Sentry Requirements and Design

A. INTRODUCTION

1. Team Name: Down To Jam
2. Team Members:
    - Trevin Tejero
    - Dylan Ubongen
    - Jan Iverson Eligio
3. Application Title: Sentry
4. Description: A web application which is used to store and manage passwords that a user utilizes for various online accounts and security features.
5. Functional Requirements Specification:
    - Users are able to create an account from which they are able to store their passwords from various accounts
    - If users need a random password then the application will generate one for the user
    - Passwords are only seen by the user (with correct credentials)
    - Admins are only able to manage user's information (name and email)
6. Type of program: Web Application
7. Development Tools:
    - Development language: JavaScript, HTML5/CSS
    - IDE: IntelliJ
    - Text Editors: Visual Studio (VS) Code
    - Version Control System: GitHub

B. REQUIREMENTS

1. Security and Privacy Requirements:

- Any type of web traffic on this site and through the web server will be required to use HTTPS to ensure encrypted web communication between the web server and the client.
- All user information will be stored in a database and encrypted.
  - User sign-in information (information that is used to identify a user) will be the only information that can and will be accessed or modified by administrators.
  - All other user information (passwords, etc) will be stored in the database and can only be accessed by the user the information belongs to.
- User sign-in passwords must be hashed/salted before stored in the database to secure passwords in the case of a breach.
  - Passwords will be required to meet or exceed a standard as defined: at least 10 characters total, at least 1 uppercase letter, at least 1 special character, and at least 1 number.
- Sensitive information displayed on the client will be obfuscated unless the user so chooses to reveal the information.
- User information displayed to administrators will be kept to a minimum to prevent displaying excessive amounts of identifiable information.
  - Only in cases where more information is needed will there be allowed more user information to be displayed.
- Security flaws/issues will be tracked through the use of GitHub Issues feature on GitHub.
  - Developers will be able to track, be assigned to, and fix any security issues that are submitted.
  - When a security issue is fixed, it will be noted in the individual commit which was submitted to fix the issue.

2. Quality Gates (or Bug Bars):
   - **Privacy**

- Users
  1. Critical
     a. Lack of User Controls
        i. User information will be stored in a database and not on the person's physical computer
     b. Use of Cookies
        i. Information that is stored in a cookie will not be encrypted.
     c. Insufficient legal controls
        i. Any transmission of data to a third party that is not under a contract with the application.
     d. Information Integrity
        i. The information stored in the database if the user loses connection to the application
  2. Important
     a. Lack of User Controls
        i. User are unable to sign-in or cannot access their account
     b. Information Integrity
        i. User's information is removed from the database
  3. Moderate
     a. Lack of User Controls
        i. Users do not have access to the database, therefore, not able to access information
  4. Low
     a. Information Display
        i. Sensitive information is displayed in plain text (without user's consent)

        - Administrators
            1. Critical
                a. Lack of Administrative Controls
                    i. Administrators have access to the user's sensitive information (without user's consent)
            2. Important
                a. Lack of Administrative Controls
                    i. Lack of method for accidental collection of user information
    - **Security**
        - Critical
            1. Privilege
                a. Users obtain administrator controls
            2. Database
                a. A user or attacker is able to delete or modify another user's information that is stored in the database
        - Important
            1. Denial of Service
                a. The functionality of the website is affected by a Denial of Service attack
            2. Spoofing
                a. A user or attacker masquerades themselves (intended or unintended) as another user in order to access their information

3. <u>Risk Assessment Plan for Security and Privacy:</u>
    - Security
        - Identification:
            1. The list of critical assets of the web application has yet to be determined. Upon future determination, a list of all sensitive

information that is stored, transmitted, or created will be collected and a risk profile for each will be created.

    - Assessment:
        1. Upon identification of security risk, an assessment will be made for each individual risk.
    - Mitigation:
        1. Upon identification of security risk, a mitigation strategy will be made to enforce security controls for all risks.
    - Prevention:
        1. Database security assessments must be made at appropriate time intervals. A list of software security issues will be compiled and relayed to administrators and developers on a month-by-month basis.
- Privacy
    - Identification:
        1. Privacy Impact Rating:
            a. The PIR is representative of the level of risk that the application assumes in terms of privacy.
            b. Key: P1 - High, P2 - Medium, P3 - Low
            c. P1 Behaviors:
                i. Continuous user monitoring
                ii. Installation of 3rd party software or undesignated software
                iii. Stores user device information or any other identifiable information
            d. P2 Behaviors:
                i. Transfering of anonymous information
            e. P3 Behaviors:
                i. None of the above

- Assessment:
    1. Upon identification of security risk, creation of a risk assessment will be created and the following scenarios will be included:
        a. Description of identifiable information stored or transferred
        b. Description of reasons the identifiable information is required for usage of the software
        c. Description of notice and consent experiences
        d. Description of how to prevent risk as defined in the assessment
- Mitigation:
    1. Determination of a risk and creation of a viable assessment. Following through with the recommendations outlined in the assessment is required.
- Prevention:
    1. Software privacy assessments must be made at appropriate time intervals. A list of possible software privacy issues will be compiled and relayed to administrators and developers on a month-by-month basis.

C. DESIGN

1. Design Requirements:
    - All user's information will be stored on a database
    - User's data and Administrator's data will be stored into separate databases
    - User's passwords will be encrypted/hashed when stored to the database and will not be in plain text form when entering/signing in
    - User's information can only be accessed by the user and the administrator (with consent)

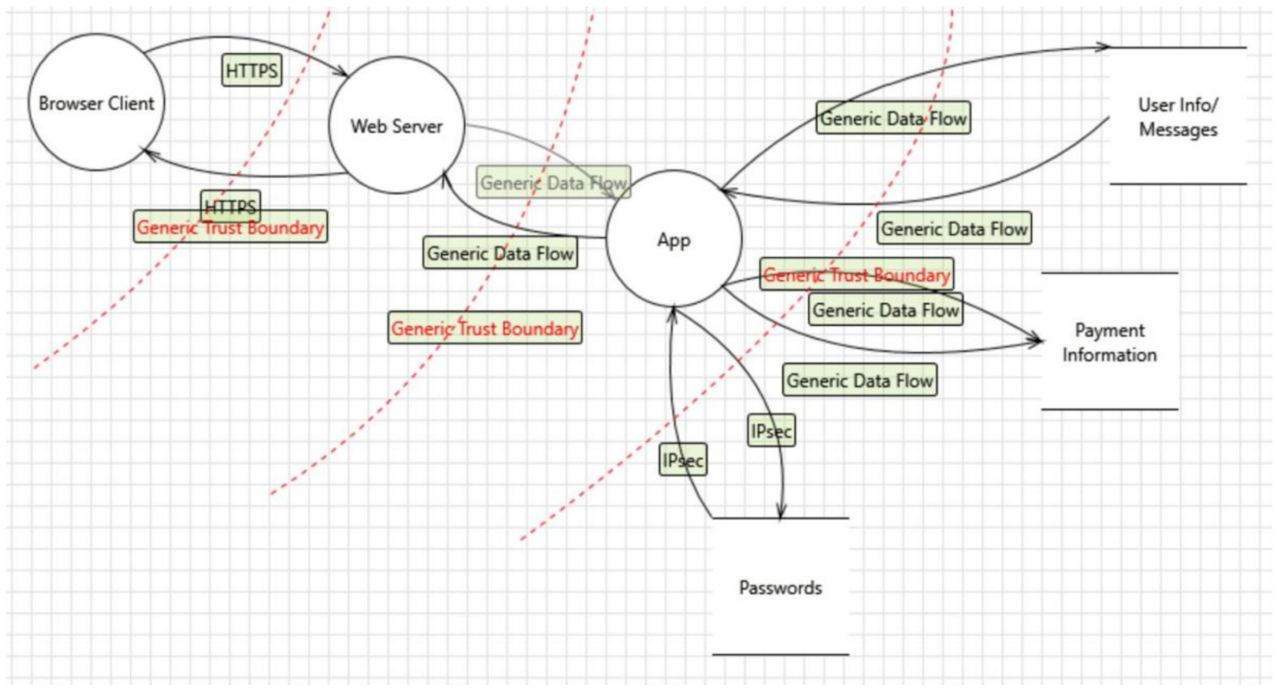
- Security flaws/issues will be tracked through the use of GitHub Issues feature on GitHub.
- Each version of the application will go through extensive internal testing before being released to the public.
- Password generator will adhere to acceptable password standards.
- Upon deletion of the user’s account, all user data and stored information will be deleted within 7 days of account termination.
- Use of any third-party applications or end-points must complete a security assessment before implementation and release to the public.
- Sensitive information of any kind will not be stored on any physical user device or on any third-party applications without user consent.
- Management or listings of information would contain the Site (location where the password is being used) and Password

2. <u>Attack Surface Analysis and Reduction</u>:
    - Privilege levels
        - Unregistered Users:
            1. U-Users will be be able to have access to the landing page, home page, about page, registration page, and password generation tool exclusively.
        - Registered Users:
            1. R-Users will have access to all of the non-administration pages, possess the ability to view their user information, and create, view, and store accounts and passwords in the client.
        - Administrators:
            1. Admins will have full access to the application, possess the ability to view and modify user information, and remove users.
    - Possible vulnerabilities (examples taken from LastPass):
        - CVE-2020-6758

1. A cross-site scripting (XSS) vulnerability in Option/optionsAll.php in Rasilient PixelStor 5000 K:4.0.1580-20150629 (KDI Version) allows remote attackers to inject arbitrary web script or HTML via the ContentFrame parameter.

- CVE-2020-6377
  1. Use after free in audio in Google Chrome prior to 79.0.3945.117 allowed a remote attacker to potentially exploit heap corruption via a crafted HTML page.
- CVE-2019-8349
  1. Multiple cross-site scripting (XSS) vulnerabilities in HTMLy 2.7.4 allow remote attackers to inject arbitrary web script or HTML via the (1) destination parameter to delete feature; the (2) destination parameter to edit feature; (3) content parameter in the profile feature.
- CVE-2019-8331
  1. In Bootstrap before 3.4.1 and 4.3.x before 4.3.1, XSS is possible in the tooltip or popover data-template attribute.
- List of other possible vulnerabilities

3. Threat Modeling:



D. IMPLEMENTATION

1. Approved Tools

Tools	Minimum Required Version and Options	Recommended Version and Options	Comments
Source Code Management System	Git/GitHub	Most Recent	Sentry
JavaScript IDE	Visual Studio Code 2020 / IntelliJ IDEA 2020 Ultimate	Most Recent	Visual Studio Code is a free to use text editor IntelliJ IDEA 2020 Ultimate is offered to students for free
Static Analysis	ESLint Version 6.0.1	Most Recent	Syntax checker
Node.js	Version 13.8.0	Most Recent	Server's side Javascript code execution

Database	MongoDB	Most Recent	Cloud database
npm	Version 6.13.7	Most Recent	Package manager for Node.js
React		Most Recent	User Interface

2. Deprecated/Unsafe Functions

- React:
    - Component mounting: componentWillMount
        - Alt: componentDidMount
    - Component receiving: componentWillReceiveProps
        - Alt: getDerivedStateFromProps
    - Updating components: componentWillUpdate
        - Alt: getSnapshotBeforeUpdate
        - Alt: componentDidUpdate
    - HTML center text tag: <center>
        - Alt: text-align
    - HTML attr to specify a background image: background
        - Alt: background-image
- MongoDB:
    - Unsafe filtering with the $where operator: NoSQL injection
        - Alt: Must validate user input by disallowing injection characters and limit input length.
- JavaScript:
    - Function expression closures
        - Alt: Use regular functions or arrow functions
    - For statements: for each...in/for...in
        - Alt: Use for...of instead
    - Date: toLocaleFormat
        - Alt: None

        - Date: toGMTString
            - Alt: toISOString

3. <u>Static Analysis: ESLint</u>
    - ESLint is a static code analysis tool used to find and fix code quality issues that a developer may have when creating a JavaScript-based web application. ESLint has a plug-in/extension for the IntelliJ IDE (an approved development tool). Through IntelliJ developers are able to load up the extension and use ESLint's rules while coding. This allows for ease of use and an automated feedback system for warnings or errors. VSCode (another approved development tool) also allows for a similar automated feedback system like IntelliJ. Another option is to run ESLint manually in the command line. While not optimal, this still gives the developers useful information as to where a warning/error occurred and what it means.
    - ESLint features:
        - Customizable linting rules
        - Simple rule pattern for ease-of-use
        - Dynamically loadable rules
        - JSX (React) support
        - Fix suggestions and automated fix application
    - Experience:
        - Most of us have used ESLint in the past with classes like ICS 314. We noted that setting up the automatic linting tools in IDEs such as IntelliJ could be problematic if issues were found and that sometimes it could be wonky. Once passed the setup process, using ESLint was simple and easy. If something went wrong in our code, ESLint would give a warning via a squiggly line and a warning symbol on our IDEs/text-editors. It also gave us suggestions on how to fix issues like line-overflows (too many

characters per line) and deprecated JS functions. Overall, the tool was convenient and easy to use once past the setup process.

<u>**E. VERIFICATION**</u>

1. Dynamic Analysis
    a. For our application we decided to use the dynamic analysis tool called iroh.js and its usages are type checking, intercept or manipulate code or data while running, code quality and test cases. Also, this tool works by recording everything that is occurring within your code without altering the original program. As of 3/15/2020 we have not used the tool yet as we are still working out some of the mechanisms of our back-end development.
2. Attack Surface Review
    a. As of 3/15/2020, there have not been any patches, changes, updates, or new reported vulnerabilities for any tools in the approved tools section above (Section D-1). We will continue to monitor those tools and will list them in this report in the event a vulnerability is found.

<u>**F. VERIFICATION Part 2**</u>

1. Fuzz Testing
    a. Input Validation
        i. Our application has three account forms, two forms are for authentication purposes and one form is for account creation. Users have two forms, account creation and logging in. In our account creation, the password requirements will consist of at least 1 uppercase letter, at least 1 lowercase letter, at least 1 special character, at least 1 number, and at least 15 characters total and the maximum amount of characters will be at most 30 characters. In order to test this, we used Postman to send HTTP requests to our server. We then tested our API routes with passwords that did not meet all of the requirements. With this in mind, we have

confirmed that our back-end form validation for registering users worked correctly. The last form was for logging in as an admin and because the admin accounts are premade we do not need to validate the creation of the account.

b. Testing Application with Robustness
    i. For testing our application, we used the node module, gremlins.js, and this is used to check the robustness of our application. What the module does is that it simulates (randomly) user actions. A horde of users or “gremlins” is sent to our app and it will then proceed to click anywhere within the window, enter random data, or commit random actions that would not be expected. The goal of gremlins.js was to trigger any errors that would make the application fail. In all of our attempts, there were not enough errors generated to crash the application..
c. Security Misconfiguration
    i. This was more of a security concern as our ESLint module was printing out misconfigurations in our front-end configuration to the web console. At the time of testing there was nothing sensitive that appeared, however, there still exists the potential for sensitive information about how the site works or if there are any exploitable misconfigurations to be displayed through this console. To mitigate this, we could disable the errors from printing to the console using the ESLint rules to disable it (no-console: “off”).

2. Static Analysis
    a. As mentioned in section D-3, our team is using ESLint for the static analysis of our code. After compiling our code and running ESLint, we found that we had a lot of cleaning up to do of our code. The main problem we had were a lot of unused variables from removing and moving things around. In our HTML files we had a few problems with the

formatting that needed to be cleaned up. Also, we noted that we should be using more arrow functions instead of the old JS function formatting.

   b. In one case, we found that one of the images we were trying to import were missing from our files or we just forgot to import it. This was a quick fix, but we would have passed over it had we not used ESLint.
   c. One person on our team noticed that the static analysis tool sometimes did not work out as expected. They ran into issues where they attempted to fix issues relating to variables that could not be resolved. However, whatever imports that were needed for the variables were in place and it was an error with ESLint.
   d. The static analysis tool, ESLint, had some problems, but in the end it did help us clean up our code for the better. It also made it a lot easier to read.

3. Dynamic Analysis
   a. Since adding in the dynamic analysis tool, iroh.js, we only recently ran it to find that there were no errors or vulnerabilities that needed to be corrected at the time of writing this report.
   b. We also used iroh.js to evaluate our application runtime performance having to do with load times and function runtimes. All the load times or runtimes were within an acceptable range and we did not feel as if we needed to fix any performance issues.
   c. Iroh.js comes with the ability to run step-by-step through any JS functions and analyze their performance and see what is going on under the hood. We used this ability to run through some of our more complicated functions and have a look. We noted that this would be a very useful tool for debugging and analyzing the tiny vulnerabilities in our app had we found one.

d. Overall, we found that the dynamic analysis tool would be useful in a work-related setting and for a bigger project. However, it was still valuable to know that there were no known vulnerabilities within our application.

G. RELEASE

1. Incident Response Plan
    a. Privacy Escalation Team
        i. Escalation Manager (Dylan Ubongen) - responsible for delegating responsibility and tasks across the privacy escalation team to ensure the completion of the incident response process.
        ii. Legal Representative (Trevin Tejero) - responsible for handling legal concerns throughout the incident response process by working with a legal team to determine the best courses of action.
        iii. Public Relations (Dylan Ubongen) - responsible for handling public relation concerns throughout the incident response process through consistent and timely response on public social media and press events.
        iv. Security Engineer (Trevin Tejero) - responsible for bridging the gap between the technical and legal aspects of incident response. Also, responsible for documenting processes and procedures throughout the incident response process to ensure similar incidents do not recur.
    b. Emergency Contact
        i. In the case of an emergency (your data has been breached, your account has been compromised, etc.), you may contact us via email at help@sentrysecure.com. A representative will respond to you within 1-2 business days. If your emergency is of a law-breaking or of a sensitive nature, please contact your local law enforcement agency.
    c. Incident Response Procedures

i. It is crucial in our line of work that uphold the security of our application and any bugs that arise from our updates must follow our incident response procedures. This is to ensure that our response to these incidents are done efficiently and mitigates all (if not most) of the damage that was inflicted prior to this. It is also a way for us to prepare, as these procedures could help us with finding the source of any future incident that might occur.
    1. Establish Team Responsibilities
        a. When addressing incidents, it is necessary to have a foundation so that we are able to resolve issues as efficiently as possible.
        b. The primary focus of this step is to determine everyone's role in this procedure as it is crucial for the efficiency rate of the team and response rate for the user's feedback.
        c. Here at Sentry we believe all of this is necessary, that is why we have already determined our roles that we are participating in.
    2. Identify Timeline Expectations and Goals
        a. The primary focus of this step is to determine when each goal is expected to be completed or met. Such goals are made from questions like these:
            i. Who is being affected?
                1. By determining the demographic that is being affected, we are able to resolve this issue in response to satisfy those customers
            ii. Who shall we notify first?

            1. We must determine the scale of this attack or incident so that we may prepare for releasing any public relation statements.

        iii. What is the cause of this incident?

            1. First, we must indicate the source of this issue.
            2. Then, in response we must shut anything down that would have been remotely affected by this issue.
            3. Last, release any updates or patches to the application so that our customers feel comfortable with using it once again.

        iv. Should we take legal repercussions?

3. Document Process
    a. The documentation of these incidents will help us respond quicker to any similar issues that a customer may have regarding the application. This helps us understand the situation that the customer is in and it will provide us a way to help them through this issue
4. Evaluation
    a. After this incident is resolved, it is up to the incident response team to decide the next course of actions so that we do not have a recurrence of previous incidents in our newest patch.

2. Final Security Review
    a. Prior to release, our team conducted a final security review of our software, Sentry. The security team followed standards set by Microsoft's

Final Security Review and Privacy Review guidelines. Our FSR process included several steps designed to highlight any security flaws in our software design.

b. FSR Grade: **Passed FSR (with exceptions)**

c. Reasoning

   i. Threat Model

      1. Sentry, the application, met all requirements outlined in our threat model, with some exceptions:
      2. Our MongoDB database connection with the local server implementation is somewhat insecure due to hardware and implementation limits. In a more company-like rollout, we would ideally have a dedicated MongoDB server that can store live information without having to load it into Meteor on startup. Sensitive information like passwords and user information is currently reset on startup and is stored in a XML file.
      3. In our current implementation, some user keys and passwords are stored in the code to make startup easier on the developers and testers. If we were rolling this out into the public and given more resources, we would store these things securely in the DB or server.

   ii. Static Analysis

      1. Sentry meets all requirements as outlined in the Static Analysis section of this document. As of release, there are no potentially insecure configurations, syntax, or settings present in our product. Our SA tool, ESLint, found no problems prior to release.

   iii. Dynamic Analysis

1. Sentry meets all requirements as outlined in the Dynamic Analysis section of this document. As of release, there were no known insecure runtime events and all load times were in a reasonable range. Our DA tool, iroh.js, found no problems prior to release.

iv. Quality Gates/Bug Bars

1. Sentry meets all requirements as outlined in the Quality Gates/Bug Bars section. Some parts of the product will not be available for release and have been omitted from the FSR. Please see our Future Development Plans in the later sections.

3. Certified Release and Archive Report
   a. Final Release of Sentry (Version 1.0)
      i. Github:
      ii. Summary
         1. Features
            a. User Accounts
               i. Users are able to create their own accounts and start storing their account passwords without any hassle. However, the use of this application requires the user to follow the Terms and service of the application. If not followed, their account will result in an immediate removal from the app.
            b. Secure storage
               i. We understand that our application will be used to store sensitive data, that is why it is our main objective to keep that data safe by encrypting it in a string within our database so

they are secured and not susceptible to hacking.

   c. Terms of Service and Privacy Policy
      i. Any information regarding our Terms and Service will be noted on our application. But a brief summary: our terms and service restricts taking any action in illegal activity.

2. Future Development
   a. HTTPS implementation
      i. In an application, the standards as of today is to run the app over HTTPS and we would like to implement that in our app so that the traffic to and from the server would be encrypted and secured.
   b. Search Functionality
      i. We understand that with our application that finding a password for a specific account would be less efficient if the user had to manually look for their password. That is why we believe if this application was actually used, the user would have multiple passwords that are stored and we would want to make it easier for the user by implementing a search bar for them.
   c. Update the Security on our Storage
      i. Although our application does have sufficient security on our database, we would like to further improve on that security so that we increase the integrity of our application.
3. Program Usage

a. Clone the Sentry repository from Github
    i. Use Github Desktop or the git command line tool to clone the repo.
b. 2. Install NodeJS
    i. The latest version of NodeJS needs to be installed in order for this application to run.
        1. Find the latest version of NodeJS: https://nodejs.org/en/
c. Install Meteor
    i. The latest version of Meteor is required to run the application.
        1. The latest version of Meteor: https://www.meteor.com/install
d. Navigate to the Sentry Application folder
    i. On your computer, navigate to the location of the cloned Sentry repository in your command line application
    ii. CD into the "app" folder found at ../sentry/app/
e. Install NodeJS application packages
    i. Ensure NodeJS has been successfully installed
    ii. In the command line, type npm install in the Sentry "app" folder
f. Run the Meteor application
    i. In the command line, type meteor npm run start
g. Open the application in your browser
    i. In your browser of choice, navigate to http://localhost:3000/

ii. You will be met with the Sentry Home Page