Team Budget Tracker

Irene Ma, Brody Uehara, Joshua Hartmann, Rashmi Arvety Ramanatha

Introduction

Our program is the Best Budget Tracker 1.0. The Best Budget Tracker is a budget tracker that keeps records of users' expenses and income to calculate a user's disposable income for the month. Users will have the option of signing up or logging in depending on their account history with our application. Once logged in, users will be able to store sensitive financial information that only they will have access to. Only authorized administrators of the platform will be able to see usernames and email addresses of users. The budget tracker takes in a list of income streams followed by a list of all expenses for the month. The tracker then returns how much disposable income a user can spend or invest for that given month. Our budget tracker will be a web-based application that anyone will have access to over the internet. Our budget tracker application will be written primarily in JavaScript and use IntelliJ Idea as our IDE. We will utilize MongoDB for the database and React/Meteor/Semantic UI for designing the user interface.

Security and Privacy Requirements

For security requirements, it is important to follow three principles when building our application; confidentiality, integrity, and availability. When logging into an account, there should be some kind of account authentication for the user to complete to determine whether or not the user is who they really are. Users must provide some sort of credential that matches the system's database (i.e. a password). For more secure authentication, a 2-step verification will add an extra layer of security to user's accounts. This will require the user to sign up via a password and another device such as their phone. Requiring users to create a secure password increases the security of their accounts. Having a password with a minimum of ten characters that contain at least one uppercase letter, a numeric character, and a special character is a common requirement in many authentication processes. Aside from authentication, authorization is another requirement that should be met. User authorization assures that the user has been granted access to the right account with the right privileges. The web application should require different levels of accounts that can be made with roles. This includes the user account which can access basic information on their end and an admin account which has access to all of the available users on the database and special privileges to manage the users accounts. It is also required for the application to be able to validate any user input to make sure that it is unharmful to the system. This includes permitting only certain characters in inputs to prevent any potential cyber attacks such as SQL injections.

For privacy requirements, it's important to assure users that their information is kept confidential and not sold to any third parties. A privacy requirement that the application should meet is having all account information secured and prevent any unauthorized user viewing their information. The application should only collect and use information that is relevant to its purpose, along with receiving the consent of users to store these information on our databases. There should be no collecting of any personal data from users without their knowledge or consent and also no selling of any user's data to third party businesses.

We will use github private repository and create a project board and each of us will create different branches for different tasks and connect these branches with the issues on the project board.

We will have a master branch which will have all completed tasks and it will be used for deployment/release. Each team member will create his own branch and write code for specific functionality. We will create one more intermediate branch for testing, before we merge the code into the master branch. The developer will initially merge the completed task from his/her branch to the testing branch.

All teammates will review the testing branch for security flaws/threats after completing the development of each usecase. If the developer accidentally pushes some keys or confidential information, sensitive information will be present in the testing branch but not in the master branch and it can be found in the review session and the flaws can be created as bugs in the issue board and the developer could resolve those bugs and re-test again. Also GitHub provides notification for security threats if we don't find it in the review session.

We are planning to use the open source security testing tool ZAP(ZED Attack Proxy) to find security related bugs and threats (Note: Based on the availability of time to learn a new tool and use it to test our application).

Quality Gates

Due to our program's online nature and storage of personally identifiable information, it is imperative to establish security and privacy levels to protect our users. In terms of privacy, it is critical that we establish data protection protocols to ensure our user's personally identifiable information is securely collected and stored in a trustworthy database that is username and password protected. In a budget tracker, only important financial information, like various income streams and expenses, should be recorded. It is critical to minimize the amount of data stored on our platform to increase user privacy. Therefore, no unnecessary information will be stored in our database. It is also critical to give notice and consent in our user interface. Our platform will give prominent notice and explicit opt-in consent in the user interface to transfer sensitive personally identifiable information. Our application will have two levels of access: a user level and an administrator level. The administrator will oversee operations on our application and will be able to access certain information on a user like their username and email address. Due to our application's multi-level approach, we must ensure a level of separation between users and administrators. Within our application, it is critical to provide data management and controls so that user's personally identifiable information is securely stored and restricted to only those who have proper authorization and valid business needs.

In terms of security, it is imperative that we are aware of potential vulnerabilities that could arise with our application. Due to our application working on the web, it is vulnerable to denial of service attacks. Moreover, due to the level of separation between regular users and administrators, our application is vulnerable to an elevation of privilege bug. This happens when an unauthorized user obtains more privilege than intended. As a result, user's personally identifiable information may be at risk for unauthorized access. Another vulnerability our application may face is information disclosure. Due to our application's collection of personally identifiable information, there is the possibility where an attacker can locate and read information that was not intended or designed to be exposed. To prevent this, each level of information access must be clearly defined and securely protected.

Risk Assessment Plan for Security and Privacy

To assess the security and privacy we will review each feature that we add to the application and update the security and privacy impact they can bring. There will be a privacy disclosure available to users so they may read on how we handle their privacy. For privacy concerns, as of now our application does not store information on the users computer, does not continuously monitor the user, does not install software or gather files on the users computer, does not transfer any user data to a third party service besides our database and does not store personally identifiable information unless the user decides they want to include it. If a privacy or security breach were to happen then while we work to fix it, we are prepared to take the application offline until a fix is made. We will look at the security and privacy requirements we laid out and test them with unit tests/penetration tests to make sure that we are following our defined requirements. Throughout development we can make analysis of each security and privacy concern to see how well we are doing with preventing leaks. Parts of the program that deal with user information such as their login and data as well as how we store and transfer data will require threat modeling and security reviews. Other parts of the program including the frameworks we use in order to maintain a secure application will definitely require security reviews. All parts of the code dealing with data such as forms, login, and api requests will be needed to be thoroughly reviewed in order to catch any possible security issues.

Design Requirements

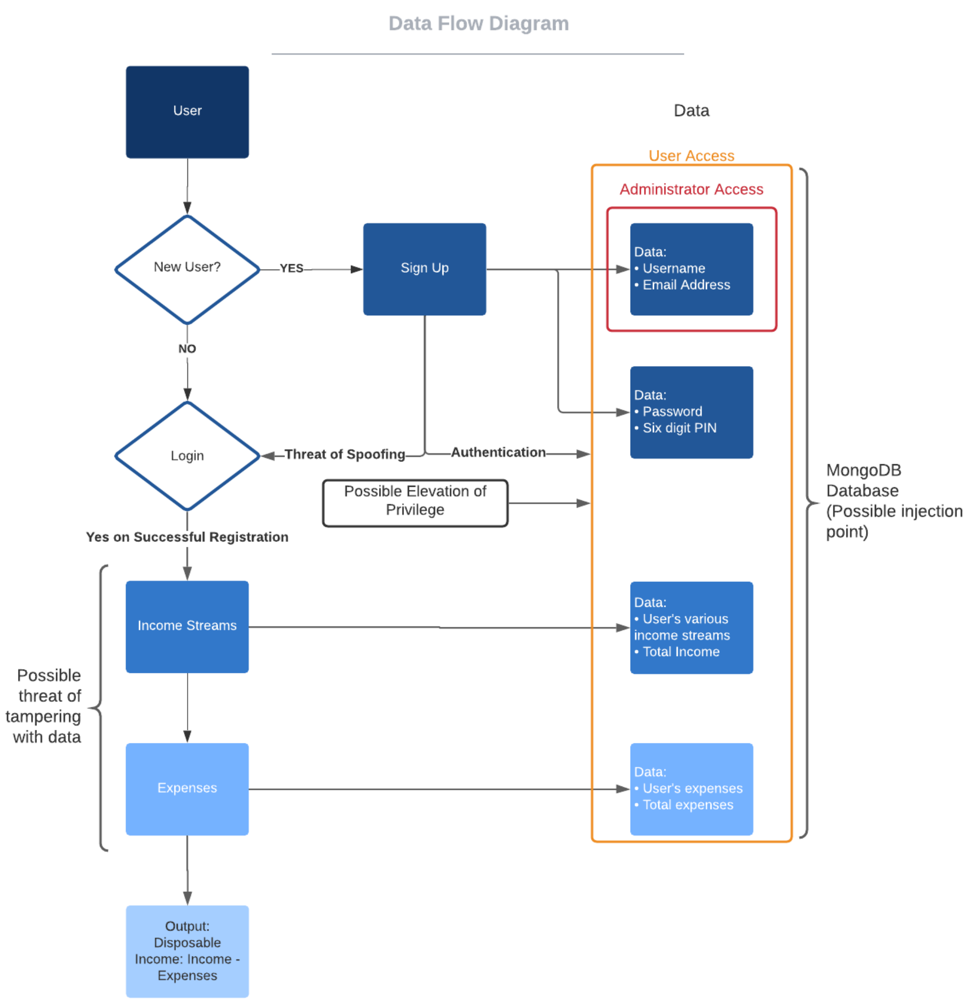
Design features upon first viewing the web application should require a homepage that explains the function of the website and the purpose of it, there should be a login page as well as a create your account button for new users. Upon creating an account, the site should display a message displaying the minimum requirements the password should meet. If the user chooses a username that is already registered to the site, an error message should display informing the user that the name has been taken and to choose another name. If implemented, there should be an option for the user to set up a 2-step verification method for logging in for a more secure account. If the user happens to forget a password or suspects any suspicious activity on their account, they are able to reset the password via email. Upon logging in, the site will show features that the user is able to perform such as inputting their financial information and allowing them to categorize them. The user should be allowed to add, edit, or remove any information. The site will display different tabs for income and expenses for easy readability and accessibility for the users. For admin users, they should have the ability to manage all the users accounts on the site. That includes viewing their usernames, emails and the ability to remove them from the system. The site's UI interface should be user friendly and provide enough information for the user to understand what feature does. They should be able to navigate the site easily without needing any help.

Attack Surface Analysis and Reduction

The privilege level that most users will have, those who will be using the web app, will be a basic level. They only have access to records that they own. With these records they gain the create, read, write, delete, append, and append to permissions related to only their records. Every basic user is separate from each other and may not access any data that is not their own. Higher privileged users such as database or system administrators will have more privileges. They should be able to access user records and are only given the delete privilege on the record. They should not be able to read any data. Since the application is still in the planning stage, the following list of vulnerabilities are those which can affect most web applications that can apply to this application: API's, databases, cookies, login/authentication entry points, admin interfaces, authorization methods, forms, url parameters, etc.. A malicious user may also exploit sections of code that deal with the api, forms, url parameters, admin login, and authentication/authorization to name a few. These sections could be exploited by a malicious user to gain access to data that they otherwise shouldn't have. Ensuring that the transfer of data to and from the database is done when authorized and only for the user that it belongs to as well as the data being encrypted will help reduce exploitation.

Other aspects such as admin control over the system should be kept to a minimum in case an admin account gets compromised, there won't be too great of a loss. Also ensuring that any third party components used don't contain any vulnerabilities and are used correctly will help to reduce attack.

Threat Model



The possible threats that could occur in our website are during authentication, storing information in MongoDB, bringing the website down and while fetching user inputs if not validated. Hackers can exploit the web application and get access to personal information like email, password, income and expenses. Hackers can inject the scripts and gain access to client side information. They can get access to cookies and session information and retrieve username and password. We are planning to use Meteor.js and react.js libraries for authentication and authorization. So we need to test thoroughly to avoid such security threats and look at the level of security provided by React and Meteor libraries. Hackers also might gain access to the information of other users like firstname, last name, email address, password, income and expenses. There might be a possibility to execute remote code to elevate user role to admin role. A bad person could potentially give false information and behave like a valid user and then gain access to the private data stored in mongo collections. An attacker might also get access to random accounts and might alter/destroy the data by falsifying (adding/deleting/modifying) income and expenses which the owner of the account does not expect. Moreover, an attacker might create a large amount of login requests which can't be handled by the server where our web application is deployed or other possible scenario could be Registering a large number of random users to our application through scripts and creating network overload to bring the server and services down. A hacker can easily create scripts and gain access to the entire server and in extreme cases might delete the mongo database that will be hosted on remote cloud servers.

Implementation

Approved Tools

Our application utilizes various tools for tracking budgets and is written primarily in JavaScript and HTML. Therefore, we utilize IntelliJ Idea as our integrated development environment (IDE) in conjunction with ESLint for code quality management. The budget tracker is also a web-based application. Therefore, we utilize JavaScript libraries such as React, Meteor, and Semantic UI for building our user interface. Finally, our program will store critical budget information utilizing MongoDB as its database. Listed below are the various tools and their corresponding versions our application utilizes:

- MongoDB version 4.4 (Database)
- IntelliJ Idea version 2020.2.2 (IDE)

- Web application security scanner version 1.6.49 (https://github.com/andresriancho/w3af)
- React version 17.0.1
- Meteor version 1.11.1
- SemanticUI version 2.4.1
- NodeJS version 14.15.5
- ESLint 7.20.0

Deprecated/Unsafe Functions

Along with each of our tools, we have compiled a list of depreciated functions and their safer alternatives below from the versions that we are using for our project. While this list does not represent all the depreciated functions from all our tools, it is a list of the ones we may use while developing our application. Moreover, not all tools are listed below because there were no depreciated functions found in the versions of our tools.

- MongoDB v4.4
    - geoHaystack index and geoSearch has been deprecated, use a 2d index with $geoNear or $geoWithin instead
    - Removed cloneCollection command, use mongoexport or mongoimport instead
    - Removed planCacheListPlans command, use the mongo shell helper method PlanCache.list( ) instead
    - Removed command planCachListQueryShapes command, use the mongo shell helper method PlanCach.list( ) instead
- Meteor v1.11.1
    - Assignments to Template.foo.rendered and so forth are deprecated but are still supported for backwards compatibility
- SemanticUI v2.4.1
    - Deprecation of gulp-util function, use replace-ext instead.
    - Several icon names have been deprecated:
        - Linkedin in is now linkedin alternate
        - Zoom in is now zoom-in

- Zoom out is now zoom-out
- Sign in is now sign-in
- Sign out is now sign-out
- Log out is now logout
- In cart is now in-cart

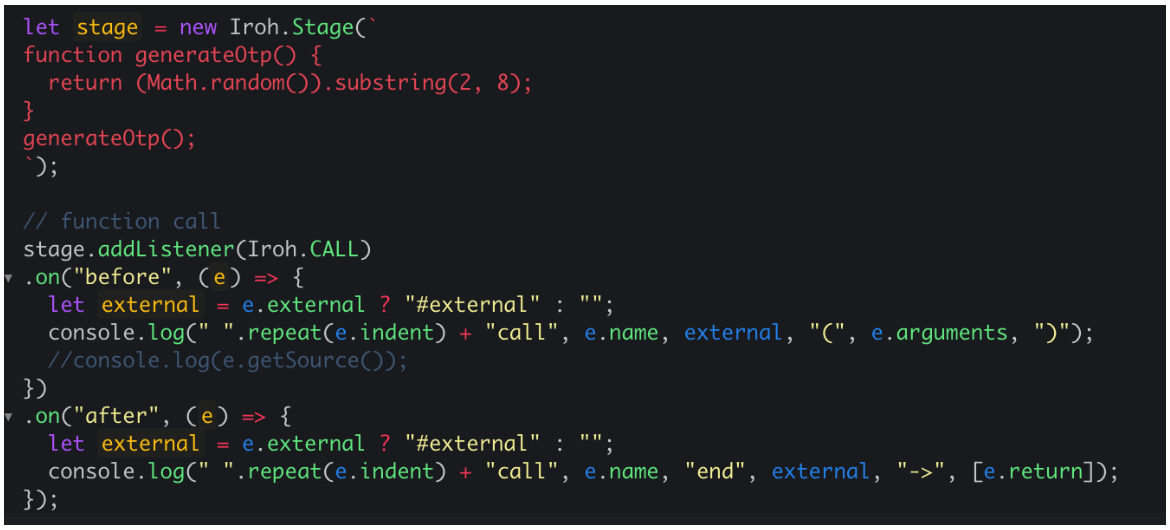
Static Analysis

In order to maintain proper code style during the development of our application, we will use ESLint version 6.0.0. ESLint is an open source static analysis tool we will use to identify and report on patterns/problems found in our JavaScript code. What makes ESLint unique is that all rules are completely pluggable. This means that we will be able to dynamically load rules at any point in time. For example, a pluggable checkstyle is Airbnb's style guide for JavaScript. In order to run ESLint, we will need to first install Node.js. After installing both ESLint and Node.js, we can configure default settings within our IntelliJ IDE to start using ESLint as a static analysis tool. During testing, ESLint was very simple to install via npm. Moreover, ESLint provided a very fast runtime environment because it was written in Node.js.

Dynamic Analysis

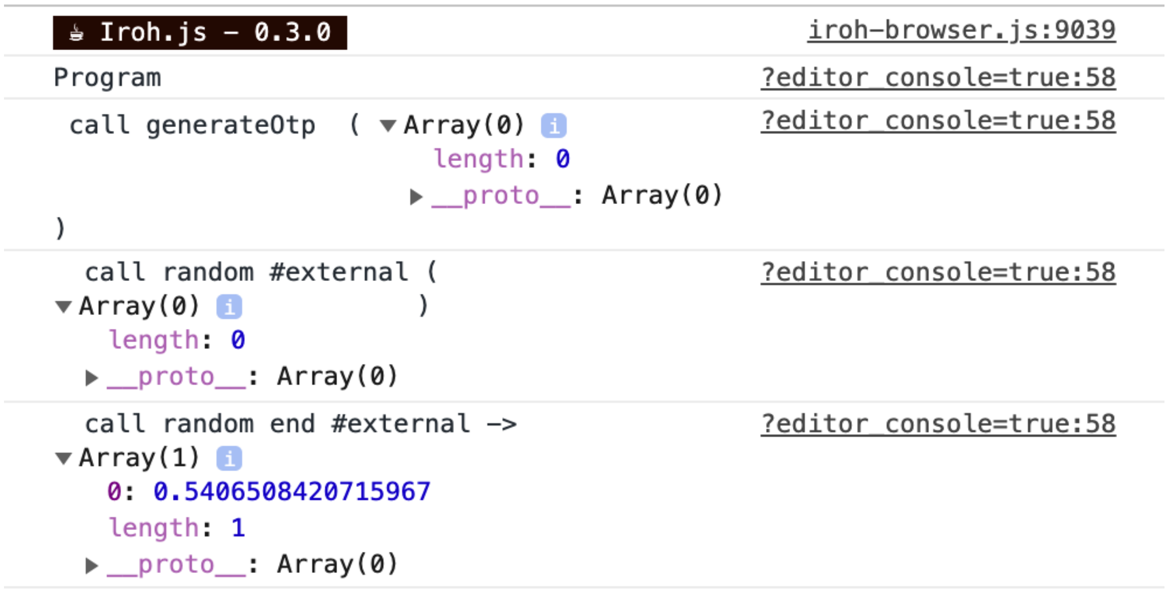
During the development process, our group implemented Iroh.js, a dynamic code analysis tool for JavaScript. Iroh allows developers to record code flow and intercept information and manipulate the program's behavior in real time. Iroh works by patching code to be able to record everything that is happening inside it without changing the original program. Iroh is a dynamic analysis tool that collects data which is only available at runtime. This is in contrast to a static analysis tool like ESLint which is performed on a non-runtime environment. We neither ran into any major problems using Iroh.js nor we have noted any positive success that helped us fix bugs or vulnerabilities in the code. We will continue to use Iroh.js in our application to verify our application is working well in all the scenarios. Iroh.js helped us to see the performance(in time) and see the execution call stack.

Example of Iroh.js used to trace the generateOtp function.



```
let stage = new Iroh.Stage(`
function generateOtp() {
  return (Math.random()).substring(2, 8);
}
generateOtp();
`);

// function call
stage.addListener(Iroh.CALL)
.on("before", (e) => {
  let external = e.external ? "#external" : "";
  console.log(" ".repeat(e.indent) + "call", e.name, external, "(", e.arguments, ")");
  //console.log(e.getSource());
})
.on("after", (e) => {
  let external = e.external ? "#external" : "";
  console.log(" ".repeat(e.indent) + "call", e.name, "end", external, "->", [e.return]);
});
```



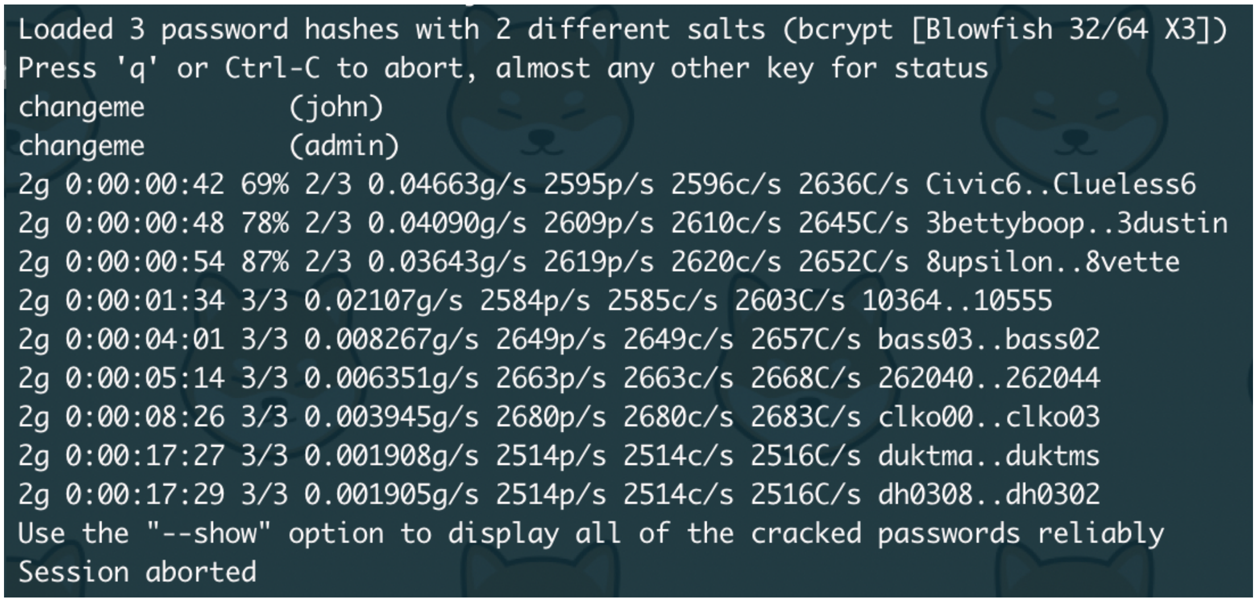
```
 ♨ Iroh.js - 0.3.0                                     iroh-browser.js:9039
 Program                                         ?editor_console=true:58
  call generateOtp  ( ▼Array(0) i                ?editor_console=true:58
                          length: 0
                        ▶ __proto__: Array(0)
 )
   call random #external (                       ?editor_console=true:58
 ▼Array(0) i            )
    length: 0
  ▶ __proto__: Array(0)
   call random end #external ->                  ?editor_console=true:58
 ▼Array(1) i
    0: 0.5406508420715967
    length: 1
  ▶ __proto__: Array(0)
```

Attack Surface Review

Looking back to our approved tools for our project, a new version for the IntelliJ Idea was released for 2020.3.3, however there were no reported vulnerabilities in this new version. Version 17.0.2 was released for React but the only update that came with the new version was removing an unused dependency to address a cross-origin isolation warning. There is a new version for Meteor with a 2.1 version that comes with the NodeJS security update. With NodeJS, there is a new version update to 14.16 that fixes some security vulnerabilities that came about with the previous version. All versions of the 15.x, 14.x, 12.x, and 10.x were vulnerable to denial of service attacks when too many connection attempts with an ‘unknownProtocol’ were established. This led to a leak of file descriptors. These versions were also vulnerable to a DNA rebinding attack when the whitelist includes “localhost6”. When the “localhost6” is not present in /etc/hosts, it is just an ordinary domain that is resolved via DNA over network. If the attacker controls the victim’s DNA server, the DNA rebinding protection can be bypassed by using the “localhost6” domain. The last vulnerability that was discovered was an integer overflow in CipherUpdate that may be exploited through Node.js. Updating NodeJS to the latest stable version will fix these security vulnerabilities. ESLint has a new update with 7.23.0, however there were no vulnerabilities reported.

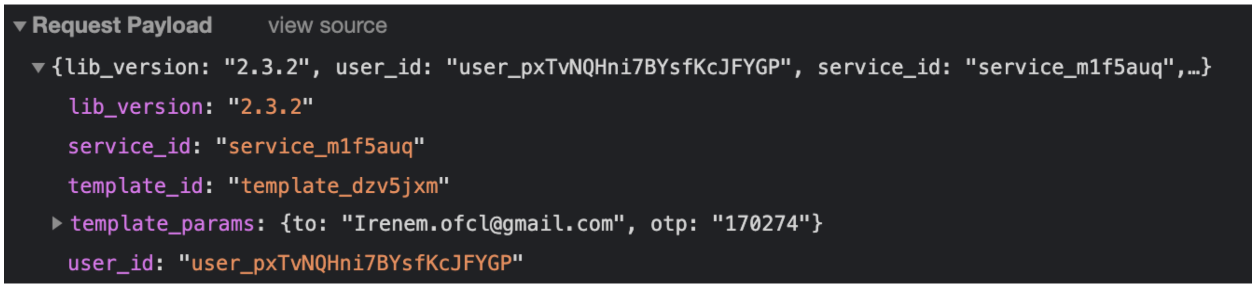
Fuzz Testing

We utilized both manual and automated attacks to try and break/hack into our application. An automated tool that we used was John The Ripper https://www.openwall.com/john. It is a pen testing tool for password cracking and provides a range of systems with many benefits. It identifies different password hashes, discovers password weaknesses within databases, and allows different modes of password cracking. In order to get John The Ripper to work, we needed to supply the tool with a copy of the password file from our local machine and edit it to include the user's username and hashed passwords. Meteor encrypts user’s passwords with bcrypt, which is the default password hash algorithm. Then, we specify what cracking method the tool should use. For our testing purposes, we chose the standard default cracking method, which goes through a series of common cracking modes; single crack mode, wordlist mode, and incremental. The single crack mode uses the login names and the users’ home directory names as candidate passwords, the wordlist mode compares the hashes to a known list of potential password matches, and incremental mode tries all possible character combinations as passwords.


```
Loaded 3 password hashes with 2 different salts (bcrypt [Blowfish 32/64 X3])
Press 'q' or Ctrl-C to abort, almost any other key for status
changeme         (john)
changeme         (admin)
2g 0:00:00:42 69% 2/3 0.04663g/s 2595p/s 2596c/s 2636C/s Civic6..Clueless6
2g 0:00:00:48 78% 2/3 0.04090g/s 2609p/s 2610c/s 2645C/s 3bettyboop..3dustin
2g 0:00:00:54 87% 2/3 0.03643g/s 2619p/s 2620c/s 2652C/s 8upsilon..8vette
2g 0:00:01:34 3/3 0.02107g/s 2584p/s 2585c/s 2603C/s 10364..10555
2g 0:00:04:01 3/3 0.008267g/s 2649p/s 2649c/s 2657C/s bass03..bass02
2g 0:00:05:14 3/3 0.006351g/s 2663p/s 2663c/s 2668C/s 262040..262044
2g 0:00:08:26 3/3 0.003945g/s 2680p/s 2680c/s 2683C/s clko00..clko03
2g 0:00:17:27 3/3 0.001908g/s 2514p/s 2514c/s 2516C/s duktma..duktms
2g 0:00:17:29 3/3 0.001905g/s 2514p/s 2514c/s 2516C/s dh0308..dh0302
Use the "--show" option to display all of the cracked passwords reliably
Session aborted
```

We tested three of our accounts against John The Ripper and within seconds, it was able to crack the first two passwords. The third password contained digits in it, which gave John The Ripper some work to do. As you can see in the screenshot, the various output messages show the process of cracking down the third password. Incremental mode is a powerful mode but would most likely not complete due to the infinite possibilities of combinations. From this test, we concluded that passwords with only lowercase letters were very easy to crack. To improve our security and fix this vulnerability, we require users to pick a password that meets certain criterias. Our passwords require at least a special character, an uppercase letter, lowercase letter, and a numeric character.

Another attack that was used was targeting the 2 factor authentication feature that we implemented in our project. Upon logging into your account, you are prompted to a page where you have to input a code that was sent to your email. This feature prevents an attacker from logging into account even if they were to obtain their password. The attack that was performed was inspecting the network traffic from the time of logging into the account to the application sending the OTP code to the email. By inspecting the request payload that was generated by the 2 factor authentication, you are able to view the OTP code without having to log into the account's email from the source code



```
▾ Request Payload      view source
  ▾ {lib_version: "2.3.2", user_id: "user_pxTvNQHni7BYsfKcJFYGP", service_id: "service_m1f5auq",…}
      lib_version: "2.3.2"
      service_id: "service_m1f5auq"
      template_id: "template_dzv5jxm"
    ▸ template_params: {to: "Irenem.ofcl@gmail.com", otp: "170274"}
      user_id: "user_pxTvNQHni7BYsfKcJFYGP"
```

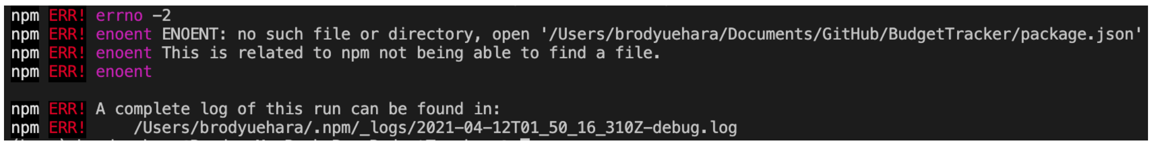
This is a security vulnerability as the purpose of the 2 factor authentication is to add an extra layer of security in addition to passwords. With this attack, the attacker will just need the password to the victim's account and they'll be able to view the OTP code without having to hack into the email itself. In order to fix this vulnerability, there is a way that we can prevent the public from viewing our network and source code from inspecting elements once the project is deployed. (The code is running in the development environment, which is accessing the application through localhost)

The last attack we tried was to manually attack the database. For the backend we store our data using MongoDB, which uses a NoSQL database. For this attack we are trying to make a NoSQL injection into the inputs within the application. This kind of attack aims to target input fields to pass in queries that the database will end up reading and acting upon. Such attacks could allow a malicious user to make the database do all sorts of things such as changing an account to an admin account, showing information they shouldn't be able to access, deleting the entire database, etc. We ended up trying various inputs to try and break into the database, however none of the attacks worked because our application was already checking if user input was clean, which is pretty much what it takes to protect against attacks. Another way is since we are using Meteor, a full stack javascript framework, which uses MongoDB for the database, attackers could be able to call the methods that are used to interact with the database from the browser console. In order to protect against this we make sure that our methods are not shown to the user. Luckily enough for us, they were already hidden as Meteor took care of that for us.

We tested our admin pages to see how safe our application was and how well it distinguishes between normal user role and admin role. Each page has different page URLs which is unique. We have created admin pages and they have specific URLs which can be accessed only by admin users. To test this we tried to manually access the admin page as a regular application user, by entering the admin page URL in our application browser, it redirects us to the login page. This is because our application checks for the admin role that is attached to the user's information in our MongoDB.
We made multiple test accounts and tested them manually multiple times and it worked 100% as expected and always redirected to the login page when trying to access admin page as a regular application user.

Static Analysis Review

Our project is completed and runs without any checkstyle errors according to ESLint. During our development process, we would use ESLint to maintain proper code style using the npm run lint command. The image below shows an example of an ESLint error after running the command.



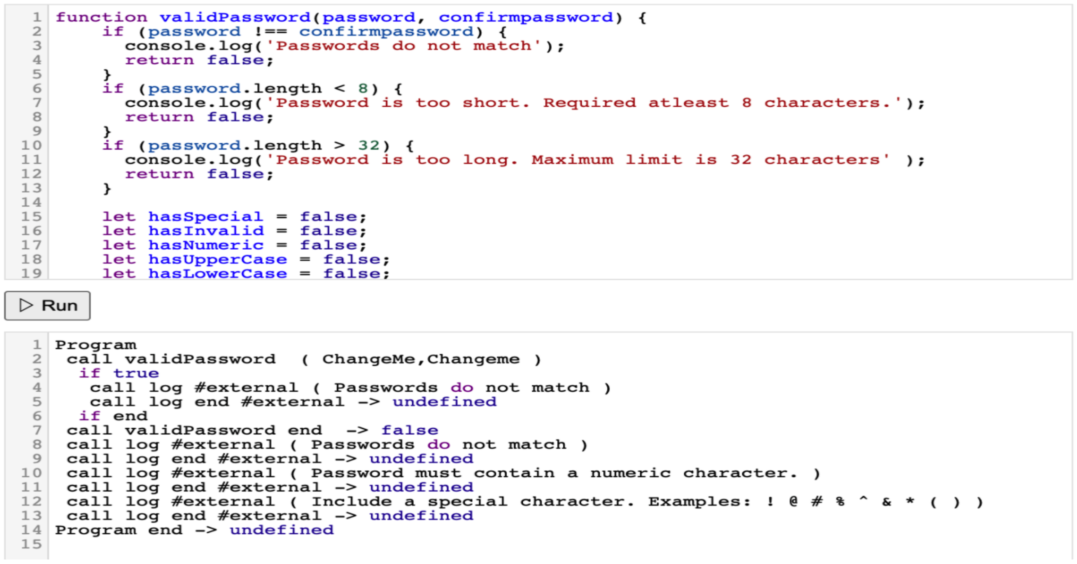
```
npm ERR! errno -2
npm ERR! enoent ENOENT: no such file or directory, open '/Users/brodyuehara/Documents/GitHub/BudgetTracker/package.json'
npm ERR! enoent This is related to npm not being able to find a file.
npm ERR! enoent

npm ERR! A complete log of this run can be found in:
npm ERR!     /Users/brodyuehara/.npm/_logs/2021-04-12T01_50_16_310Z-debug.log
```

Apart from running npm run lint, ESLint provides automated code style review. This ensured our code style was the same across all features and programs.

Dynamic Review

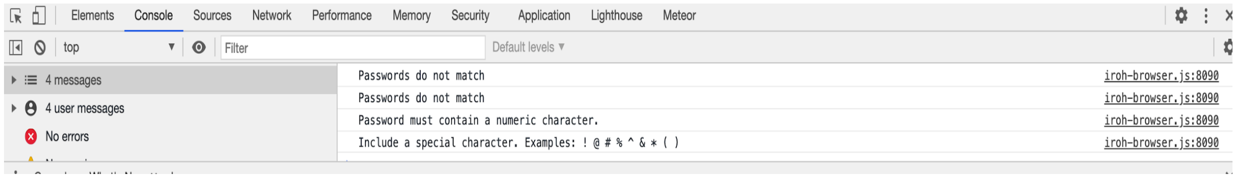
The dynamic analysis tool our group chose to use was Iroh.js. In the last assignment we had tested creating the OneTimePassword using Iroh.js. For this assignment we have tested a scenario to check password requirements scenarios which has multiple if else statements. Also each time we develop a new feature we have written multiple console.logs to make sure the execution of our code is correct and to view the results dynamically at runtime. Below is the code sample using Iroh.js that checks for password requirements.



```
function validPassword(password, confirmpassword) {
    if (password !== confirmpassword) {
      console.log('Passwords do not match');
      return false;
    }
    if (password.length < 8) {
      console.log('Password is too short. Required atleast 8 characters.');
      return false;
    }
    if (password.length > 32) {
      console.log('Password is too long. Maximum limit is 32 characters' );
      return false;
    }

    let hasSpecial = false;
    let hasInvalid = false;
    let hasNumeric = false;
    let hasUpperCase = false;
    let hasLowerCase = false;
```

```
Program
 call validPassword  ( ChangeMe,Changeme )
  if true
   call log #external ( Passwords do not match )
   call log end #external -> undefined
  if end
 call validPassword end  -> false
 call log #external ( Passwords do not match )
 call log end #external -> undefined
 call log #external ( Password must contain a numeric character. )
 call log end #external -> undefined
 call log #external ( Include a special character. Examples: ! @ # % ^ & * ( ) )
 call log end #external -> undefined
Program end -> undefined
```



```
Passwords do not match                                          iroh-browser.js:8090
Passwords do not match                                          iroh-browser.js:8090
Password must contain a numeric character.                      iroh-browser.js:8090
Include a special character. Examples: ! @ # % ^ & * ( )        iroh-browser.js:8090
```

RELEASE

Incident Response Plan

If there are security and/or privacy issues with the BudgetTracker application, the response will be handled by the Privacy Escalation Team.

Privacy Escalation Team:

Escalation Manager: Brody Uehara

- Help, coordinate and prioritize actions throughout all the phases of incident response (evidence collection, analysis, and containment phases).
- Analyze the severity of the event and take necessary actions based on nature and severity of the incident
- Responsible for coordination with external resources if any and ensuring for the completion of the process.

Legal Representative: Joshua Hartmann

- Legal representative for our application who provides legal advice during the incident response in case there are any criminal charges.

- Guides the organization on any potential future issues.

Public Relations Representative: Rashmi Arvety Ramanatha

- Representing the product(application) and team to the public, all communications with the public are handled and any questions from the media or public should be directed to the Public Relations Representative.
- Protects organization. Settings and communicates with outside entities when necessary.

Security Engineer: Irene Ma

- Provides expertise technical skills and assistance during incident identification, containment, and remediation.
- Updates security measures while maintaining incident response and also develops a security plan to avoid future issues.

Emergency Contact Email:
If there is an emergency, please contact Rashmi Arvety Ramanatha at rashmiar@hawaii.edu.

Incident Response Procedure:
The escalation process begins when an email notification of the incident is received.

1. *Identification and Prioritization*
   The escalation manager determines the severity and prioritizes the issue with the security engineer and the legal representative. Prioritization may change as we understand the incident in detail. Meeting discussion includes a rough outline of how severe the issue is, summary of the known facts, the validity of the incident, how long it may take to fix, users or features being affected and how it affects the overall application.
   Classifying incidents based on priority:
   Low: Incident is prioritized low when it affects very few users or a short-term breakdown or due to overloaded traffic and temporary issues.
   Medium: Incident is prioritized Medium when compromise of multiple accounts or an attack/compromise that is specifically targeted to our application or long term breakdown of any features.

High: Incident is prioritized High Severity when confidentiality or integrity is compromised or attempts to hack/hacked PII, SQL injection, password stealing or any major breakdown of application features.

2. *Containment and Intelligence Development*

For any incident, it is very important to collect and store the evidence and act quickly to mitigate the risk and take necessary steps to prevent the problem occurrence to other users

3. *Remediation*

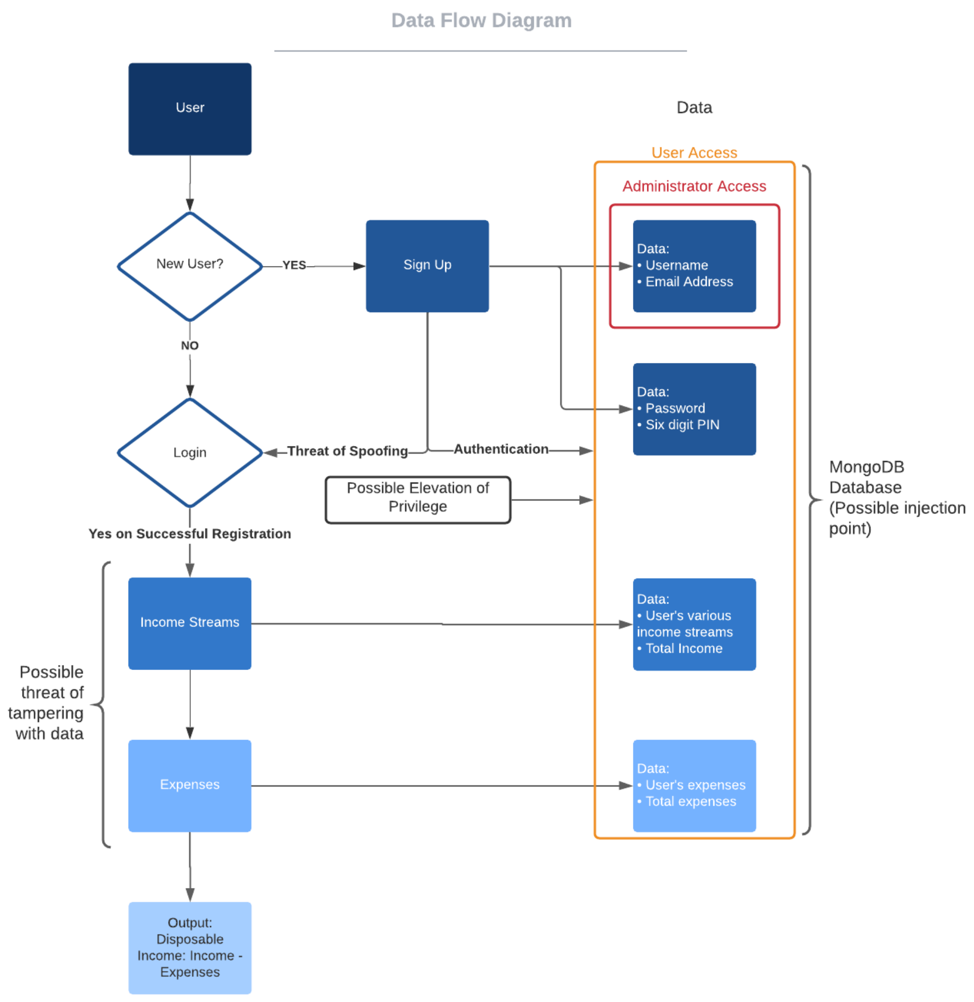
Escalation manager and the security engineer must ensure that the procedures have been followed to isolate the affected systems and preserve all forensic information. Once everything has been addressed, the security engineer will begin fixing the issue that has been reported. The legal representative will further determine the legal issues that may arise because of the incident and any legal implications that may already exist.

4. *Recovery*

The purpose of the recovery phase is to bring back affected resources/systems to production and test, monitor, and validate that the systems work well and are resistant to similar attacks. When the issue is resolved Public Relations Representative will work and release a statement to the public notifying the issue is resolved. During this phase, the incident response plan/procedures must be reevaluated and updated if needed to prevent similar future issues. Maybe write an internal incident plan for developers if it’s a technical incident, give training to employees and write online help articles.

Final Security Review

The design of this application when we first made our threat model as well as listing of the threats that could occur are to this point the same. However now, all possible threats have been identified and now have been taken care of to our best possible way of preventing attack. During fuzz testing we found that using simple weak passwords of just letters are easily crackable however passwords with special characters and other requirements will take an unknown time but be very long. Trying to break into the database through code injection would be a difficult task as well due to there being hardly any openings. Our static and dynamic analysis tools have been useful during the development with keeping our code clean and kept to a standard as well as seeing the execution of various functions and how they behave. The application will receive a FSR grade of passed FSR. All issues that had been found in the past during security meetings have been corrected and ensured to be covered. We have also made sure to include an updated privacy policy to reflect our privacy requirements to let users know how we deal with their data. Everything included in this report goes over the SDL requirements from security requirements, threat modeling, listing approved tools, etc. The current release version of the application includes all the issues we've previously identified and fixed. Below is our updated threat model.



Certified Release and Archive Report

The Budget Tracker 1.0 has a number of security features including 2-factor authentication, a secure database to store user’s information, and separation between user and administrative privileges. All these security features ensure our user’s financial information is secure as this application requires the input and storage of sensitive income and expense information. Some future development plans would be to improve the user interface for more streamlined experience. Also, adding additional financial features such as calculating your monthly savings, investing, etc.

In order to install our program, you must first install Meteor. Second, go to our GitHub repository and “Clone or download” our repository to your local system. Third, cd into the /app directory of your local copy of the repo, and install third party libraries with the command: $ meteor npm install. Once the libraries are installed, you can run the application by invoking the command: $ meteor npm run start. Fourth, Go to the browser and type http://localhost:3000 and begin using the BudgetTracker application!

Work Cited

- https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307404(v=msdn.10)?redirectedfrom=MSDN
- https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307403(v=msdn.10)?redirectedfrom=MSDN
- https://www.zdnet.com/article/top-10-web-service-security-requirements/#:~:text=The%20key%20Web%20services%20security,%2C%20data%20protection%2C%20and%20nonrepudiation.&text=Authentication%20ensures%20that%20each%20entity,it%20actually%20claims%20to%20be.
- https://www.pmi.org/learning/library/importance-of-security-requirements-elicitation-9634
- https://www.mitre.org/publications/systems-engineering-guide/enterprise-engineering/engineering-informationintensive
- https://docs.microsoft.com/en-us/power-platform/admin/security-roles-privileges
- https://www.veracode.com/security/owasp-top-10
- https://cheatsheetseries.owasp.org/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.html
- https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.174.6874&rep=rep1&type=pdf
- https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307393(v=msdn.10)?redirectedfrom=MSDN
- https://nodejs.org/en/blog/vulnerability/february-2021-security-releases/
- https://github.com/maierfelix/Iroh
- https://www.openwall.com/john/doc/MODES.shtml
- https://medium.com/rangeforce/nosql-injection-6514a8db29e3
- https://medium.com/rangeforce/meteor-blind-nosql-injection-29211775cd01

•