1 (*Vaultron*). This password manager will create strong passwords and encrypt them. It will remember the password for the website it is being created for.

## 1.2 Project Scope

This software is a password manager that creates cryptographically secure passwords. It will store the hashed passwords in a JSON file for safe keeping. There can be multiple profiles, each one will have a master password of its own that will unlock *The Vault* gaining access to the passwords. The master password will be created by the user so that they can remember it. The password manager can however create a good master password that the user can write down to remember. the master password will not be sent through email or text so that there will be no chance of it being stolen from a malicious attacker.
Using this password manager allows the user to have strong and secure passwords that they will not have to remember. Not needing to remember allows for a strong password that has a very little chance of being cracked. The user only needs to sign in with their master password and copy and paste the desired password from the vault into the website, or other password field.

## 1.3 Intended Audience and Document Overview

Client: The intended audience of this software is anyone that uses the internet and currently keeps track of their passwords by hand or lets the browser keep track of them. This software will offer a safe, secure, and isolated way to store any passwords. Security is our number one priority, if something can be secured it will, so we can ensure that no user data can be leaked.
This document will be used as a starting point to overview the software. It will describe how it can be used, the security behind the password storage, and what technologies will be used to create the software. The rest of the document contains all of the specifications for the software, which includes product functionality, operating environment, interface design, functional requirements, behavioral requirements, security requirements and other product details.

## 1.4 Definitions, Acronyms and Abbreviations

**CSS:** Cascading Style Sheets, used to style our electron application

**Electron:** Cross platform desktop application development environment using HTML, CSS, and JavaScript
**HTML:** HyperText Markup Language, used to design our electron application
**JavaScript:** Programming language used on the web and Electron
**Vault:** Password storage platform created for this project

## 1.5 Document Conventions

IEEE formatting requirements were generally followed when writing this document. Notable conventions include:

- All companies, products and programs are mentioned in itallics (i.e. *electron*)

## 1.6 References and Acknowledgments

[1] RSA Laboratories. *PBKDF2*. URL: `https://www.emc.com/collateral/white-papers/h11302-pkcs5v2-1-password-based-cryptography-standard-wp.pdf`.

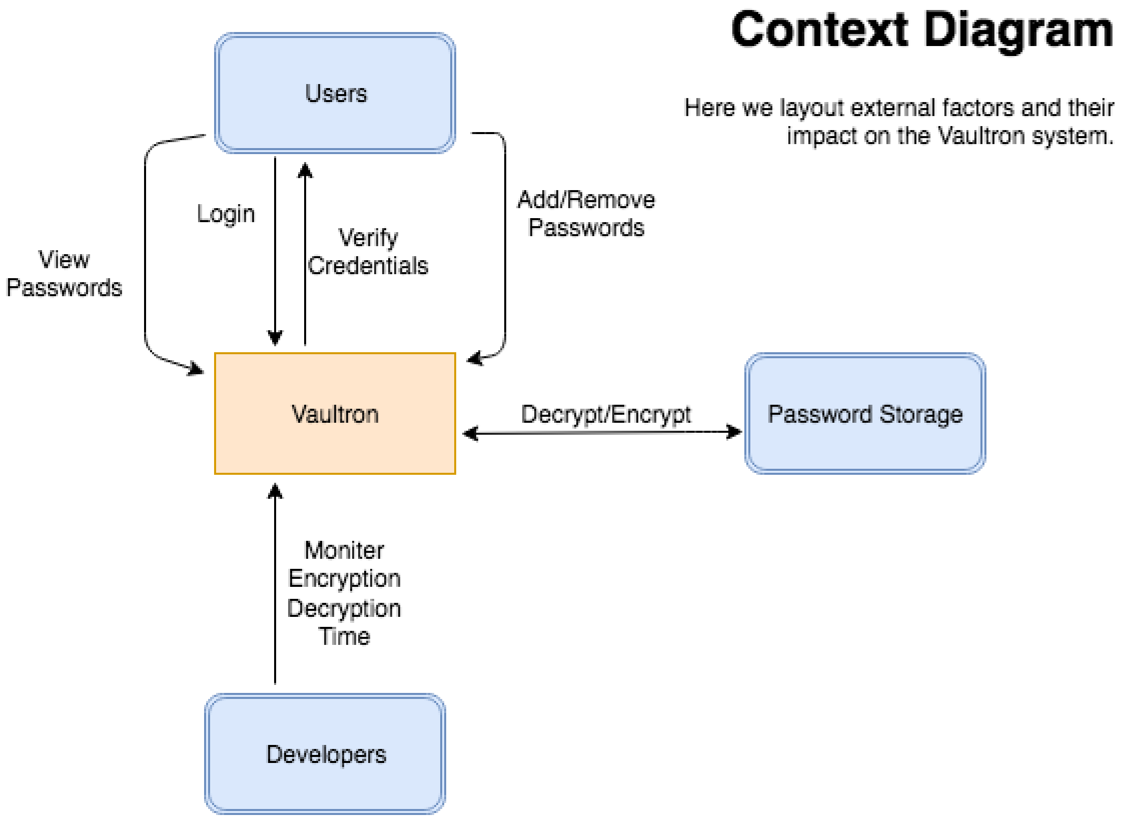
[2] JGraph Ltd. *Draw.io Diagram Generating Software*. URL: `https://about.draw.io/`.

[3] W3 schools. *W3 schools CSS styling*. URL: `https://www.w3schools.com/`.

# 2 Overall Description

## 2.1 Product Perspective

At the time of design this project is intended to fill a similar niche in the password storage and serving field that *1Password* and *LastPass* fill. Thus it is in the same family as the two products previously mentioned, a system that stores passwords securely and allows later retrieval. This product will be implemented in two main parts. One, a desktop application that will serve as the *Vault* storing the passwords, creating the passwords, and allowing later access to the passwords. The second main part is the browser extensions that will be used to auto-fill in usernames and passwords into any website that the user adds to their vault.



## 2.2 Product Functionality

- Generate secure passwords (any length or complexity)
- Authenticate user to decrypt passwords (required before use)
- Generate a *Vault* on first run of software
- Search through passwords using either the website name or username

- Allow creation of new entries within the *Vault* (passwords, notes, etc)
- Utility to sync users *Vaults* between computers (Dropbox, Google Drive, etc)

## 2.3 Users and Characteristics

The users for our software will only need a basic knowledge of how to use a computer to use this product. Our users will be a vast majority of people that want to only think about one password rather than keeping track of a different password for every account they have. All users will be important and every user should be confident in the security of their passwords and they should also be confident that there will be enough space to have all the passwords that they need to store.
Different Types of users:

- Basic User
  - The user that will only have a few passwords saved in the vault
- Power Users
  - Users that have over 1000 passwords in the vault

## 2.4 Operating Environment

*Vaultron* works on a wide variety of systems with minimal requirement mainly due to *Electron's* requirements. The base system specs are as follow:

**Mac**

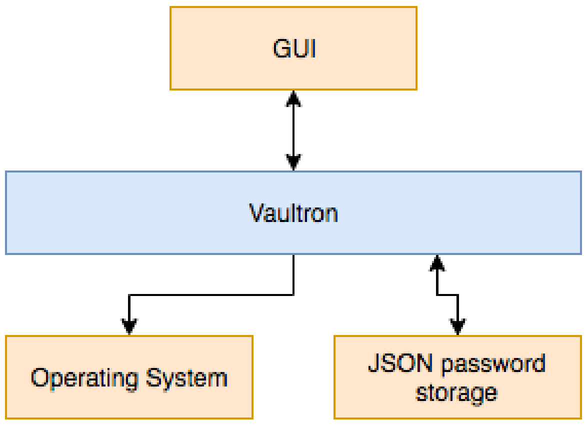
- OS: 64bit
- Minimum Version: macOS 10.9

**Windows**

- OS: 32bit (x86) or 64bit(amd64)
- Minimum Version: Windows 7 and later

**Linux**

- OS: 32bit (i686) or 64bit(amd64)
- Tested Version: Ubuntu 12.04
- Valid Versions: Ubuntu 12.04 and later, Fedora 21, Debian 8



## 2.5 Design and Implementation Constraints

The biggest constraints for this software are time and security considerations. We need to make sure that our software follows popular, effective, and accepted security protocols. Given the time frame in which to create this product and the importance that our password manager successfully encrypts and protects our passwords, we need to work hard and efficient.

## 2.6 User Documentation

We will provide a simple help page that will go over how to use and run our software. The help page will go over how to create and add a new entry in the Vault. How to create a new password. How to delete an old entry. How to update an entry.

## 2.7 Assumptions and Dependencies

Assumptions for *Vaultron* are split into two categories, assumptions being made about the user and assumptions that are required about components that are used to create it.

### 2.7.1 Assumptions about the User

The assumptions that we are making for the design of this software for the users is that the users will be using a secure password, and they will not leave their computers unsecured. On the password side, assumptions have to be made since we can only put so many constraints on the contents of the master password. Forcing complexity can result in the user forgetting their master password causing them to loose all access to data stored in *The Vault*. The user leaving their computer unlocked with *Vaultron* open could result in someone with access to their computer copying all the passwords they have stored in *The Vault*. Overall we have to assume the users will do their best to secure their passwords.

### 2.7.2 Assumptions about the Software

On the software side we are making the assumption that any components that are going to be used will not change quickly and without notice. *Vaultron* as a whole requires *Node.js* and *Electron* to function, if either were to change it would require a rewrite of the core components of this product.

# 3 Specific Requirements

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

Our product will have a login window that will have a username text box and password text box and an enter button. The enter button will be pressed after the password and username are inputted. The login window will blur out the rest of the window behind it. We will have a minimize, maximize and exit button in the top left corner. we will have tabs along the top that will take you to different password profiles, like work, play, etc. There will be a create password button that will create a pop up that has entries for a url and a drop down for picking the password profile.

### 3.1.2 Hardware Interfaces

*Vaultron* has no specific hardware interface requirements, everything is run within the operating system of the computer.

### 3.1.3 Software Interfaces

The main interface between *Vaultron* and any other software is it's usage of *Node.js* and *Electron*. They deal with system calls for many things including generating pseudo-random secure bits for salt's and master keys. Files will also be stored within the filesystem of the computer to allow storage of passwords and other data. They allow us to develop for any operating systems they support without having to write system specific code decreasing our time working on compatibility.

### 3.1.4 Communications Interfaces

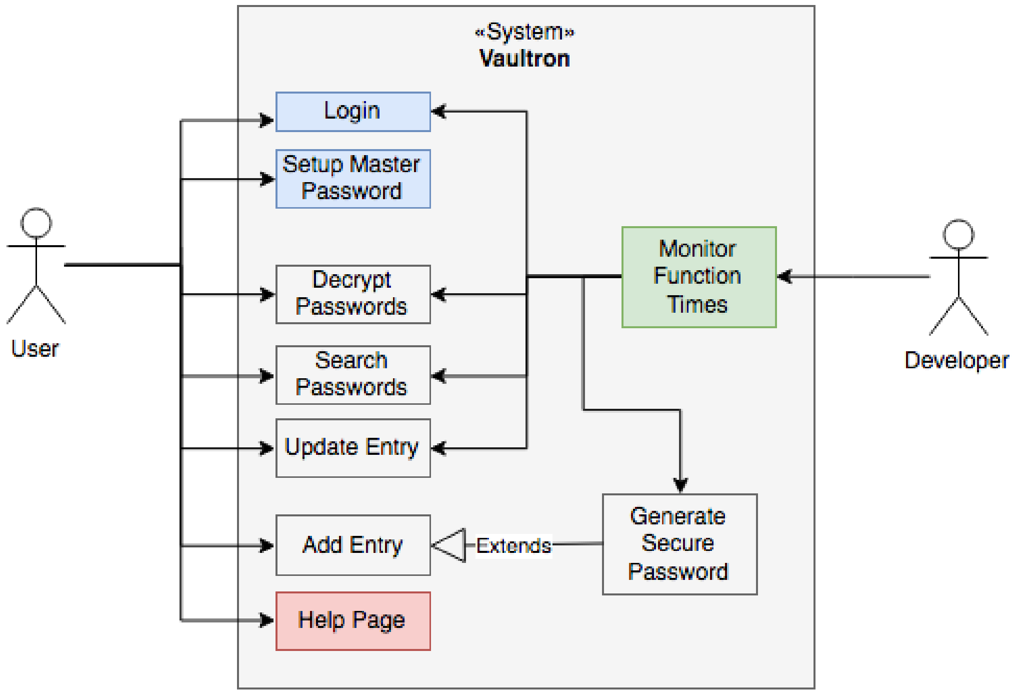
Since *Vaultron* is a password manager encryption has to be used to store the passwords securely to prevent passwords from being revealed. The master password that the user creates to secure their *Vault* will be hashed using PBKDF2 with a pre-generated salt that is unique to their profile. All stored passwords will be encrypted using the SHA-512 algorithm. Usernames for stored services will also be encrypted.
Other communication interfaces that might be used include, some system to facilitate the transfer of a users *Vault's* to another computer. The current options for that include *Google Drive, Dropbox, Nextcloud* and *Syncthing*. One of these would be used to sync all of a users passwords to another computer to keep passwords up to date everywhere.

## 3.2 Functional Requirements

## 3.3 Behaviour Requirements

In our Use Case Diagram the actors involved are the user and the developer. The developer will monitor the different function times for all the functions that the user will interact with such as logging in, decrypting passwords, searching passwords, updating entries, adding entries, and generating secure passwords.

### 3.3.1 Use Case View



# 4 Other Non-Functional Requirements

## 4.1 Performance Requirements

The following requirements are based on a system with minimum specs:

- Log-in time should not take more than 2 seconds (verifying user)
- New entries into the Vault should be less than 1 second before software can be used again
- Updating entries should take less than 1 second
- Password generation should take less than 1 second
- GUI freezes should not last for more than 1 second

## 4.2 Safety and Security Requirements

Safety and security are the main goals of *Vaultron*, if it is unable to safely store passwords there is no reason to use it. Therefore we must abide by some rules. First, any keys created must be generated from a cryptographically secure random number generator. Our implementation uses the *Node.js* `randomBytes()` function to generate secure random numbers. Secondly, no decrypted passwords will ever be on any storage device. If a decrypted password was ever stored on a drive there is a chance that it is recoverable, so passwords will only be decrypted on the fly when they are needed and promptly removed from memory when they are not. Third, all passwords will be stored in their hashed form within a JSON file. If this file were to be stolen there would be no direct path to take to decrypt the password hashes because our implementation will not expose the master decryption key ever. Fourth, *Vaultron* will follow the best practices currently in the industry to safely store users passwords as to minimize possible threats to the users security. This includes using pbkdf2 hashing to verify the users master password, and stored passwords will be encrypted using SHA-512 using a master key that is descrypted from the master password. Salting will also be used to enhance the security of each password stored.

- PBKDF2 password hashing for authenticaiton of master password
- SHA-512 + salting of stored passwords
- Decrypted passwords will never be stored on the computers drive
- Safeguards to prevent tampering with software
- Secure transfer of *The Vault* between computers
- There are no specific certifications that must be followed
- The issue of privacy will be dealt with by storing the vault in a secure place in the users home directory

## 4.3 Software Quality Attributes

This software can be broken down into a couple quality attributes, reliability, security, cross platform compatibility, and portability.

### 4.3.1 Reliability

*Vaultron* should be able to accomplish all of the features mentions in this document without crashing or otherwise corrupting *The Vault*. If the vault were ever to be corrupted all passwords could be lost of otherwise never recovered. This is something that during the developement we are striving to ensure. Backups will be taken of the vault periodically to potentailly save the user's passwords should anything happen to their current *Vault*.

### 4.3.2 Security

Since the security implications of this software are vast *Vaultron* must continue to be updated to follow the lastest security standards. This will be accomplished by storing what version of our encryption is used on each password so decryption will always be possible even once a new standard is released. Users will also be given the opportunity to update all of their passwords to this new standard following release.

### 4.3.3 Cross Platform Compatibility

Cross platform compatibility is almost entirely covered by the use of *Electron*. It is able to create binary files for Windows, MacOS, and many Linux distrobutions. If *Electrion*, one day, does not cover all of the needs that *Vaultron* requires other options will be considered.

### 4.3.4 Portability

Portability is being considered the ability to transfer users passwords between their computers so everything is availiable all the time. The current thoughts on how this can be accomplished is the use of services like *Dropbox*, *Google Drive*, or maybe an open source solution like *Nextcloud*, or *Syncthing*.

## 5 Other Requirements

# A Data Dictionary

| | |
|---|---|
| Vaultron | our security software |
| Vault | the place where the user's passwords and entries are stored |
| Electron | javascript wrapper for application development |
| Node.js | javascript package for backend related tasks(i.e reading, writing, ect) |
| sqlite | SQL database being used to store passwords and keys |
| Basic User | Vaultron user with minimal password requirements |
| Power User | Vaultron user with extreme password requirements |
| PBKDF2 | key derivation functions with a sliding computational cost used for encryption. |

## B Group Log