Team Name: Bug Catchers

Team Members: Chak Hon Lam, Su Lao, Ian Manzano, Shengtong Jin

Application Title: Just Budget

Type of program: Web Application

Development tools: Meteor, React, MongoDB, HTML, CSS, Javascript, Intellij (IDE), Github

Introduction

Just Budget will be a simple budget planner/tracker web application that allows the users to track and plan their spending with ease. To use Just Budget, users will have to enter the site in their preferred desktop browser, users can then either login or create their account. To create an account, users will have to input their email address, and choose a password. Once a user creates an account, they will be able to login and plan their monthly budget by entering their monthly income and spending, the application will then calculate the total savings the user will have by the end of the month.

Just Budget will be coded using HTML, CSS and Javascript, we will also be using Meteor and React as the foundational framework to our application. The reason we chose to use Meteor and React is because our team has gotten very familiar with both frameworks in our software development classes, as such we will also be using meteor-application-template-react to quickly get started on our project. The templates already include the Mongodb packages, which we will be using to manage and store the user data. We have also decided to use Intellij as our integrated development environment for this project, since we already have the code style setting for it from our software development classes, it will be easy for us to keep our coding style consistent within our team. Lastly, our team will be using Github to work, share and manage our project during its development.

Security and Privacy Requirements

Since Just Budget requires the use of users' private information(such as income and spending) in order to work it is important for us to make sure that these private information are kept secure and only its user has access to it, but at the same time we want to keep the user experience simple and convenient, we do not want the users to enter their income and spending every time they visit the site, which is why we will need the user to create an account to use the

application. This way the income and spending data will be stored on to the MongoDB database, and the users will have access to it every time they log on.

While having a user account means that it will be convenient for the user, it also creates additional security risks to the project. We will have to make sure that user login data and private information are kept safe or else others could get access to it. We have to make sure that these sensitive data are encrypted and only its user can view it. Another feature we will need is the ability to remove their data from the database when they decide to, if the user feels that they no longer require our service we have to give them the option to erase their data from our database.

To keep track of project progress, additional security flaws, other issues and bugs that may arise during the development of this project, we will be using GitHubs' project board "Automated Kanban" feature. If a team member notices any security flaws or bugs during the project's development, they can create and document the new issue in the To Do column. Once a member picks up and has begun working on the issue they will move the card from the To Do column to the Progress column. When the issue is resolved or if the problem has been addressed, the issue will be moved to the Done column.

Quality Gates (or Bug Bars)

Security Bug Bar:

- Elevation of privilege: The ability to execute arbitrary code
    - Example: injection of javascript
- Denial of service: Must be "easy to exploit" by sending a small amount of data or be otherwise quickly induced
    - Examples:
        - Sending a small number of packets that causes a service failure
        - Sending a small number of packets that causes the system to be unusable for a period of time
- Information disclosure
    - Cases where the attacker can locate and read information *from anywhere* on the system, including system information that was not intended or designed to be exposed
        - Example: Disclosure of PII (email addresses)

- Spoofing
    - Example: user masquerades as another user
- Tempering
    - Example: modifying user data
- Security features: Breaking or bypassing any security feature provided.
    - Example: using the site without logging in

Privacy Bug Bar:

- Lack of notice and consent
    - Example: storing users private information without notice
- Lack of user controls
    - Example: users are unable to edit/remove their data
- Lack of data protection
    - Example: data are stored without encryption
- Improper use of cookies
    - Example: users sensitive information are stored in a cookie without encryption
- Lack of internal data management and control
    - Example: admins are unable remove data off of database
- Insufficient legal controls
    - Example: user data is sent to a third party without a legally approved contract.

Risk Assessment Plan for Security and Privacy

As mentioned in above, Just Budget will require the users to enter their email in order to login or create an account. The user email serves as a way for us to contact the user after events of security breach or data leak. Their email address can also serve as a unique identifier for each user so we could identify and provide support to the user, for example, if a user forgot their password. We will notify the user that their email will be used for the above reasons and ask for consent to do so in the signup page. The user can alway visit the info page to review the public disclosure section, to see what we are doing with their email addresses. If the user decides that they no longer need to use our application, they can remove their email address by either deleting their account from the profile page, or by contacting our team and request us to remove their account from the database.

Other private information that we will be collecting is of course; the users' income and spendings, since we will need this information to help the users track and plan their budget, and as mentioned above we store this data to provide a simple and convenient experience. We will notify the users on how we will use the information and the risk of doing so on the bottom of the budget planning page. The user can alway visit the info page to review the public disclosure section, to see what we are doing with their income and spending information. The user can also edit or remove their data from the budget planning page if they wish to do so.

We will do our best to prevent unauthorized access to this sensitive information by encrypting the data and keeping the users informed on the state of the data. Our contact information will be found on the footer and the info page of our site, and we will be committed to answering any questions or concerns our users may have regarding the operation of the site or their data, as long as our site is up.

Design Requirements

For this application, We are going to use JavaScript, HTML, and CSS as our coding language. Also, we are going to implement Meteor, React frameworks to create a clean user interface with basic functionality. All the user data will be stored in MongoDB, which provides network isolation, role-based access management, and end-to-end encryption to help us protect the user data.

In our Just Budget application, we need to have a landing page. On the landing page, we need to show the user the basic introduction and the usage of our application. We will also need a sign-in and sign-up page. For the new users, they will need to sign up for an account to use our application. On the sign-up page, we will be taking the user's name, email, and password to our database. On the sign-in page, the user will be able to log in to their account with their email and password, both page's portals will be from the navbar on the landing page. After the user login, they will be led to the user-landing page. On this page, a huge calendar will be displayed in the middle, which shows the history of the user's spending and earnings in that month. Users will be able to select a month from the calendar, and add spending or earning to the month, once the user inputs the data, it will be saved and shown on the calendar. Lastly, we will have the user profile page, the portal to this page will be at the navbar of the user-landing page. On the user profile page, it will auto-calculate and display the total earnings and spending throughout the

month, and shows net-earning overall. Also, a delete account button will also be on this page, if a user decides to stop using our application, this button will remove all the user's information from the database. Meanwhile, we need to create a very simple, and clean user interface for users to interact with our application.

Attack Surface Analysis and Reduction

The privilege level for our users is the standard user account type. Users can access their persistent storage data only after logging into the application. Users can control their personally identifiable information such as name, address, phone number, and more. Any information that can identify or contact the user is not visible to other users. For sensitive PII, such as username, password, it should be handled according to the rules of Sensitive PII. For example, only collect necessary sensitive PII such as username and password. Data like ethnicity, religion, which maybe could lead to discrimination and facilitate identity theft should not be collected. Sensitive PII is invisible to other users or even the user himself/herself, e.g. when the user tries to enter the password on the screen, passwords need to be replaced with asterisks *. Users must obtain explicit consent before any PII is collected. User accounts data must be protected and have sufficient legal controls.

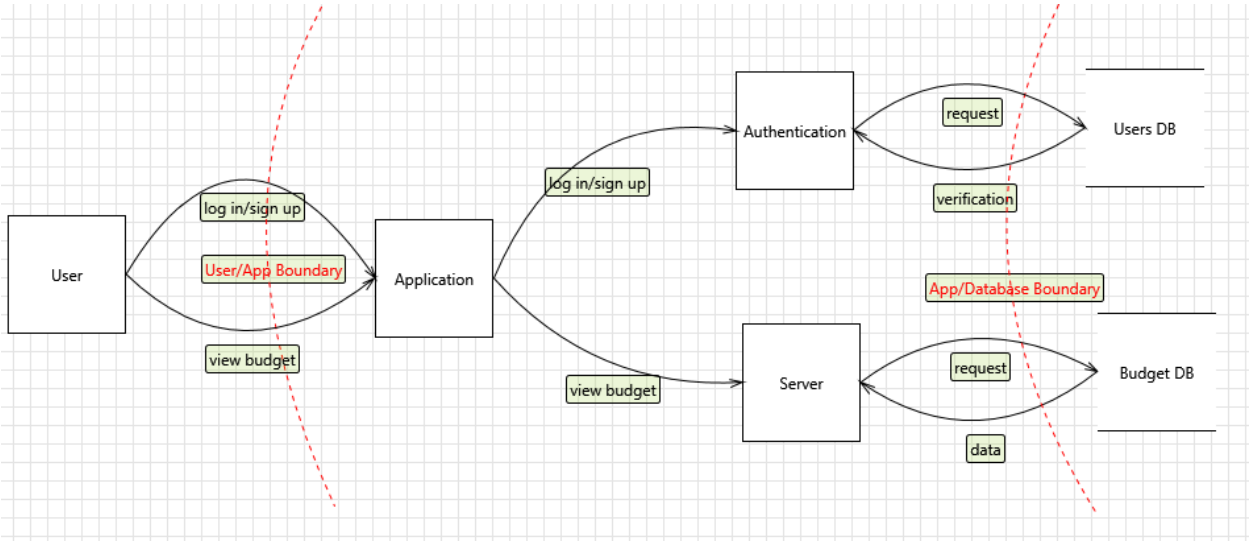
After performing an attack surface analysis on meteor-application-template-react similar to our application, we compiled a list of which our program might be vulnerable.

List of Vulnerability:

- Access controls and roles: users should only be able to access their data, but malicious users may try to exploit the vulnerability to access other user data or crack the authorization system to gain administrator control of the program.
- Authentication: hackers may gain access to people's accounts by stealing passwords or performing a brute force attack to find the password.
- Program directory: malicious users could try to create scripts to help find important data in the program directory. For example, the model program directory config contains a file settings.development.json that contains administrator account information, which means they can access the application as an admin.
- Domain/URL: use the specified URL to the specified page. For instance, use the localhost/#/signout URL to access the signout page.

- Collection publications: if the developer makes mistakes with the collection publishing, then the users will be able to view, edit and delete data that does not belong to them.
- Data packets: web applications involve TCP connections between client and server, hackers can command or Wireshark to see the raw bytes of packets that is what the client sends to the server, and what the server sends back to the client, which may include usernames and passwords and other information.
- Multithreading: malicious users may try to execute threads on the program to create a race condition, which may cause the program to lose updates or corrupt the entire program. Modify HTTP header, data or information may be leaked by sending modified HTTP headers, such as POST, DELETE, etc.

Threat Modeling



Possible Threats by Category

- Spoofing Identity
    - Spoofing of Source Data Store Budget DB
    - Spoofing of Destination Data Store Users DB
    - Spoofing of Destination Data Store Budget DB
    - Spoofing of Source Data Store Users DB
- Tampering with Data
    - The Users DB Data Store Could Be Corrupted
    - The Budget DB Data Store Could Be Corrupted

- Repudiation
    - External Entity Server Potentially Denies Receiving Data
    - External Entity Application Potentially Denies Receiving Data
    - Data Store Denies Users DB Potentially Writing Data
    - Data Store Denies Budget DB Potentially Writing Data
    - External Entity Authentication Potentially Denies Receiving Data
- Information Disclosure
    - Weak Access Control for a Resource
- Denial of Service
    - Data Flow response Is Potentially Interrupted
    - Data Store Inaccessible
    - Data Flow log in/sign up Is Potentially Interrupted
    - Data Flow request Is Potentially Interrupted
    - Data Flow view budget Is Potentially Interrupted
- Elevation of Privilege
    - Weakness in SSO Authorization

Implementation :

Approved tools :

Compiler/ Tool	Minimum Required Version and Switches/Options	Optimal/ Recommended Version and Switches/Options	Comments
Google Chrome	Most recent version	Any javascript es6 compatible browser	
IntelliJ IDEA	Most recent version		
node.js	Most recent version		
Meteor	Most recent version		
MongoDB	Most recent version		
git	Most recent version		
GitHub Desktop	Most recent version		optional
eslint	7.32.0		
testcafe	1.16.0		
babel/runtime	7.15.4		
classname	2.3.1		
core-js	3.18.0		
prop-types	15.7.2		
react	16.14.0		
react-dom	16.14.0		
react-router	5.2.1		
react-router-dom	5.3.0		
semantic-ui-css	2.4.1		
semantic-ui-react	1.3.1		
simpl-schema	1.12.0		
uniforms	3.6.0		

uniforms-semantic	3.6.0
uniforms-bridge-simple-schema-2	3.6.0
eslint-config-airbnb	18.2.1
eslint-plugin-import	2.24.2
eslint-plugin-jsx-a11y	6.4.1
eslint-plugin-meteor	7.3.0
eslint-plugin-react	7.25.3
eslint-plugin-react-hooks	4.2.0

Deprecated/Unsafe Functions

Unsafe or deprecated functions	Alternative safer functions or methods
== or != operators (two equals)	=== or !== operators (three equals)
var (declaration of the variable)	const or let (declaration of the variable)
let (on never change variable)	const (on never change variable)
new Object() (create object)	{} (use object literal syntax)
new Array() (create array)	[] (array literal syntax)
“don’t use” + variable	`use ${variable}` (backticks)
while or for loop	map
_.map or _.filter (lodash or underscore)	array.map or array.filter
{array.map(item => {item})}, missing key.	{array.map((item, index) => <li key ={index}>{item})}
eval()	Function() or JSON.parse()
db.collection.insert()	db.collection.insertOne() or db.collection.insertMany()
db.collection.remove()	db.collection.deleteOne() or db.collection.deleteMany()

db.collection.update()	db.collection.updateOne() or db.collection.updateMany()
HTTP.call(method, url, [options], [asyncCallback])	fetch()
util.print(), util.puts()	console.log()
class className extends React.Component { render() { return ( );}}	const className = () => { return ( );}
util.debug(), util.error()	console.error()
util.isArray()	Array.isArray()
Number.toInteger()	parseInt()
for each...in	for...of
UNSAFE_componentWillMount()	componentDidMount()
UNSAFE_componentWillUpdate()	componentDidUpdate()
_.matches	_.matcher

Static Analysis

For static analysis, we will be using the same tool that we have been using in our Software Development classes which is ESLint. This is an open-source tool that supports Javascript and is used to enforce coding standards. It is also supported in the IDE that we will be using for this project. ESLint automatically finds errors in our code as we write it, and it provides us with suggestions to improve or fix the code. In addition to using ESLint as we write code, we are also using continuous integration to ensure that our code passes all ESLint rules every time a team member commits code to the master branch. We can view whether it passes by going into the “Actions” tab on GitHub, then looking at the workflow run of the latest commit. If it passes you would see a green checkmark.

One of the advantages of using a tool like ESLint is that we can see our errors as we write code, and we would be able to fix most of the problems before running the application. This would save us a lot of time and make debugging less stressful since there will be fewer errors

when we run the application. Additionally, many static analysis tools come with built-in rules. Another advantage that ESLint has over other tools is that it is customizable, which means that we can create our own linting rule. This is a very powerful functionality since the built-in rules cover most of the common and important rules, but as we add new packages, we can customize new rules to avoid for example the deprecated or unsafe function we mention above, which will save us time and energy by avoiding them.

To use ESLint, we have to install NodeJS using npm. Once NodeJS is installed for our OS, we have to install the NodeJS plugin in Intellij by going into the settings. After that, we can configure the ESLint settings. Using the "Automatic ESLint configuration" option will configure ESLint with the default settings. The "Manual ESLint configuration" option can be selected if we want to customize the settings. Hence, we got the ESLint running in our IDE, so whenever we type some code that violates the rule, it will have a red wavy line under the code, and we can check which rule we violate by just hovering on the code, it will give detail of what we have done wrong. To fix the error, we can simply right-click on the code and click the option which ESLint provides to automatically correct it. Also, the green wave line will appear when there is a suggestion available to make your code more secure. ESLint also works on the command line interface, if you have too much file need to view, you can simply use the command "eslint --quiet --fix --ext .jsx --ext .js ./imports", which will run ESLint on all the .js files in the imports directory, and look for errors and suggestions.

After using ESLint a few times, we haven't had any problems with it. Since we have already used this tool in our Software Development classes, we already had ESLint installed and working in advance. It works as it did before, by pointing out errors in our code.

Dynamic Analysis

The dynamic analysis tool that we decided to use is called the TestCafe. TestCafe is a very powerful automated testing tool that uses Node.js end-to-end for website applications, it's open-source, and can be used on many different platforms such as Windows, Linux, and MacOS. There are several reasons we choose to use this as our dynamic analysis tool, first, it's easy to set up. We choose TestCafe because we can easily integrate this tool into our application, plus we already have a simple template implemented with the website template we use. Most importantly, Testcafe acts like a user with a mouse and keyboard, when testing it will "click" and

“type” on the website elements we programmed it to do, then it will check if the outcome is expected, which is great for real world use cases. Since our application might contain lots of different functionality we need to test, we will have to create a test plan for every page, and for every functionality within our application. TestCafe has its own built-in function to check if our UI is stable or not. Also, TestCafe is efficient for doing rapid tests. Since we want our application to be as secure as possible, after each time we update or implement new functionality and code, we will need to run the test to make sure it will not cause any errors or potential vulnerabilities. In this case, we would need to run the test very often, by using the TestCafe, as long as we set up the test correctly, we can run the test by simply calling a single command, in our case its “meteor npm run testcafe chrome:headless tests/*.testcafe.js -q --app”, and it will run through all the test code that we wrote, with the report on the console, which saves lots of time and ensures the security of our application.

During the development of our application, we’ve kept using the TestCafe as we move forward on our application little by little. We don’t have any difficulties at all since we all have experience with TestCafe from the past, the only challenge that we have is to apply the correct test case for each functionality in our application. However, as long as we set up the test correctly, it will run all the tests smoothly each time and save us lots of time. In our case, TestCafe helps us catch a minor error on the landing page, as we improve the UI for our application, we’ve done some CSS changes for our landing page, but after we run the TestCafe, it reports there is something that causes the landing page was not stable when loading up, which that we didn’t really realize cause the landing page was running fine, but after the report, we took a look, when loading the landing page, there is few milliseconds where there is multiple navbar and before it is back to normal, and after investigation its because some minor code error we use for our CSS that force the navbar to load. This is some of the success that we found by using the TestCafe, which not only checks the errors and vulnerabilities in our application but also includes the UI that might possibly break our application. Overall, we all agree that is a good experience by using the TestCafe as our dynamic analysis tool. With its flexibility and its own powerful built-in function, we can simply test out our application rapidly to ensure the security of our application in the development process.

Attack Surface Review

As we move forward with developing our application, the approved tools we used do not have any changes or updates, most of the approved tools we used are in their stable release version, so there is no reason to change or update them to the latest version which might contain possible vulnerabilities.

Approved tools :

Compiler/ Tool	Minimum Required Version and Switches/Options	Optimal/ Recommended Version and Switches/Options	Comments
Google Chrome	Most recent version	Any javascript es6 compatible browser	
IntelliJ IDEA	Most recent version		
node.js	Most recent version		
Meteor	Most recent version		
MongoDB	Most recent version		
git	Most recent version		
GitHub Desktop	Most recent version		optional
eslint	7.32.0		
testcafe	1.16.0		
babel/runtime	7.15.4		
classname	2.3.1		
core-js	3.18.0		
prop-types	15.7.2		
react	16.14.0		
react-dom	16.14.0		
react-router	5.2.1		
react-router-dom	5.3.0		
semantic-ui-css	2.4.1		

semantic-ui-react	1.3.1
simpl-schema	1.12.0
uniforms	3.6.0
uniforms-semantic	3.6.0
uniforms-bridge-simple-schema-2	3.6.0
eslint-config-airbnb	18.2.1
eslint-plugin-import	2.24.2
eslint-plugin-jsx-a11y	6.4.1
eslint-plugin-meteor	7.3.0
eslint-plugin-react	7.25.3
eslint-plugin-react-hooks	4.2.0

Verification

Fuzz Testing

Javascript integer overflow. Our applications are budget planners, so it is most important to ensure that the application never underflows or overflows to protect the correct data. To test for integer overflow, simply add a very large number to the budget planner's ending balance. As a result of fuzzing, if a positive number exceeds Number.MAX_VALUE (defined by Javascript) or a negative number exceeds Number.MIN_VALUE (defined by Javascript), will not add up. For example, Number.MAX_VALUE + 1 is not equal to Number.MAX_VALUE + 1, but Number.MAX_VALUE. This is the same as Number.MIN_VALUE - 1. Once the number reaches MAX or MIN, nothing can be changed other than subtracting MAX or adding MAX back to 0. One way to work around this vulnerability is to ensure that the value never exceeds MAX or MIN, but Javascript provides a minimum safe integer(Number.MIN_SAFE_INTEGER) and a maximum safe integer (Number.MAX_SAFE_INTEGER). So a better way to be safe is to ensure that the value never exceeds the safe minimum or safe maximum. Our solution was to add

an if/else statement to check if the value was in the range between MAX and MINI. For example, Number.MIN_SAFE_INTEGER <endingBalance &&endingBalance < Number.MAX_SAFE_INTEGER. This will fix the Javascript integer overflow vulnerability.

Define an approach that ensures all data are explicitly validated. Our application collection and input form use simpl-schema to ensure all the incoming data is valid. For example, in the Budget collection, this.schema = new SimpleSchema({ name: String, amount: Number });, meaning that the name can only accept strings, while the amount field will reject any input type that contains anything other than numbers. And in our code we use 'uniforms-bridge-simple-schema-2' to implement the same schema into the form so it is consistent. For testing, I tried different types of input, especially those fields in the schema that assign numeric types. I type all other types of characters except numeric types. As a result, input forms and collections will reject anything different from the type assigned to them, e.g. numeric types will not accept anything other than [0-9]+. In conclusion, simpl-schema ensures that all data is explicitly validated when trying to create or update data and documents in a collection.

Access control. It is important to ensure that the application has proper authentication and access restrictions implemented correctly. Authentication uses Roles from 'meteor/alanning:roles'. All the access restrictions like ProtectedRoute and AdminProtectedRoute are defined in '.\just-budget\app'. ProtectedRoute checks for login information before routing to the requested page, if not, it redirects to the login page. AdminProtectedRoute is similar to ProtectedRoute, but it also checks if the currently logged in user is in the admin role. Access control is tested by trying to point the browser to a hidden page using the page URL. For example, a user without the admin role tries to access the admin page. Without an administrator account, everything related to the administrator role will be hidden. However, if the user knows the admin page URL, it can still be used to force the web browser to visit the hidden page directly. So if a malicious attempt is made to access a page without the correct role, the page will redirect to the login page. Therefore, we can conclude that our application is safe in terms of page access control.

Static Analysis Review

Our team has been using Eslint as our primary static analyzer, before every commit our team members would run Eslint on their machine to check if the changes they made contained any errors or warnings. This approach has been serving us well, so far most if not all of the reported errors and warnings are simple mistakes like, missing semicolon or no newline at end of file. While we are satisfied with the outcome, it is also worth noting that Eslint is a rather basic linter, it does not possess the ability to check for more difficult issues such as Cross Site Scripting vulnerabilities. One member of our team suggested that we add CodeQl as another static analyzer tool since he has experience with it in his work. CodeQL is a code analysis engine that query code as though it were data, it is great for detecting vulnerabilities. The downside of CodeQL is that it take a bit of time to setup and it is not integrated into our project, if we want to query our code, we would first have to generate a code database with CodeQL, the process takes ~15mins to complete and we will have to repeat the entire process if there is any change in our code, it is not something that we would run every commit. After we ran CodeQL, the report showed a couple warnings and errors, but none of them are from the code that we wrote, this led us to think that it could be false positives. Overall, our project is looking great from our current static analysis review.

Release

Dynamic Analysis Review

The tool that we chose for dynamic analysis was TestCafe, which we are still using for our project. Previously, we had only set up TestCafe tests for the landing, sign in, and sign out pages. These tests ensure that these three pages are working properly. Since then, we have added a new test for the create planner page which will ensure that users are able to create a new planner. We also use this test to verify that the insert function is working properly. We plan on creating more tests for the budget planner page to ensure that users are able to manage their budgets, and to verify that the update function is working properly; and also another test for the feedback page. We have not run into any problems with TestCafe yet, and after running the tests on our project, we found that our project is working as intended since all tests were passing.

Incident Response Plan:

Role	Name	Description

Escalation manager	Chak Hon Lam	The escalation manager will be responsible for the creation and management of the entire team and is responsible for driving the process to complete each member's task in the incident response. Also, this will be the one who makes the final decision for the team.
Legal representative	Su Lao	The legal representative will contact the professional lawyer to help resolve any legal appeal during the response. This position is consistent throughout the response. The representative is responsible for being consistent throughout the process.
Public relations representative	Shengtong Jin	The PR representative will serve as the organization's representative, explaining problems and solutions to the public throughout the response process, answering questions and eliminating user panic. Answers must be consistent with the escalating manager's final decision.
Security engineer	Ian Manzano	A security engineer will be the one who solves application vulnerabilities. Security engineers need to fix bugs as soon as possible, so work overtime until everything is fixed.

Submitting Privacy Escalation Requests

Contact email: lamtechmailguy@gmail.com

This is the feedback email used on the feedback page and also for urgent contact with our team.

Privacy Escalation Response Process

If the escalation manager believes that the escalation requires more information. We will start by building an incident response channel on our development group Discord server, as we have been using Discord throughout our development. The primary purpose of the channel will be to connect about potential privacy upgrades. The primary purpose of this channel will be to contact panelists as they work on privacy escalation requests. Discord channels will be used by the entire team to determine the source, impact and breadth of the escalation, as well as the effectiveness of the incident. Then, aggregate all known informational facts. Create a timeline log that includes when the event started, what happened during it, and when the escalation is expected. Also, talk to employees who have been working to find out.

Execute the response: the escalation manager will create a resolution plan and distribute the plan to the privacy escalation core team. Each member of the group will act according to the role assigned to them. Escalation managers can also assign workloads to other unrelated roles to ensure that all aspects of the escalation are resolved on time. If any issues require resolution during the escalation process, the decision will be made by the escalation manager.

Final Security Review

Threat Modeling:

We have reviewed our threat model and have taken measures to prevent each threat. To prevent ‘Spoofing Identity,’ we require users to enter a valid email and password. The user will be given an error message if their email or password does not match an account in the database. The login page will also limit the user’s ability to login after five failed attempts. For ‘Tampering with Data’ and ‘Elevation of Privilege,’ we only allow users to do simple functions such as creating and deleting a planner, or adding and removing income/spending. We have removed the admin page for security reasons. To delete an account, we ask users to contact an administrator by using the “Send us your feedback” button at the bottom of the page. We are also using simpl-schema to ensure that all user input is valid and to prevent any injection attacks. For ‘Repudiation’ and ‘Denial of Service,’ we provide users with instructions to run the application on their own computers. For ‘Information Disclosure,’ we do not request any additional data

from the user after they have signed up. Any income/spending data that the user chooses to delete is also completely removed from the database.

Static Analysis:

We have been using ESLint as our static analysis tool and we have not run into any issues. It has been a very useful tool for improving the quality of our code, especially since we were working in a team. We have set up our workflow to run ESLint every time one of our team members pushes an update to the master branch. We have the green checkmark on our latest commit, which tells us that our code is satisfactory.

Dynamic Analysis:

We have been using TestCafe as our dynamic analysis tool and have set up tests to ensure that all pages are working properly. After running our TestCafe tests a final time before release, we found that all tests are still passing. This is good because it tells us that our application is working as intended.

FSR Grade:

After running the program a few times and testing all the features, we have decided that our program should get a grade of Passed FSR. Every security issue identified by the tools we used (threat model, static analysis, dynamic analysis, quality gates/bug bars, etc.) were taken care of, and we have concluded that our project meets all SDL requirements.

Certified Release & Archive Report

Link to the release version of the program:
https://github.com/bug-catchers/just-budget/releases/tag/v1.0.0

Our application, Just Budget, allows the user to sign up for an account and create a plan for their monthly budget by entering their monthly income and spending, the application will then calculate the total savings the user will have by the end of the month. Also, there is a feedback page where users can report their feedback, including asking to delete their own accounts. This is version 1.0.0 of our application. For the further improvement of our application, we would like to add the functionality of searching specific events in your monthly or yearly income and spending, which allows users to easily look at events in their plan. Also, we are considering adding different categories for each spending, by the end of each month, the

application will auto-generate a report that shows in which category you spend the most or least, helping users to organize and plan their monthly budget better.

Installing & Deploying Just Budget Locally

1. To run this application on your own computer, you will first need to install Meteor.
2. Once Meteor is installed, download the Just Budget project from GitHub as a ZIP file.
3. Extract the content of the ZIP file, from there open command prompt and run these commands:

```
cd app
meteor npm install
```

4. Once the program has finished installing the necessary packages, you can run this command to deploy the application:

```
meteor npm run start
```

Which should produce the following output:

```
PS C:\Users\chakh\Documents\GitHub\just-budget\app> meteor npm run
start

> meteor-application-template-react@ start
C:\Users\chakh\Documents\GitHub\just-budget\app
> meteor --no-release-check --exclude-archs
web.browser.legacy,web.cordova --settings
../config/settings.development.json

[[[[[ C:\Users\chakh\Documents\GitHub\just-budget\app ]]]]]

=> Started proxy.
 => Started MongoDB. I20220430-17:01:02.571(-10)? Monti APM:
completed instrumenting the app
=> Started your app.

=> App running at: http://localhost:3000/
    Type Control-C twice to stop.
```

*Note that if you are running the application for the first time, it might take a few minutes for the application to load.

5. You can now connect to http://localhost:3000/ to access the application.

Uninstall Just Budget

1. Delete the file where Just Budget is located.

Developer notes

We use meteor-application-template-react as a template for our project.

For the release of Just Budget we removed the send email from the Feedback page due to security reasons, if you were to deploy Just Budget the email feature will NOT work.

The feedback email is a proof of concept on how we could maintain the project and response to future threats.

The admin page did not implement as planned due to security reasons. Since we are using meteor for our application, meteor's default is that users would not able to modify or delete an account on the client-side, the function calls Meteor.users.remove() where prohibited because meteor does not want to take any security risks since this function call might mess up the database if there is vulnerability. So we decided to add an option on the feedback page, where users can make a request to delete their account. If we deploy the application, once we receive the request, we will manually log on to the Mongo database and remove the account information from the database.

To reset the application, open command prompt where the application located and run these commands:

```
cd app
meteor reset
```

Links

- Repository: https://github.com/bug-catchers/just-budget
- Wiki: https://github.com/bug-catchers/just-budget/wiki
- Readme: https://github.com/bug-catchers/just-budget/blob/master/README.md
- Release: https://github.com/bug-catchers/just-budget/commits/v1.0.0

Github repository link:
https://github.com/bug-catchers/just-budget

References site:

Appendix K: SDL privacy escalation response framework (sample). Microsoft Docs. (n.d.). Retrieved May 1, 2022, from https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307401(v=msdn.10)?redirectedfrom=MSDN

Arce, I., Clark-Fisher, K., Daswani, N., DelGrosso, J., Dhillon, D., Kern, C., … Others. (2014). Avoiding the top 10 software security design flaws. IEEE Computer Society.

Docs: Meteor api docs. Meteor API. (n.d.). Retrieved February 22, 2022, from https://docs.meteor.com/

Handle number overflow in JavaScript. Algotech Solutions. (2021, April 2). Retrieved April 10, 2022, from https://www.algotech.solutions/blog/javascript/handle-number-overflow-javascript/

Longshotlabs. (n.d.). SIMPL-schema: A JavaScript schema validation package that supports direct validation of mongodb Update modifier objects. GitHub. Retrieved February 22, 2022, from https://github.com/longshotlabs/simpl-schema

Meteor Api Document. Meteor API Docs. (n.d.). Retrieved February 22, 2022, from https://docs.meteor.com/api/http.html

Node.js V17.5.0 documentation. Deprecated APIs | Node.js v17.5.0 Documentation. (n.d.). Retrieved February 22, 2022, from https://nodejs.org/api/deprecations.html

Philip, J. (n.d.). Improve code quality using ESLint in Intellij. E24: Improve code quality using ESLint in IntelliJ | Review ICS 314. Retrieved February 22, 2022, from http://courses.ics.hawaii.edu/ReviewICS314/morea/coding-standards/experience-install-eslint.html

Pluggable javascript linter. ESLint. (n.d.). Retrieved February 22, 2022, from https://eslint.org/

React top-level API. React. (n.d.). Retrieved February 22, 2022, from https://reactjs.org/docs/react-api.html

Welcome to the mongodb. MongoDB Documentation. (n.d.). Retrieved February 22, 2022, from https://docs.mongodb.com/

What are uniforms? uniforms. (n.d.). Retrieved February 22, 2022, from https://uniforms.tools/docs/what-are-uniforms/

CodeQL. (n.d.). Retrieved April 11, 2022, from https://codeql.github.com/