Introduction

Team Name: PasswordTeam

Team Members: Victor Jodar, Jay Paul Luben

Application Title: SecureMyPasswords

Description:

SecureMyPasswords is a simple, easy to use, and secure password management web application in which users can store all of their online accounts' passwords in one place and under one master password. Users can store existing passwords or use SecureMyPasswords to generate a password of a specified length which can fulfill any number of password requirements (i.e. must have capital letters, numbers, special characters, etc.).

Function Requirement Specification

- User account creation (sensitive)
- User login (sensitive)
- View passwords list
- Add password
- Modify password
- Remove a password
- Copy password to clipboard

Type of Program: Web Application

Development Tools:

- Javascript
- HTML
- CSS
- React
- Meteor
- MongoDB
- IntelliJ / Visual Studio Code
- Github

Requirements

1. **Security and Privacy Requirements**
    - The application must ensure that the users' login credentials are kept secure and private.
    - To keep track of security flaws that may arise throughout the development process, our team plans to note flaws and work to solve them using the projects tab on Github. Github provides a simple and organized way to list flaws and work on them using the Automated Kanban template. Our team will be able to color-code issues indicating severity of the flaw, allowing us to mark them as solved when moving the issue to the 'Done' section. While moving forward, it is essential that we don't delete any flaws present on the board in case any issues arise from them.
2. **Quality Gates (or Bug Bars)**

    Privacy

    Critical

    - Lack of notice and consent (the application transfers sensitive user data from the user without notice or consent)
    - Lack of data protection (no authentication mechanisms to protect users' sensitive information such as their passwords)

    Important

    - Lack of notice and consent (the application transfers sensitive user data from the user without notice or consent)

    Moderate

    - Lack of internal data management and control (data stored does not have a retention policy)
    - Improper use of cookies (users aren't logged out after a certain period of time)

    Low

- Lack of notice and consent (hidden metadata of information)

Security

Critical

- Elevation of privilege (execution of arbitrary code or obtain more privilege than authorized)

Important

- Information disclosure (attackers can locate and read system information not intended to be exposed)
- Spoofing (attackers displaying a “fake” login login prompt to gather login credentials)

Moderate

- Denial of service (prevents interaction with the web application)

Low

- Tampering (temporary modification of any data that does not persist after restarting the app)

3. **Risk Assessment Plan for Security and Privacy**
   - **a.** How
     i. During the creation of the web application, regular testing should be conducted to ensure that no critical flaws are left unsolved.
        - Testcafe is an excellent tool to automate testing.

     ii. Parts that require threat modeling
        1. Login feature
        2. Client interaction with the database

Design

1. **Design Requirements**

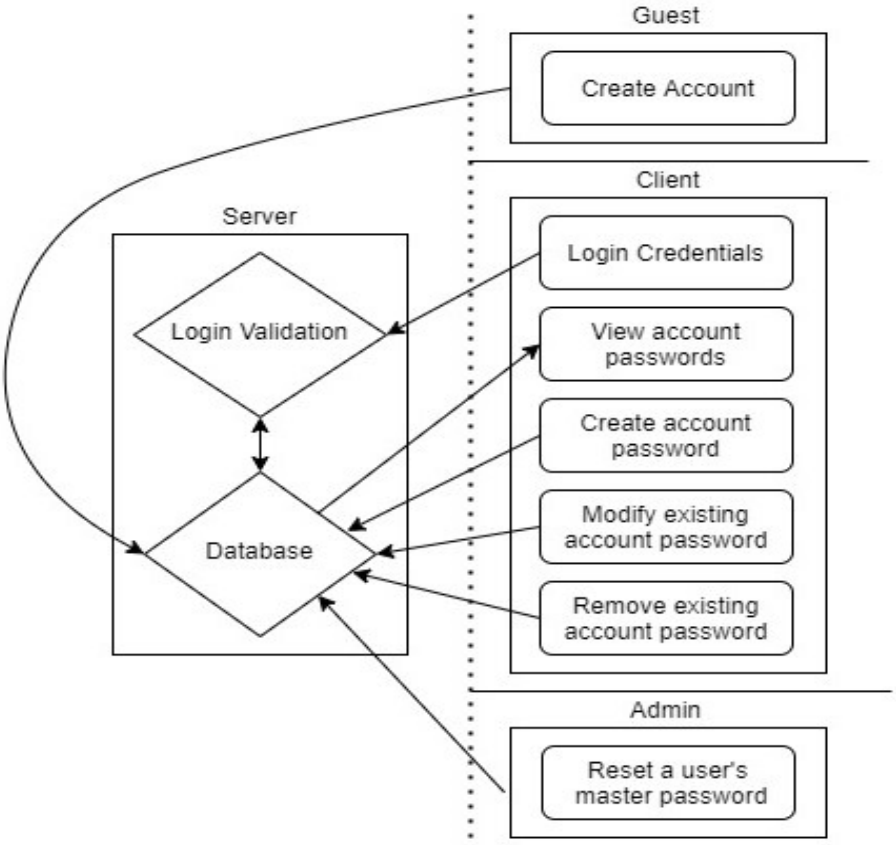
a. As our application primarily deals with passwords, the most important overall requirement is to keep these passwords, especially the master password, as secure as possible. Privacy must be maintained between the user and server as well as between other users. The master password should never be visible to anyone other than the user themselves when specified and should always be encrypted when travelling between client and database. Even after a user's master password is entered, the list of account passwords on their homepage should not be visible and should default to ambiguous black dots representing each character of the password to prevent threats of more physical nature.
b. As for the functionality requirements, our application's primary goal is to provide a secure way of consolidating our users passwords in one place to make their online work simple and safe. Therefore, the application should be simple and intuitive so that users don't need to spend any more time than they need, limiting the opportunity for malicious attacks. To the same end, the application should also implement an automatic time out feature that automatically logs users out after a few minutes of inactivity.

2. Attack Surface Analysis and Reduction

a. Using submittable forms, users are able to:
    - Create master account
    - Read and Write to their list of online accounts and associated passwords.
    - Generate passwords
b. Using forms, admins are able to
    - Perform the same actions as users
    - Reset a users' master password given their account name and/or associated email address
c. Attack surface analysis
    - In addition to the vulnerabilities and areas targettable to malicious attacks associated with the tools we will be using that are listed in section A.5, our application can also be targeted through unsafe code practices such as displaying submitted form information in the URL, or simply bugs in our code that causes errors like stack overflow.

d. Vulnerabilities found in other password managers
    - Phishing attacks done by disguising an illegitimate autofill option
    - Master password brute force attacks due to no limit on master password attempts
    - Ignoring subdomains and HTTPS autofill exploits
    - Users using weak or predictable master passwords

1. Threat Modeling:



Implementation

1. Approved Tools

With our app using an ICS template called meteor-application-template-react, all of the dependencies and their versions are listed via the package.json found here. Below are the more noteworthy tools that we will be primarily using, either taken from the list or tools we personally use.

Tool/Framework	Version
Node.js	16.2.0
HTML	5
Semantic UI	2.4.1
Meteor	2.1
React	15.0
MongoDB	3.4.1
ESLint	7.20.0
TestCafe	1.11.0
IntelliJ IDEA	2021.1.2
Microsoft Visual Studio Code	1.57
Github Desktop	2.8.3
Iroh.js	0.30

2. Deprecated/Unsafe Functions

- **HTTP**
  - "center" tag is deprecated, use "text-align" instead
  - "big", "font", and "strike" are deprecated, use CSS styles instead
- **Semantic UI**

    - findDOMNode is deprecated. Instead, a red should be added directly to the element to be referenced.
- **Node.js**
    - The constants module is deprecated. When requiring access to constants relevant to specific Node.js builtin modules, developers should instead refer to the constants property exposed by the relevant module. For instance, require('fs').constants and require('os').constants.
    - The fs.read() legacy String interface is deprecated. Use the Buffer API instead
    - util.print() has been removed. Please use console.log() instead.
- **React**
    - Placing a url in javascript code (i.e. ...) is a common attack surface because it gains full control over the page. Instead, attach event listeners and preventDefault
- **Meteor**
    - The deprecated Meteor.http object has been removed. If your application is still using Meteor.http, you should now use HTTP instead
    - DDP's connection.onReconnect = func feature has been deprecated, the DDP.onReconnect(callback) method should be used to register callbacks to call when a connection reconnects
- **MongoDB**
    - db.collection.findAndModify() is deprecated,
        - alternatives: db.collection.findOneAndUpdate(), db.collection.findOneAndReplace(), db.collection.findOneAndDelete()
    - db.collection.remove() is deprecated,
        - alternatives: db.collection.deleteOne(), db.collection.deleteMany()
    - db.collection.update() is deprecated,
        - alternatives: db.collection.updateOne(), db.collection.updateMany(), db.collection.replaceOne()
- **ESLint**

        - Deprecated global have been removed from the node, browser, and jest environments. Use of these gobals will trigger an error.
            - You can add a “globals” section to your configuration to re-enable any globals needed, however this is not advised as deprecated globals are no longer maintained.
            -

3. **Static Analysis**
    - The static analysis tool we chose to use was ESLint, a primarily Javascript code linting tool that finds and flags programming errors, stylistic errors, bugs, and suspicious code constructs. We’ve chosen ESLint as we are both familiar with it from our time in ICS 314 and ICS 414 and because it is built into our other tools, namely IntelliJ IDEA and Github.
    - Using ESLint is very simple and straightforward; it points out coding errors and will even provide explanations on what errors are being made by the programmer, giving us another line of defense even before compiling the code. It also points out inconsistencies or less than ideal programming practices such as using single quotes instead of double quotes and not having the correct amount of whitespace throughout the code. This ensures our code stays consistent and follows the same coding standards so that it is easier to review and less prone to errors.

Implementation & Verification

1. **Dynamic Analysis**
    - Of the few dynamic analysis tools that are made to be compatible with Javascript, our team decided to use Iroh.js. Iroh.js is advertised to allow users to record code flow in real time, intercept runtime information, and manipulate program behavior in real time. More information can be found on its website here: https://github.com/maierfelix/Iroh
    - Iroh.js was quick and straightforward to install, and after some testing, we were able to use Iroh to monitor variable values with the use of listeners, visualize function behavior in real time by viewing what functions are called with

parameters as well as conditional statements and the boolean values they return, and viewing performance statistics such as how long a function took to complete. These provided some powerful tools for debugging our program during runtime.

- Iroh.js did present some issues for us as well. Firstly, in order to utilize listeners and other tools Iroh.js offered, we must first manually code them alongside our program, which takes quite some time and adds many extra lines of code. Secondly, and more importantly, Iroh.js seems to be designed specifically for pure Javascript. Our program is written primarily using the javascript extension React and its component model, with minimal functions written in pure javascript. As a result, we are unfortunately unable to use dynamic analysis on a large portion of our project. That being said, we have only just started learning how to use Iroh.js, and it still offers some powerful and highly customizable runtime debugging tools that could make our code easier to debug and more secure.

2. **Attack Surface Review**
    - Microsoft Visual Studio Code has been updated for May 2021, to version 1.57. This update includes a number of improvements to the editor as well as some fixes to performance and compatibility on iOS devices, but nothing that particularly seems to directly affect our project. The update also brings a security enchantment in the ability to trust the different authors of the code.
    - No other updates, patches, or changes have been made for our other approved tools so far as there has only been a week's time since our previous report. Any new tool updates will be posted here in the future as necessary.

Verification

1. **Fuzz Testing**
    - The first test conducted was on the registration page by creating a new account with an existing username. As per expected behavior, this was unsuccessful and I was given a "username already exists" error message.
    - The next test was trying to brute force an accounts' master password. This too was unsuccessful because our meteor template application had some built in protection from this type of behavior. After a few unsuccessful login attempts, the

user is given an error message letting them know that they must wait 10 seconds before attempting another login.

- The next test was to try to manually access pages a user shouldn't be able to by typing in the URL of a known page such as the admin page while only being logged in as a regular user or not logged in at all. This too was unsuccessful due to the previously mentioned protection from our meteor template application. Upon manually entering a URL of a page without the proper user privileges, the application automatically redirected them to the sign in page instead. However if the user was already logged in with an admin account and tried manually entering the URL of an admin page, they would be successful.
- Another test conducted not really of security but rather of application functionality was trying to break a page by entering ridiculously long strings on forms. This was done on the Add Account form and the Password Generator form. On the Add Account form this was successful and the user is able to create an account/password that is so long that it breaks through its container and isn't fully viewable at least on my monitor. The Password Generator fared better as the form limits generated passwords to 25 characters.
- Something that didn't really need testing but definitely was a security concern was our application's ability to remember login status. If a user logged in, their login wouldn't expire without clearing their browser's cache and a malicious user could take advantage of this to access their account if the user forgot to sign out. This should be fixed by implementing an automatic sign out feature.
- Another security issue that didn't need testing was that the passwords on the account/passwords viewing page are plainly visible and not encrypted or hidden in any way. This will require a simple fix to hide it from plain view but is very important.

2. Static Analysis Review Update

- Our team has continued to use ESLint as our static analysis tool of choice to manage our code, keep it up to standards amongst team members, and help prevent a few errors. ESLint remains powerful and easy to use and it consistently brings syntax errors to the team's attention, who then proceed to fix immediately.

3. **Dynamic Analysis Review Update**
    - Our team has continued to use Iroh.js as our dynamic analysis tool of choice to help view information of our application at runtime. Unfortunately, our usage of this tool remains rather lacking due to the previously mentioned design of Iroh.js being specific to pure Javascript whereas our application is almost exclusively written in the Javascript extension React. The only notable exception however being the functionality of the Password Generator tool which Iroh.js has been used to view performance statistics when generating passwords with varying lengths and features (special characters, capital letters, etc.)

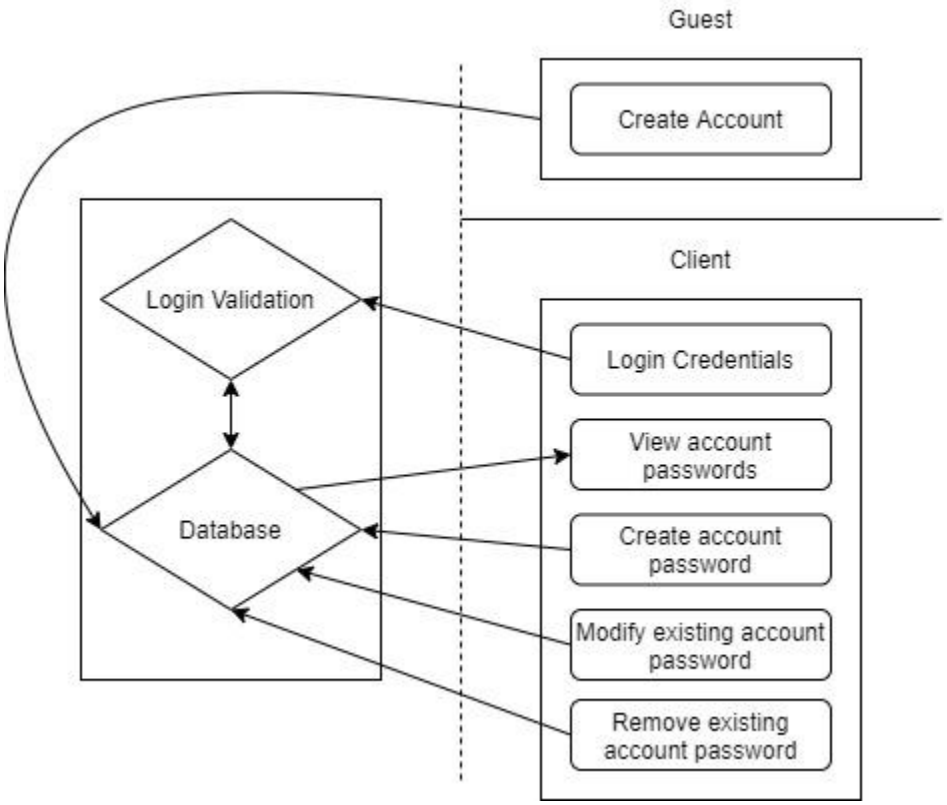
Release

1. **Incident Response Plan**
    - In the event of bugs or unintended application behavior, especially those causing potential data leaks or privacy concerns and issues, an incident response plan will be enacted to resolve said issues. Both members of the team will do their part in resolution within their roles as follows: Victor Jodar as Public Relations and Legal representative and Jay Paul Luben as Escalation Manager and Security Engineer. The team can be contacted with regards to security issues or concerns at PasswordTeamSecurity@gmail.com
    - Jay Paul Luben will be responsible for both Escalation Management and Security Engineering, providing guidance and supervision to the team during a high priority security issue while also creating a plan of action to proceed with issue resolution efficiently to minimize damage. He will also take charge in isolating the source of the issue and ensure the team resolves it while also ensuring future security measurements are then implemented to prevent the issue or a similar one from occuring in the future.
    - Victor Jodar will be responsible for both Public Relations and Legal Representation, providing consistent status updating on the current standing of the process of issue resolution to the public to provide assurance while also managing communication between the team's legal department and security department in

addition to negotiating legal actions between affected parties to find a suitable settlement.

- In the event of a security incident discovered by the team, security specialists, user reports, or another party, the team will follow these standard guidelines when resolving the issue:
    1. Determining the severity of the security issue
    2. Gather information on the issue through communication with discovering party, team members, or through research
    3. Isolate and inspect the source of the issue if possible
    4. Establish a timeline for the incident response
    5. Perform public relations and legal representation if/when necessary
    6. Execute the incident response plan
    7. Release application fix/patch to address future security issues similar in nature when applicable.

2. Final Security Review



- Our Threat Model has been slightly modified with the removal of the admin page and functionality. This was done because having full access to all users in the database by way of an administrator account meant that they also had access to all users' stored accounts and passwords due to the way MongoDB's subscription based data model works. We decided that this was too great of a security concern, prompting its removal. Our current Threat Model is used to review the data flow throughout execution of the application. SecureMyPasswords uses Meteor and MongoDB's subscription based data model to ensure regular users are only ever able to access their own data stored on the database.
- The static analysis tool that was used throughout the development of our application was ESLint which kept coding standards consistent between team members and acted as an extra layer of defense in terms of finding syntax errors and suspicious code constructs before compiling. The dynamic analysis tool the team used was Iroh.js which provided significantly less assistance due to its incompatibility with Javascript extension React, but nonetheless was able to provide some performance statistics for pure javascript functions, namely the Password Generator, during runtime.
- After testing SecureMyPassword thoroughly through the perspective of a new user, our team has come to the conclusion that its grade is Passed FSR. Previous issues discovered through Fuzz Testing have since been resolved, application source code passed ESLint testing, and our Threat Model in combination with the Meteor framework has been shown to handle data flow from database to each user securely.

3. Certified Release & Archive Report

- The release for SecureMyPasswords v0.1 can be found here: https://github.com/PasswordTeam/SecureMyPasswords
- Summary of Features:
    - Secure login system
    - Simple and secure password account management with option to add new entries or modify existing entries

    - Easy to use password generator with options for character length, upper case inclusion, numbers inclusion, and special characters inclusion
- Future Development Plans
    - General UI improvements
    - Integrate password generator tool into Add Accounts or Edit Accounts page for added ease of use
    - Implement autofill functionality have SecureMyPasswords automatically fill in account information to stored websites when user visits them
- Technical information of the application such as specifications for usage, installation, and download page can all be found on the GitHub repository: https://github.com/PasswordTeam/SecureMyPasswords