Secure Development Life Cycle - Requirements and Design Report

A. INTRODUCTION

Team Name: PBJ

Team Members: Sang Nguyen, Shawn Anthony, Jon Hsu

Application Name: Peanut Gallery

"Peanut Gallery" is a web-based application that allows users to create and submit surveys. New users will be encouraged to go through an account creation process in which they submit their full name and email address and create a username and password. User information and login credentials need to be stored and properly secured in a database system to uphold their privacy.

The application will contain multiple surveys that a user can select in order to direct them to that survey. Each survey will poll, question, or prompt users to scroll through and complete their responses on a given subject matter. The results from each survey will be added and updated into a collective database that authorized users will be able to view. User roles will need to be defined and implemented in order to manage access control.

With regard to development tools, this application will include the use of HTML, CSS, and JavaScript alongside libraries and frameworks like React and Semantic UI. Our IDE of choice will be IntelliJ IDEA and we'll be using GitHub as a code repository and means of organized version control. We also hope to find a service that will host our website publicly.

B. REQUIREMENTS

1. Security and Privacy Requirements:

To match the level of sensitive information, our program should incorporate security measures to ensure that the customer's information remains secure. To do this, we have to examine the areas where a potential breach/attack can happen. Initially, the customer's account information should be encrypted so that if a data breach occurs, the information that is released will not be able to be used without the decryption key. From there, the encryption/decryption method should be properly hidden so that the account information cannot be decrypted. The survey results must also be securely transferred to a protected database that will only allow authorized users to access it. This system would be managed with user roles that have access to different levels of access so that users should only be able to view things that they should view. To clarify, basic users should be able to view the sites, surveys, and their account information. A higher level user, which could be an administrator, would be able to view the survey results and change survey questions in addition to the previous features of a basic user and the highest level would also have access to editing the site. This means that we should make sure that administrator access credentials are securely hidden and unable to be guessed easily. As for privacy, the customer should be aware of what kind of information we are asking from them and how we are going to use their information. This should be presented in an easy to read format, such as a terms and conditions box, with a way to confirm that they have read and agreed to giving out their information. The survey results should not be able to be traced back to any individual and should be a part of a collective total to avoid singling out any specific input. Our site should be consistently monitored for the sake of detecting any discrepancies or potential attacks, in which case it should be shared with the customers. We are looking at using GitHub as our main method of keeping track of potential flaws or vulnerabilities that may arise during the creation process since we can create tasks that can be looked at and organized appropriately.

2. Quality Gates (or Bug Bars):

   Our project involves handling user credentials such as login and password, and also identifying information such as name, email or address, as well as information recorded from the user responses to survey questions. Many of our potential issues include the way data is stored or maintained, as well as more common vulnerabilities that can affect web applications.

   Privacy Bug Bar

- Critical
  - Lack of data protection
    - PII stored without a secure way to update or correct PII
  - Privacy Features
    - PII not hashed or encrypted on server
- Important
  - Lack of user controls
    - PII stored on servers that cannot be revoked upon submission
  - Lack of child protection
    - Age not collected for survey
  - Lack of notice and consent
    - Transfer of PII between user and survey distributor
- Moderate

    - Lack of internal data management and control
        - Data stored without retention policy
    - Tampering
        - Data interception between client and servers

Security Bug Bar

- Critical
    - Security Features
        - Login credentials not hashed or encrypted
- Important
    - Denial of Service
        - Web page is susceptible to DoS attacks
    - Elevation of privilege
        - Gaining access to admin account could allow malicious behavior to occur
    - Tampering
        - Modification of unencrypted data
- Moderate
    - Spoofing
        - Similar UI used to capture user information unknowingly

3. <u>Risk Assessment Plan for Security and Privacy:</u>

In order to assess our risk, we need to take a look at the types of information we're gathering from our users and weigh the risks that come with that sort of acquisition. Our database gathers personally identifiable information (PII) such as names, email addresses, usernames, and passwords. Protecting and maintaining PII with an encryption system lowers the risk of unauthorized access. The database also anonymously gathers user input data (ie. via survey results) and gives access to that data to privileged users. This is a privacy risk and it demands that a transparent set of systems be put in place to guarantee the privacy and consent of our users.

A terms of service and disclosure agreement at account creation could be one such system. A checkbox could be placed under the terms and conditions that allows us to receive a user's consent. While survey data is stored, it will not be linked to any personal information and will therefore not be traceable to any particular individual. Anonymously, these results will only be viewable by authorized users. Results may only be used by the correct parties with facilitating the improvement of quality and service. In the event of a data breach attack, its existence and the nature of what information was compromised must be communicated to our users.

Updates to our implementation and feature decisions based on our threat models are our best method of assessing and correcting our security and privacy flaws. Threat models should examine the types of information being stored on our databases with a particular focus on login implementation and parts of the database shared across users of varying authorizations.

<u>C. DESIGN</u>

1. <u>Design Requirements:</u>

The point of this project is to allow users to answer surveys. The input types on the survey fields should include “radio”, “checkbox”, “dropdown”, and “free response”. Free responses in particular will require an adequate level of input sanitation. Examples like character count limits, character type limits, and contextual data filtering come to mind. More input features may be added on at our discretion.

Users should be able to create survey prompts, answer survey questions, and view survey results. Users that have created surveys should be able to view the results of their own surveys. They should be able to see how a specific individual user responded to all of the questions on the survey as well as how all users responded to a specific question in an organized aggregate. Users filling out a survey should be able to see which surveys they are able to answer.

An account should be required in order to use the service safely and distinguish user privileges. This would require a signup and signin feature as well as an information disclosure agreement. Passwords should be hashed and properly stored, rather than stored in plaintext. To prevent brute force attacks on accounts, IP addresses should be rate limited on how many logins they may attempt per minute. To reduce the potential for phishing attacks like spoofing, users should be reminded to check that they are on the correct website before attempting to login. Two-factor authentication is recommended as a means of account hijacking prevention.

2. Attack Surface Analysis and Reduction:

Three levels of privilege will be applied: Developer, administrator, and clientele. Developers have functional edit privileges to the source code, database, version control, front-end, and back-end interfaces. Users are clientele by default but gain administrator privilege only on surveys that they have created. Administrators have read and delete access to the results of the surveys with which they have created. Clientele will only be able to submit and fill out the forms and fields sections of the surveys that they have been granted access to by an administrator.

This program features five elements of entry/exit that qualify as attack surfaces. Developers manage the functional properties of all five: the survey forms, database (and any necessary data cryptography), HTTP headers/cookies, access control, and version control. All users use HTTP URLs and have access to login features. Clients update the database by filling out forms and fields. Administrators create, read, and delete certain elements of the database by creating survey prompts, reviewing clientele answers, and deleting their surveys. These may be subject to change based on added features. For example, pending implementation, we may allow administrators to choose whether or not they wish to release their survey results to their clientele post-submission.

These surfaces inherently open the system to certain vulnerabilities. All login interfaces share a weakness to weak or stolen login credentials. SQL injections can wreak havoc on any database dependent platform. HTTP header attacks and version control abuse are dangers to our access privilege policies. Poor encryption is vulnerable to data breach. DDoS attacks threaten the operation of the service.

An abridged version of this analysis is listed below:

Goal: Determine privilege levels and compile a list of software vulnerabilities (some from other similar programs).

Privilege levels:

- Developer
- Administrator
- Clientele

Entry and exit sources:

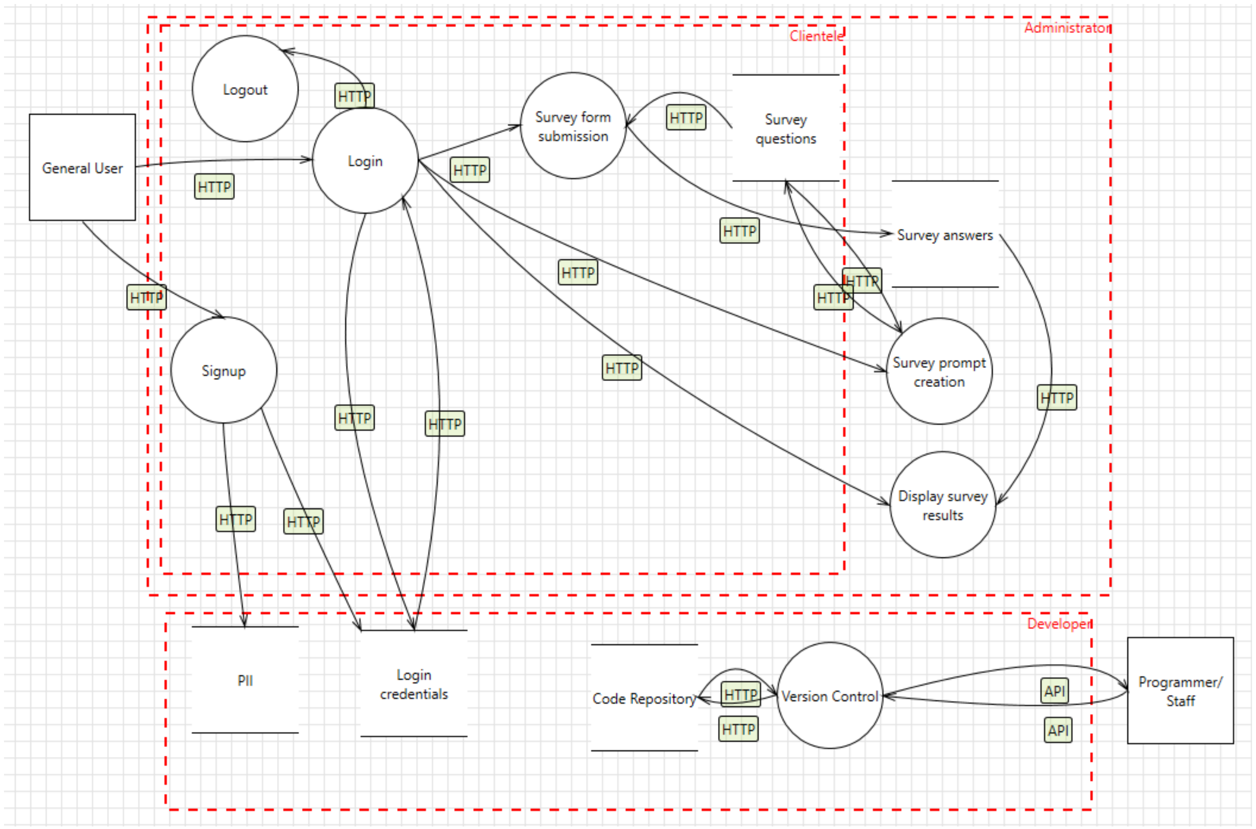
- Forms and fields
- Database
    - Create, read, update, delete
    - Data cryptography
- HTTP headers and cookies
- Access control
    - User authentication, authorization, and interface
- Version control
    - Backup code/data
    - Old or unused code/features

Attack vectors:

- Distributed Denial of Service (DDoS)
- Weak or stolen login credentials

    - Brute force attack
- SQL injection
- HTTP host header attack
- Poor encryption
- Version control abuse

3. <u>Threat Modeling:</u>



Spoofing

- Impersonating a programmer/staff member could be a means of obtaining login credentials or PII.

Tampering

- Version control abuse compromises the integrity and functionality of the service.
- Clientele and administrators often submit information that is inserted into and read from shared database elements, making SQL injection a threat to the integrity of those elements.

Information disclosure

- If non-administrator users gain access to survey results when they are not meant to, this would constitute a disclosure breach.
  - Poor encryption practice could risk this type of breach.

Denial of service

- Storage
  - Data stores are at risk of malicious submissions of large freewrite survey responses and large quantities of survey or user account creations.
- Because this is a web application, it is vulnerable to an excess of internet traffic.

Elevation of Privilege

- HTTP host header attack
  - Nearly every vector of data flow traverses via HTTP. The right HTTP attack could grant users unprivileged access to web features.
- Weak access control in the form of weak/stolen passwords and unlimited login attempts allow for unprivileged access to PII and survey data.

D. IMPLEMENTATION

1. **Approved tools:**

This application approves the use of IntelliJ IDEA Ultimate Edition as the main compiler for HTML, CSS, and Javascript. At the time of writing, the current latest version of IntelliJ, which is version 2020.3.2, will be used. The development team has prior experience with building and launching applications using Meteor and will be using it as a Javascript platform to build the web application and test its basic functionality. The database platform-of-choice will be MongoDB to keep the data stored in a secure and accessible area. The source code will be kept in a GitHub repository and updated using GitHub Desktop to push changes made using developer systems. The repository can be found at: https://github.com/Peanut-Gallery/peanutgallery

2. **Deprecated/Unsafe Functions** :

Unsafe Property: innerHTML

Alternative Recommendation: dangerouslySetInnerHTML using dompurify sanitization library

Comments: HTML from code is dangerous because it's vulnerable to XSS attack. Using dompurify to sanitize it is necessary when rendering HTML.

Deprecated Function: componentWillMount()

Alternative Recommendation: this.setstate

Comments: Unsafe option is deprecated to UNSAFE_componentWillMount and will be removed.

Deprecated Functions: componentDidUpdate() and componentWillReceiveProps()

Alternative Recommendation: Synchronize component state to the props without calling this.setstate.

Comments: Unsafe option is deprecated to UNSAFE and will be removed.

Deprecated Functions: componentDidMount(), shouldComponentUpdate(), componentWillUnmount()

Alternative Recommendation: Use alternative React "side effects" or AJAX calls.

Comments: Unsafe option is deprecated to UNSAFE and will be removed.

3. **Static Analysis:**

The static analysis tool chosen for this project is ESLint as a plugin for Intellij IDEA. The development team has prior experience using ESLint and its instant feedback nature is effective and helpful to the code writing process. ESLint is a linting tool for identifying problematic patterns in Javascript code. It is highly configurable and customized rules can be created to better fit the needs of each individual project. It gives immediate feedback upon encountering problematic code that violates any of the currently active rules in place. Each rule created in ESLint is able to be turned on or off separately such that developers can control precisely what is checked for.

After using ESLint several times, due to the fact that it is an always on static analysis tool, it is hard to pinpoint any specific successes or difficulties that were encountered within the tool while it checked the source code for any problematic sections. While it picked up many things that could be changed, the nature of changing them individually while coding made the successes less memorable than they would have been had we checked at compile time and then fixed multiple problematic sections of code at once. The end result however is the same, the code that we write will conform to the standards of the rules we enable which will allow better quality control and consistency across the project. One difficulty that we may need to handle is figuring out which of the rules we would like to use and evaluating periodically to see if there are additional rules that we should turn on or write to fit better with what we need. Another difficulty came during configuration, by default ESLint has all rules disabled, and with the vast number of different rules, it can be hard to pick out what exactly should be turned on. Over time we should be able to better understand which rules are beneficial to turn on, and which ones might be better to turn off if they aren't useful for what we are doing.

E. VERIFICATION

1. Dynamic Analysis:

The dynamic analysis tool of choice for this application will be Google Chrome's developer tools. Conveniently, it seems that every browser comes equipped with some set of developer tools like dynamic program analyzers. For this web application, we use Google Chrome's debugger to create a runtime autopsy on the applications network traffic. Chrome's debugger is surprisingly configurable in that it allows our developers to analyze our code via breakpoints. With respect to this particular project, what is particularly useful is that the debugger allows us to set these according to form submissions, mouse clicks, XML/HTTP requests, and animations. These breakpoints more or less identify and make it easy to understand the specific lines of codes that could be considered attack vectors.

The biggest difficulty we encountered in using a dynamic analysis tool was the debugging process. The tool did a great job of identifying weaknesses in our code in which the client could change data that they should not have had edit privileges to, however understanding how to patch those vulnerabilities was a difficult task. However, regular analysis has been a great learning experience for the team. We can understand that the server should not be accepting data from the client without validating it. Code is not necessarily secure just because it shows itself to be the case at compile time.

2. Attack Surface Review:

During the period between our last write-up for Assignment 2 and this current assignment, there have been multiple reported updates to our tools. Specifically, ESLint was updated multiple times and is currently on version 7.23.0 as of March 26th. These development updates can be found at https://eslint.org/blog/. In addition, our IntelliJ IDE released a new version on March 15th, with version 2020.3.3. This is found at https://confluence.jetbrains.com/display/IDEADEV/IntelliJ+IDEA+2020.3.3+(203.7717.56+build)+Release+Notes. With regards to Meteor, there was a recent security update with Node.js on February 24th, where some vulnerabilities were addressed. These included vulnerabilities to a denial of service attack, which caused a leak to file descriptors leading to excessive memory usage. In addition, a DNS rebinding vulnerability was addressed for affected Node.js versions. These updates were found at https://nodejs.org/en/blog/vulnerability/february-2021-security-releases/. Finally, there was an update right after our write-up for Assignment 2 with MongoDB that now supports Client-side Field-level Encryption. From the update, it states that the change "allows developers to selectively encrypt individual fields of a document using the MongoDB drivers (and now with mongosh as well) on the client before it is sent to the server. This keeps data encrypted (but still queryable) while it is in-use in database memory, and protects it from the providers hosting the database, as well as from any user that has direct access to the database." This is a big security change that could make it easier for our application to secure client data from unauthorized access. This update can be found at https://www.mongodb.com/blog/post/new-mongodb-shell-now-supports-clientside-fieldlevel-encryption.

4/11/21 VERIFICATION - Cont.

Due to the time restraint and the deadline date approaching, we are going to revise the concept of our site such that it will mostly involve the basic user role and the development/administrator role. We will also revise the concept of each survey to include predetermined questions/ratings for now but will be changed to customizable questions in the future.

1. Fuzz Testing

______The security testing of this application featured three different methods employed in an effort to break into the current iteration of the program. The first method was a brute force attempt into the sign-in page. Given enough time, a brute force method theoretically would have a 100% success rate of gaining access to either a basic user account or an administrator. However, in practice, it is not unlikely to succeed because the attacker would need to guess the exact email initially, followed by the password. This attack may be more likely to succeed if the attacker had knowledge of a user email beforehand. Since our current system does not have requirements for emails or passwords, this is an easy way for an attacker to hijack accounts that may not have secure passwords. In addition, the current iteration does not detect for high volumes of login attempts making this attack possible. This type of attack could also imply that the application is susceptible to DoS attacks using automated systems that can flood it with continuous requests. To secure this, the development team would have to implement a system that either tries to detect unusual high volumes of failed login requests or limits the number of login attempts to a defined value for each user that tries to sign in to a specific email account. In addition, requirements could be set, demanding users to create strong passwords using common requirements such as one upper/lower case, number, special character, etc.

The second method used was an attempted NoSQL injection attack used against the MongoDB database. Rather than entering in a normal user input to a form field such as "email", a different input that resembles a command in our JSON data collection was used instead. When entered, the database treats the line as a valid query that will execute in the database. An attacker can use this by entering in something like {‘$gt:0’}, which would return all the values in that data collection. This attack has a very high success rate and indicates that the current iteration of the program is susceptible to this kind of attack. To protect against this, a similar input and data sanitation policy to the one suggested with the previous method could be implemented. If it detects something similar to a query command, it should automatically reject the input. In addition, MongoDB has a module that can be used to sanitize user inputs, to prevent any received data that will not harm our application.

Lastly, the team examined a very basic web application attack that anyone could do: exploit the application by simply looking at the page elements using their browser’s inspect element tool. From this feature, we wanted to ensure that any changes made on the user’s side cannot permanently affect the web application on the server’s side and that the user is unable to view any sensitive information that could be left over in the elements. In the execution of this attack, we checked through multiple different sources from our browser but did not find any instance where our JSON files were discoverable. In addition, we did not notice any long-lasting changes when making changes to the code when on the user’s side. This indicates that our data collections and data stored in our JSON files are safe from being viewed on the local side, rendering this attack surface harmless.

2. Static Analysis Review

Regarding static analysis status since the last report, the team has continued using it to check the source code and ensure that the current code is in the proper formatting and is functioning correctly. In addition, the development process has continually used it to sanitize any new code contributed into the project and check for compatibility issues with our current software. After more continued testing, the team noticed that while it takes some time for ESLint to finish its checks, it still remains consistent in its coverage. Overall, it has been incredibly helpful to have featured in the web application and continues to be useful throughout the active development period.

3. Dynamic Review

The development team thought it might be best to add a second dynamic analysis tool that more formally scanned through the application such that it would be compliant with industry standard and regulatory security protocols (particularly NIST, OWASP, ISO, and PCI standards). That being said, the application has added HCL Software's Appscan (formerly known as IBM Appscan) to the dynamic analysis process. Appscan runs a scan on the application URL, detects security issues, and generates a summary report of its findings in a UI that seems to mirror that of a typical antivirus software scan. The advantage to this tool is its ability to detect larger quantities of security issues in the web application. The disadvantage is that it can be even less clear where to move forward in the debugging process to eliminate those issues. When Appscan lists 3 issues with XSS, 1 issue with SQL injection, and another through "Phishing Through URL Redirection", the descriptions of the issues end there. In that respect, it gives more broad clues indicating the vulnerabilities behind the application and its attack surfaces. This contrasts with Chrome's ability to point to more specific areas of code. Even still, it gives the team a better understanding of where the application stands at an industry level and promotes a healthy discussion of what features (and subsequently attack surfaces) should and should not be added or kept in the project.

F. INCIDENT RESPONSE PLAN

List of Contacts

In the event of an emergency, notify and email the Peanut Gallery escalation manager or one of the listed backup points of contact.

Peanut Gallery Escalation Manager - Sang Nguyen - snguyen2@hawaii.edu

Backup - Legal and Public Relations Representative - Jon Hsu - jonhsu@hawaii.edu

Backup - Security Engineer - Shawn Anthony - Shawn8@hawaii.edu

The Computer Incident Response Team (CIRT) divides itself into four key areas of responsibility. Peanut Gallery Escalation Manager Sang Nguyen, will act as the coordinating body responsible for assigning the specific tasks of the incident handling process to the appropriate representations of the team. The escalation manager drives the process from start to finish and in severe cases, all four divisions may be involved in the coordination effort. This includes running qualitative and quantitative risk assessments to determine the validity, source, threat level, scope, prioritization, and scheduling concerns of an escalation. Peanut Gallery Legal and Public Relations Representative Jon Hsu, is responsible for the legal and PR concerns throughout the incident handling process. This includes legal compliances, human resource management, and necessary external communications such as breach notices, public outreach, training, customer service, and help guides. Lastly, Peanut Gallery Security Engineer, Shawn Anthony is responsible for testing and screening security software, scanning and monitoring systems and networks for security intrusions or breaches, product/service changes, encrypting data, and updating documentation.

Peanut Gallery's CIRT team procedures borrow from the Sans Institute division of incident handling into the 6 respective processes: preparation, detection, containment, eradication, recovery, and follow-up analysis. Because Peanut Gallery is a web application, its chief concern deals with the threat of web application attacks that succeed in probing or breaching data stores and sensitive files. In responding to a probe or hack, the escalation manager needs to strike an important balance between two opposing trade offs based on the risks around the attack. The tradeoff sits between immediately locking the attacker out versus allowing the attacker to intrude in order to gather as much identifying forensic data as possible for the purpose of a criminal prosecution.

If the objective is to collect forensic data, the first order of business is to identify the source of the attack, capturing and documenting audit trail data such as log files. It is imperative that the escalation manager is contacted within 30 minutes who will provide instructions on how to proceed based on the information. In the event that the attack source cannot be identified, Peanut Gallery's CIRT team should consult the Community Emergency Response Team (CERT). If the approach is to remove the hacker from the system immediately, the security engineer needs to record a forensic snapshot of the system logs and files before setting up Web application firewalls (WAF) and eradicating any processes the attacker used or files the attacker left behind. Account credentials need to be changed once the intruder is locked out. Any necessary breach disclosures need to be released. Restoring the system to a normal state takes precedent at this point. Once recovery is complete, the legal or public relationships representative needs to report it to other security agencies and a follow up report needs to be documented.

G. FINAL SECURITY REVIEW

A final security review was conducted prior to release of Peanut Gallery in order to evaluate the threat model, static analysis, dynamic analysis, and quality gates/bug bars. The findings of this review were then used in order to evaluate the final status of Peanut Gallery's requirements for a successful release. The team gave the application a grade of Passed FSR with Exceptions, as detailed below.

Given the threat model diagram compiled back in the Design section of this report, it seemed that attack methods such as spoofing in order to gain greater access to developer tools, tampering by exploiting vulnerabilities due to version specific vulnerabilities or SQL injection, and improper information disclosure leading to the access of information by non-admins or developers were adequately addressed. However, due to being a web application, Peanut Gallery is still yet susceptible to different means of DoS attacks. While services such as cloudflare could provide some protection from targeted DoS attacks, the login page itself can also be overwhelmed by a DoS attack consisting of a large number of new registrations or login attempts.

Static analysis during development was provided by ESLint. Because the tool's purpose is primarily intended to audit code in real time in order to look for specific sets of errors applicable to the project by the developers, it seems there is little to no security concern associated with this tool. ESLint becomes problematic when the tool is set up poorly. In those instances, we risk vulnerable code choices passing through unnoticed.

Dynamic analysis was initially provided only by Google Chrome's developer tools. The Chrome debugging tool allowed us to create runtime autopsies of the application network traffic and breakpoints around events such as form submissions, mouse clicks, XML/HTTP requests, and animations. The Chrome debugging tool was particularly useful in flagging vulnerabilities that gave users the ability to edit data that they otherwise should not have had the permission to. The subsequent use of HCL Software's Appscan offered a more thoroughly industry compliant "second opinion" approach with a UI and functionality that more closely mirrored an antivirus scan. Performing regular dynamic analysis allowed the team to recognize potential avenues of attack in order to eliminate the possibility of them being used maliciously.

During the design stage, our team created security gates and bug bars that could potentially be issues in the implementation of our project, in an attempt to guide implementation in a way that would address these issues early. Since our product was closely involved with the PII of our users, our privacy and security bug bars focused on the handling of this PII and ways in which a malicious attack could attempt to gain access to it. In the implementation of our product, we ensured that our users PII would be adequately protected from malicious attacks and stored securely in a mongoDB database.

H. CERTIFIED RELEASE & ARCHIVE REPORT

Release Version: The Peanut Gallery via Github

The Peanut Gallery is a survey hosting web application that allows users to create a login account and take surveys that are hosted in the web application. Those representing organizations would also be able to host their surveys and receive the responses of users who completed the survey.

Version Number: Initial release version 1.0

Future development plans include a more dynamic system for allowing surveys to be created and hosted on our web application, and quality of life features such as allowing users to query all their previous survey responses. If possible, working on a reward system in order to incentivize users to complete surveys would also be investigated more heavily.

Developer's/Technical Guide:

1. Install Meteor.

2. visit the Peanut Gallery application GitHub page and click the "Use this template" button to create your own repository initialized with a copy of this application. Alternatively, you can download the sources as a zip file or make a fork of the repo. Then, download a copy of the repo to your local computer.

3. In your local command prompt/terminal, cd into the text-realm/app directory and install libraries with: **"$ meteor npm install"**

4. Run the system with: **"$ meteor npm run start"**

If it all goes well, the application will appear at http://localhost:3000.

The settings.development.json file is intended to hold default database settings. This file contains default definitions for Users, Survey Entries, and Ratings/Reviews and the relationships between them. The initial system run may not initialize the database that comes with the template. To initialize this, run the system using: **"meteor run --settings settings.development.json"**

Works Cited

JetBrains. “IntelliJ IDEA.” *JetBrains*, JetBrains, 2019, www.jetbrains.com/idea/.

“The Most Popular Database for Modern Apps.” *MongoDB*, 2019, www.mongodb.com/.

“Meteor.” *Meteor.com*, 2019, www.meteor.com/.

Surveymonkey. “SurveyMonkey: The World’s Most Popular Free Online Survey Tool.” *Surveymonkey.com*, 2018, www.surveymonkey.com/.

“ESLint - Pluggable JavaScript Linter.” *ESLint - Pluggable JavaScript Linter*, 2013, eslint.org/.

Cowper, Bruce, et al. “SDL Security Bug Bar (Sample) - Security Documentation.” *Microsoft Docs*, 3 Dec. 2018, docs.microsoft.com/en-us/security/sdl/security-bug-bar-sample.

“Appendix M: SDL Privacy Bug Bar (Sample).” *Microsoft Docs*, 22 May 2012, docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307403(v=msdn.10)?redirectedfrom=MSDN.

“DOM Elements.” React, 2021, reactjs.org/docs/dom-elements.html.

Sukhov, Valerii. “React 17 Lifecycle.” Medium, Medium, 26 June 2018, medium.com/@valerii.sukhov/react-17-lifecycle-5b68946c813c.

Abramov, Dan, and Brian Vaughn. “React v16.9.0 and the Roadmap Update – React Blog.” *React Blog*, 8 Aug. 2019, reactjs.org/blog/2019/08/08/react-v16.9.0.html.

“Appendix k: SDL PRIVACY Escalation RESPONSE FRAMEWORK (SAMPLE).” Microsoft Docs, Microsoft Corporation, 22 May 2012, docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307401(v=msdn.10)?redirectedfrom=MSDN.