Derrick Luyen
Haochen Wang
Eddie Wen
Micah Kim
ICS 427
May 31, 2020

Secure Development Life Cycle: Requirements and Design

Introduction

Team Name

Team Splendor

Team Member List

Derrick Luyen, Haochen Wang, Micah Kim, and Eddie Wen

3. Application Title, Description and Functional Requirements Specification

The title of our application is **Flick Critik**. The application is essentially a site that lets users create their own accounts in order to review movies provided by streaming services. We will allow users to provide their own reviews provided that they have an account. They can give their own ratings and add comments as well. Users will be given a pool of movies to choose from that they can review.

Type of Program

Our program will be a web-based application that can be hosted on the localhost of a computer and possibly used as a web application as well.

Development Tools

We will be using JavaScript as our language for this project. For the IDE, we will be using IntelliJ. We will likely be using React for the UI services. The selected development language is a Javascript Web framework written in Node.js, specifically Meteor. The selected database tool is MongoDB.

Requirements

Security and Privacy Requirements

Security Requirements:

The program would ensure that account input information matches with the database. The program must validate all user input to not exceed the limit specified, which is used to prevent buffer overflow. The program will try to prevent injections (SQL Injection) flaws and Cross-Site Scripting for all the user input fields.

Privacy Requirements:

The administrator user has a list of all the user's accounts and has the capability to delete accounts. The administrator can view all the movies with their reviews and comments, furthermore, make their own reviews and comments. The standard user can view all the movies with their reviews and comments, furthermore, make their own reviews and comments. The user that does not have an account, or is not logged-in will not be able to view any of the movie reviews.

The system of keeping track of security flaws that may arise throughout the development process is documentation. For each security flaw found, we will record what the security flaw is, when the security flaw was found, what version/stage the program was at, which files and/or sections the security flaw may affect, and additional description about the security flaw. Once a security flaw is resolved and patched, the documentation for that security flaw will be marked as resolved, how it was fixed, and what version of the program it was fixed at.

Quality Gates

Privacy:

End User Scenarios	Admin Scenarios
<u>Critical</u> ● Lack of notice and consent *Example: not notifying the user of the usage of their account information, ratings, and reviews* ● Lack of user controls *Example: user cannot control the sharing of their personal information, ratings, and reviews* ● Lack of data protection *Example: user account information, ratings, and reviews can be collected from the database without user authentication* ● Lack of internal data management *Example: control of user's information is not limited to administrators and user* ● Inadequate legal usage *Example: user's information, ratings, and reviews is shared with a outside party without permission*	<u>Critical</u> ● Lack of administrative control *Example: administrator cannot control the usage of information for the application* ● Inadequate privacy procedures *Example: administrators do not have a privacy policy*
<u>Important</u> ● Lack of notice and consent *Example: not notifying the user of the usage of their ratings and reviews* ● Lack of user controls *Example: user cannot control the sharing of their ratings and reviews* ● Lack of data protection *Example: user ratings and reviews can be collected from the database without user authentication*	<u>Important</u> ● Lack of administrative control *Example: administrator cannot protect the collection of information from the application* ● Inadequate privacy procedures *Example: not all administrators follow or is aware of a privacy policy*
<u>Moderate</u> ● Lack of notice and consent *Example: not notifying the user of the administrator's ability to view their information* ● Lack of user controls *Example: user cannot control the moderation*	<u>Moderate</u> ● Inadequate privacy procedures *Example: administrators have and follows a privacy policy but does not have adequate protection for the disclosure of it*

of their account	

Security:

<u>Critical</u> ● Unauthorized obtaining of roles *Example: users can obtain administrative abilities without permission* ● Unauthorized control of application *Example: users can alter application without permission*
<u>Important</u> ● Data overflow *Example: users can cause errors by overflowing the application with data* ● Spoofing *Example: users can pose as admins*
<u>Moderate</u> ● External links *Example: users can post links in reviews that leads to harmful websites without moderation*
<u>Low</u> ● Wrongful use of application *Example: users can use review section for non-review purposes and give dishonest ratings without moderation* ● Spamming *Example: users can spam in review sections without moderation*

Risk Assessment Plan for Security and Privacy

The initial assessment for both security and privacy risks is to determine the impact towards the program, including the description and their behaviors, and an estimation of work needed to be compliant. Next, is to try to find ways to lower the risk and compile a compliant design towards the risk using the threat model. After the design has been reviewed, implement the design to resolve security or privacy risk. Then, test the program using different test cases to verify if the program complies with the security or privacy requirements. Lastly, if the issue has been resolved, then write up a documentation as a follow-up about the security or privacy risk.

Design

Design Requirements

One design feature we will have is a rating feature. This feature will allow the user to rate the movie of their choice on a given scale (from 0 - 5 stars). Another feature we will have is a comment feature which allows the user to add some additional comments on what they thought of the movie. We will also have an admin feature that is different from a regular user feature which allows us as the developers to control the site if any problems arise. We will have a page that shows the users what movies they can review so they can pick which ones they want. We will have a form page that the users will review the movies in which will contain the rating feature and the comment feature. We will also make sure to have a sign up form to let users sign up for new accounts if they wish to. We will also need a database to store all the account information in so we will likely be using MongoDB for that like we mentioned above. We will also have a feature for admin only that will allow them to delete accounts if a user were to get out of hand. There is also another feature for admin that allows them to view the list of user accounts that have been registered on the site. We will also make sure to impose limits on fields that require user input in order to make sure there is no buffer overflow.

Attack Surface Analysis and Reduction

Privilege Levels:

Admin - can delete accounts, can view list of all accounts, can make reviews and comments, can view reviews and comments from others

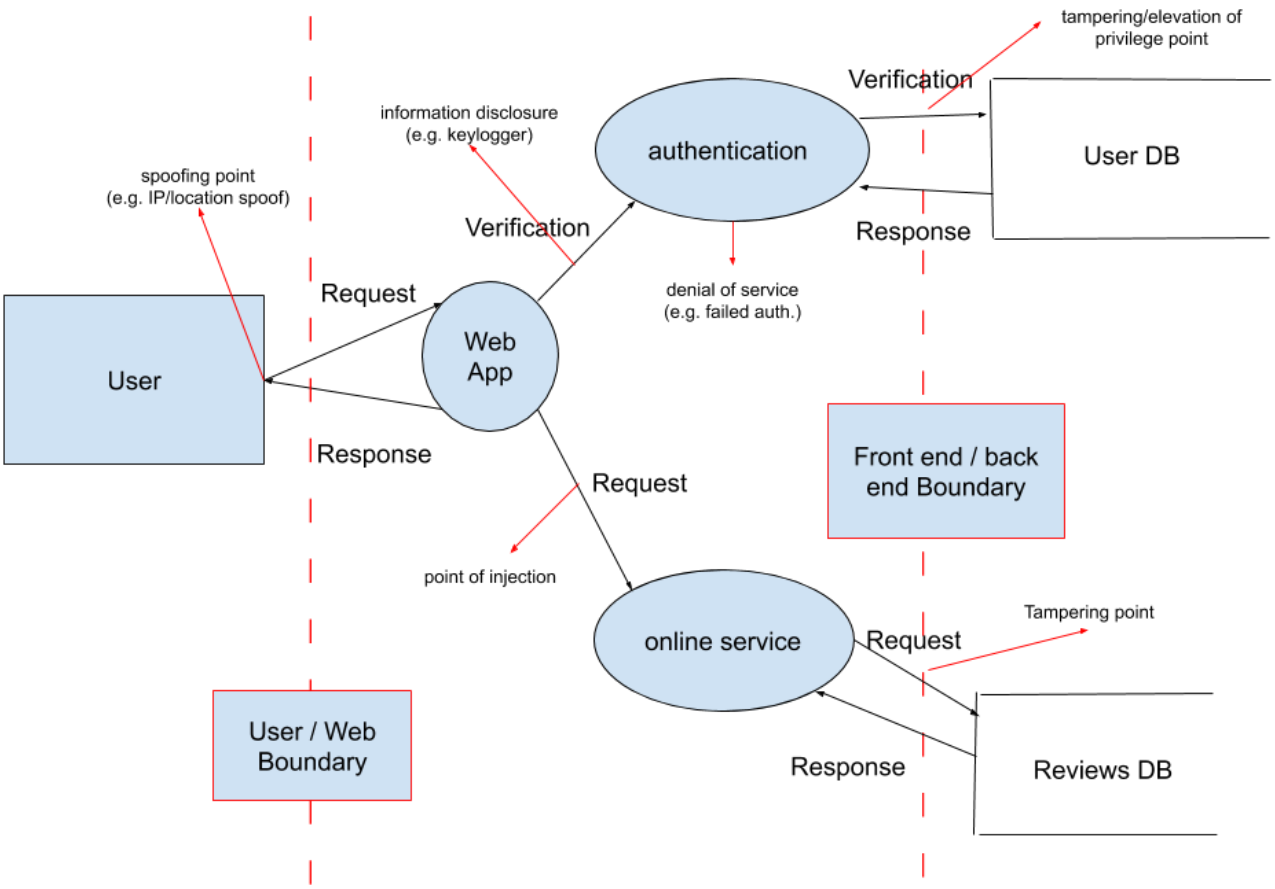
User w/ account and signed in - can make reviews and comments, can view reviews and comments from others

User w/o account or user not signed in - cannot do any of the above, can sign up for an account to get access to features that users w/ accounts have

List of vulnerabilities:

- Leaked passwords / unencrypted passwords
    - This could happen if passwords are not kept secure / encrypted, hackers could find out valuable information about the user
- Buffer overflow by way of SQL injection
    - This could overload the database / server thus causing a crash, likely would occur by manipulating user input
- Accessibility to objects
    - Developers left access to a variable on public when it should be private thus letting anyone be able to access it

Threat Modeling



Threats by Category (STRIDE):

Spoofing:

- Spoofing the user (identifies as an admin when they are not)
- Website Spoofing

Tampering:

- Corrupting data
- Changing user passwords
- Cross-Site Scripting

Repudiation:

- Misrepresentation of users
- Rejecting service for users

Information Disclosure:

- Leaking personal info of users
- Using user data without consent

Denial of Service:

- Denying service to those that either fail authentication or violate policies
- DoS attack by hackers causing overload

Elevation of Privilege:

- Letting variables be publicly accessible rather than private or protected
- Imitation of a admin to get access to user data

Implementation

Approved Tools

JavaScript Tools

Name	Version	Source
node.js	v12.18.0	https://nodejs.org/en/

JavaScript Libraries/Frameworks

Name	Version	Source
Meteor	v1.10.2	https://www.meteor.com/
React	v16.13.1	https://reactjs.org/
react-router	v5.2.0	https://www.npmjs.com/
react-router-dom	v5.2.0	https://www.npmjs.com/
semantic-ui-react	v0.88.2	https://www.npmjs.com/

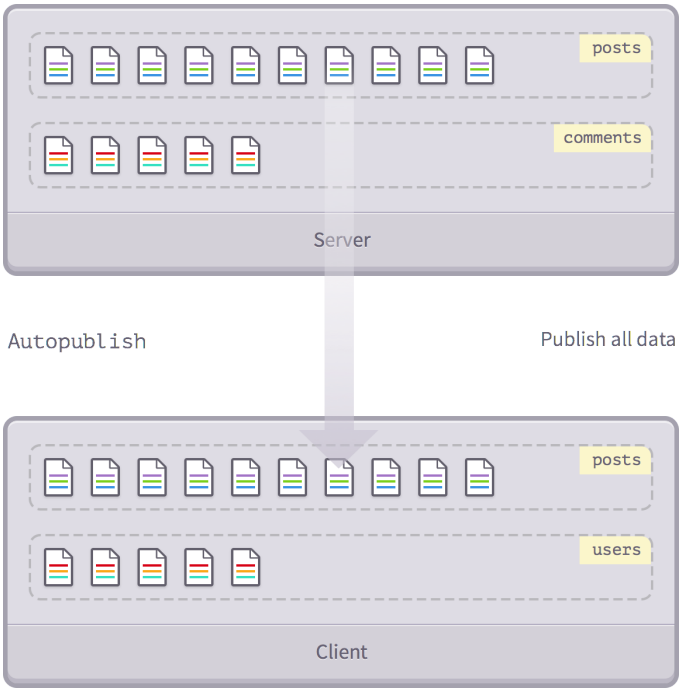
Code Editors

Name	Version	Source
IntelliJ Idea	v2020.1.2	https://www.jetbrains.com/

Deprecated/Unsafe Functions

- eval()
    - unsafe function used in Node.js
    - It is unsafe because it executes whatever code the user passes to it. Therefore, if the user is a hacker and they type something dangerous, bad things could happen which is why it is not recommended for use in code.
    - One alternative is to use JSON.parse() which essentially does the same thing as eval() except it is safer to use.
    - Another alternative is to use Joi which essentially validates the input so that the user cannot type anything they want; they would have to type something that fits what the input is looking for.
- dangerouslySetInnerHTML
    - unsafe function used in React

    - It is unsafe because generally, setting HTML in the area you are coding in is something that is risky. It allows users to be exposed to a Cross Site Scripting attack even if it was unintentional.
    - The developers of React remind us that it is unsafe to do so by adding in dangerously into the function name. It is something that we can do but it is not recommended at all.
    - An alternative is to use a library to convert raw HTML code to React DOM structure. Some libraries to parse raw HTML code are ‘html-react-parser’ and ‘html-to-react’.
- componentWillMount, componentWillUpdate, componentWillReceiveProps
    - unsafe functions as of React v16.3.0
    - Labeled with the UNSAFE_ prefix tags (ie. UNSAFE_componentWillMount)
    - It is unsafe because these functions were misused with the current instance, ‘this’. These functions were often misunderstood that could lead to misuse and exploits.
    - Alternatives:
        - componentWillMount componentDidMount
        - componentWillUpdate getSnapshotBeforeUpdate
        - componentWillReceiveProps getDerivedStateFromProps
- meteor create myApp
    - Unsafe command due to the two packages (which are collections of functions): autopublish and insecure
    - Autopublish basically lets the client see the server side database due to the fact that it mirrors it to the client
    - Example of autopublish in action:



- Insecure basically allows the client modify data in the database
- Although these two packages are insecure, they greatly speed up the initial stages of app development by postponing security considerations until a later stage
- No alternative, because as soon as they are removed, the app will stop working until the proper procedures are reestablished

Static Analysis

The tool that our group is using for static analysis is ESLint. ESLint is a static code analysis tool that focuses on identifying problems in your code. It can be used to solve code quality problems as well as code styles problems too. ESLint is basically a code checker but one thing to note is that it does not require code to be compiled to check it. It can check your code for these errors mentioned above while you are coding. I feel like this has its advantages over static analysis tools that check your code only upon compilation because it can teach you how to code better and to not make those mistakes in the first place as you are fixing the errors right then and there while you are coding. It enforces the coding style and makes sure that you as the developer sticks to it. To use ESLint, you must first install NodeJS and NPM. For NodeJS, it is recommended to install the “LTS” version. After doing that, you then must activate the NodeJS

plugin on IntelliJ. Using ESLint with IntelliJ, it can easily be set up by selecting the Automatic ESLint Configuration option. After completing all of these steps, it is now ready to use and it is a good way to make sure your code follows the coding guidelines. For using ESLint, in our group, we have all used it before in a previous class and found it to be super helpful in terms of finding the mistakes that we had made in code. As for this assignment, so far, we have not run into any problems while using ESLint. It has once again proven to be a helpful tool as it keeps us in check and points out any flaws that we have in the code right away. One particular success that we had that was memorable was when one member was working on their part of the code and his page was not loading. ESLint easily found the error that was hidden in the file that the member had missed out on. Once that error was fixed, the page finally loaded and it was thanks to ESLint that the problem was solved. Overall, ESLint is a very helpful tool to use while you are coding as it helps you learn from your mistakes and teaches you how to not make those same mistakes the next time.

Verification

Dynamic Analysis

The tool that our group chose to use for dynamic analysis for this project is called Iroh.js. Iroh.js is an open source project on Github created by Felix Maier. This open source project allows users to apply different tests during runtime to test areas such as call tree graphs, type checking, and code quality. It also intercepts functions that are unsafe like eval, which we mentioned above, and setTimeout which is a function that calls eval. What Iroh basically does is it “records” what is happening inside your code by addings listeners but this does not change the functionality of the code. It instead enhances your ability as a developer to make sure you are able to know exactly what your code is doing by giving you a good visualization of the calls being made.

When using this dynamic code analysis tool, at first, we had a lot of difficulties using it. We could not figure out how it worked as there were not many examples of the usage of it

online. Eventually, we found a good example on the site under the Node.js example area which was under the examples section. After following that example, we were able to start using iroh.js for our project. We made sure to test certain areas of our code to see if they were producing what we expected it to produce. It indeed produced the desired results. In terms of difficulties, the main one was mostly just figuring out how to make it work in the first place as we all struggled with that at first. This was something new to all of us so we had our little bouts with it. It was mostly what we expected when we ran the tests. We will continue to use iroh.js in the future as we further develop our program in order to assure that everything is running smoothly and the way we intended it to. This kind of dynamic analysis tool is very helpful for any coding project as it helps you keep track of things you expect to see and lets you know what each part of the code is doing.

The official website for iroh.js is https://maierfelix.github.io/Iroh/, and the repo and documentation for this project is at https://github.com/maierfelix/iroh.

Attack Surface Review

There has been no development changes, updates, and patches in any of our approved tools. There also has been no new vulnerabilities reported in any of our approved tools. However, there was a small update for npm (the package manager under node.js) from version 6.14.4 to 6.14.5, but this did not add any additional vulnerabilities to our code. Therefore, the approved tools list is the same as the one above and looks like the following:

Approved Tools

JavaScript Tools

Name	Version	Source
node.js	v12.18.0	https://nodejs.org/en/

JavaScript Libraries/Frameworks

Name	Version	Source
Meteor	v1.10.2	https://www.meteor.com/

React	v16.13.1	https://reactjs.org/
react-router	v5.2.0	https://www.npmjs.com/
react-router-dom	v5.2.0	https://www.npmjs.com/
semantic-ui-react	v0.88.2	https://www.npmjs.com/

Code Editors

Name	**Version**	**Source**
IntelliJ Idea	v2020.1.2	https://www.jetbrains.com/

Fuzz Testing

As there are various techniques to hack a website, one attempt we used to hack into our own program is through the login system. Specifically, we attempted to check if multiple accounts can be created from a single email account. On all of our attempts, none of the user accounts that had an existing email account in the user database were created. Our program returns a message stating registration is unsuccessful. This is a good security feature because having multiple accounts from the same email address can create complications. One thing is that having multiple accounts from the same email address is unnecessary since all email addresses are unique. Another thing is if our program can reset account passwords, it would technically reset multiple accounts under the same email address, which would not be secure. Hence, a security feature in our program is to prevent the creation of multiple accounts with the same email address.

Another security feature we have tested is to prohibit unsuccessful login attempts if the attempted password does not match with the associated email address. When attempting five consecutive incorrect passwords towards an existing account, the program recognizes it and does not give access to the account while stating an unsuccessful message. Once it reaches the sixth unsuccessful attempts and onward, the program states that there were too many unsuccessful requests and times out the person for about eight seconds. A future implementation could be to

make the time out duration longer to prevent brute force attack on passwords. This is another important security feature that will only allow the correct email address and its corresponding password to login to the program.

The third security feature is to check if the unauthorized user can access a specific page of the program via the address bar. The first test case is if the user who is not logged in can access the website pages. By changing the URL links to the page, this user was unable to access any of the pages except the default ones that were the Landing page, Sign In page, and Sign Out page. The next test case is if the logged in user can access the admin page. Again, this user was unable to access the admin page. Each time it is an unsuccessful attempt, the user's page is redirected to the Sign In page. This is a crucial security feature because it blocks out any unauthorized access to the respective pages in our program.

Static Analysis Review

For the static analysis portion of this project, we have continued to use ESLint as our static code analysis tool. ESLint has been very helpful throughout the experience of making this project and we have had a great experience with it. ESLint has assured us that our project is running smoothly and that it helped make the program's code consistent despite multiple people editing the program. We were also able to use it to help find errors that were made in order to help each other fix the code to make it fully functional. When looking through the project as a whole, there are no major errors given by ESLint, but there are a couple of warnings here and there. However, the warnings are nothing to worry about as they do not affect the performance or security of the program. Some of them are a result of redundancy which is fine in our case. Other than that, ESLint has been a very helpful tour throughout the journey of coding our project. It has provided ways for us to check the syntactic correctness of our program while we are coding so that everything is able to execute the way we expected it to. If there is an error, ESLint will point it out right away to us and we are able to fix it quickly before it becomes an issue. Overall, our experience with ESLint has been a great one. Like we mentioned before, we have used it in other classes / projects as well and it has always been a helpful tool for us when coding. We likely will

continue to use it as we progress forward into the future and we highly recommend others to follow us in using the tool as well as it will be a great benefit to them as well.

Dynamic Analysis Review

For the dynamic analysis portion of this project, we have continued to use Iroh.js as our dynamic code analysis tool. During the week of assignment 3, we were trying to figure out how to properly implement it in our app. We had figured out how to use it in some ways but we felt we were still lacking in our knowledge about how to fully take advantage of it. This week, we have made progress in implementing it and understanding how Iroh.js operates. We have taken a deeper look into how it works and what it can truly do. We have also continued to use Iroh to dynamically analyze our code in certain areas to make sure those parts of the code are doing exactly what we expect them to do. With Iroh.js, we have also been able to successfully visualize the call stack for function calls. We did not test the parts that relate to React as those areas are mostly UI based so we can know what to expect just by looking at it. However, for the parts that involve Javascript functions and such, we used Iroh in those areas to make sure those functions do exactly what we need them to do. Iroh itself is an exceptional dynamic analysis tool and it really allows us as developers to visualize what our code is doing. It gives us a good sense of what our code actually does. This is useful when we are finding errors if they exist as it helps to pinpoint where exactly the errors could occur based on the visualizations that are provided by the tool.

Release

Incident Response Plan

Privacy Escalation Team:

Escalation Manager: Derrick Luyen

Highlights or flags or certain issues within the project, so that the team can appropriately respond to these situations and monitor the resolutions. Also, helps

to re-prioritize and reassign a situation to a satisfactory completion, depending on the severity/nature of the incident.

Legal Representative: Haochen Wang

Provides legal support during the incident response and will stand as a legal representative for our team if the incident were to be escalated into a more severe manner.

Public Relations Representative: Micah Kim

Communicates with the general public by planning/directing media posts in order to enhance our public image. Will also be in contact with potential collaborators to produce fresh takes on our project to further spread our public image.

Security Engineer: Eddie Wen

Designs, implements, and monitors security measures of the general project, such as the application and data information. Along with handling the security architecture and flaws in the application.

Emergency Contact Email:

In case of an emergency, please contact the Escalation Manager, Derrick Luyen, at dluyen@hawaii.edu.

Incident Response Procedures:

The first phase is responding to the breach, or security incidents that are identified or reported. The escalation manager manages this phase by reviewing the security incident. Some of the questions that should be answered are: When and where did the event occur? How much does the event affect the application? Where was the point of entry of the event found? Would the event affect the application service? The escalation manager will work with the clients that reported the issue and other related parties to resolve questions about the issue and to ensure that further damage is not spread. Information on this process is processed by the escalation manager and shared to relevant parties by the public relations representative. Should the incident involve legal matters, the legal representative is consulted for developing a resolution to the incident. The

second phase is assigning tasks to the right response personnel and to resolve the incident. The escalation manager works with the security engineer to design a goal, a plan, and a timeline for resolving the incident by eliminating the source of the breach. This would be removing the malware and patching the vulnerability. The next phase is restoring the application. It would be to ensure that everything is operating normally knowing that the event has been contained. Lastly, the entire team will come together to analyze and discuss the entire breach to make sure any issues can be resolved. New documentation and practices, such as service changes and training, are implemented to prevent further cases of the incident. Notifications for the resolution of the incident will be sent out to clients and related parties.

Final Security Review

Threat Modeling Review:

Going back to our threat model, we have focused on preventing the threats from occurring by following the model and applying the necessary measures to our code in order to provide a secure product and environment for the user. In regards to spoofing, we have successfully been able to defend against logging in with false information and we also have consequences for too many failed login attempts as well by providing a denial of service if this case occurs. We have also prevented injection by checking for errors in the login process to assure that there is nothing suspicious going on. To prevent tampering and elevation of privilege point, we have only allowed users to do certain things in the code whereas admins can restrict what users do as well as delete the ratings they have made or if it is necessary, their account as well. By following the threat model that we established in the beginning, our group was able to come up with ways throughout the project to prevent the threats from having a major effect on the program. This will allow users to feel secure and it allows the app itself to follow the Secure Development Lifecycle guidelines that we were expected to follow while coding the project.

Final Static Analysis Review:

As mentioned before, we have used ESLint as our static analysis tool from the very beginning of the project until the end. ESLint has been a very useful tool for helping us as

developers find code quality and code styling mistakes in our code. It has helped us throughout the project and we have gotten more used to using it as the project has gone along. As for our code, ESLint has not given us any major errors that we need to worry about in it which is a good sign. We have made good strides since the beginning of the project and ESLint has pushed us along the way to be the best developers that we can be by letting us know of our mistakes so that we can correct them in a timely manner.

Final Dynamic Analysis Review:

We have been using Iroh.js as our dynamic analysis tool during the process of working on this project. In the beginning, we did struggle with using this tool as it was our first time taking a look at using it. ESLint was something we had all used before so we did not have as much trouble with that but Iroh.js was a different story. We all tried to understand how to use Iroh.js for our program and eventually figured it out. Overall, we had a good experience using it and it helped us visualize what our code was actually doing. It allowed us to see if our code was doing exactly what we wanted it to do. If it was not, we were able to locate the error and fix it to give us our desired result. However, with all that being said, we still think that there is so much more to learn about Iroh.js and about dynamic analysis in general. We have only scratched the surface in this project. In the future, we hope to learn more about the tool and about dynamic analysis as it is something that seems very helpful as a developer.

FSR Overall Review:

After running the program as a whole as if we were an end user, our team determined that our project received a grade of Passed FSR. We have concluded that our program has successfully met the SDL requirements. With the help of all the tools we have used along the way, including but not limited to Iroh.js, ESLint, our threat model, all of our approved tools, and etc, we have successfully built a project that is able to abide by the guidelines of the SDL requirements. We have used all the tools mentioned above in the process to correct our code and make sure all functionality works as it should in order to meet all of the requirements given to us by the Secure Development Lifecycle.

Certified Release & Archive Report

The link to the release version of our program can be found here.

Summary of features (v1.0):

- Ability to create an account to do the following:
    - View ratings and reviews of the latest movies
    - Add ratings and reviews for the latest movies
- Admin feature
    - Allows admin to have all features of a regular end user
    - Allows admin to delete ratings
    - Also allows admin to delete accounts of certain users

Future development plans:

- Add in an average rating feature that allows the user to see the average rating of a specific movie
- Add in a database for movies so that movies can be added in regularly by admin

Installation:

1. Install meteor, which can be found in the Approved Tools section of our report.
2. Download the latest release of the program here.
3. Install meteor libraries using the command “meteor npm install”, within the app directory of the repo.
4. Also make sure to install Sweet Alert using this command: “npm install sweetalert --save”
5. Start the program using the command “meteor npm run start”.
6. You can now access Flick Critik at “http://localhost:3000/”.