SECURITY RISK ASSESSMENT REPORT

Code Review – Subscribe

https://sit-chameleon-website-0bc2323.ts.r.appspot.com/

VERSION 0.0.1

02/12/2023

Miriam Azmy

TABLE OF CONTENTS

1. Code to be Reviewed:

```
import React, { Component } from 'react';
import { Button, Form, Col, Container, Row } from 'react-bootstrap';
import Chameleon_Logo from "./images/Chameleon_Logo.png";

/* The purpose of this form is to allow users to
signup to the Chameleon company newsletter service */
class newsSignupForm extends Component {

 /* Constructor class, utilising props and state to set email state and allow toast message functionality */
 constructor(props) {
  super(props);
  this.state = {
   email: '',
   showToast: false,
   toastMessage: '',
  };
 }

 // Set email function to set the email value that is captured from user input
 setEmail = (event) => {
  this.setState({ email: event.target.value });
 }

 /* Subscribe function, used to send POST request to server (Proxy API handles endpoints)
 and utilise error handling to inform user if they have successfully subscribed with toast,
 or toasts error message  */
 subscribe = (event) => {
  event.preventDefault();

  // Make a POST request to the '/subscribe' endpoint
  fetch('', { //Host URL of future GCP deployment to go here with subscribe proxy endpoint
   method: 'POST',
   headers: {
    'Content-Type': 'application/json',
   },
   body: JSON.stringify({ address: this.state.email }),
  })
   .then(response => response.json())
   .then(data => {
    console.log(data);
    // Update state and show successful toast message
    this.setState({
     showToast: true,
     toastMessage: 'Subscription to the Chameleon newsletter successful!'
    });
   })
   .catch((error) => {
    console.error('Error:', error);
    // Update state and show unseccessful toast message
    this.setState({
        showToast: true,
        toastMessage: 'Subscription to Chameleon newsletter failed. Please retry shortly.'
      });
    });
  }

  render() {
    return (

      /* div family utilsed to apply style to web form and its various assets */
      <div style={{ display: 'flex', justifyContent: 'center', alignItems: 'center', height: '100vh', width: '40%', margin:
'20px auto' }}>
        <Col>
          <div style={{ backgroundColor: 'white', padding: 20, border: '1px solid black', height: '100%' }}>
            <Container className="d-flex flex-column align-items-center">
              <Row className="text-center">
                <Col style={{ padding: 10 }}>
                  { /* Chameleon logo */}
                    <img src={Chameleon_Logo} style={{ width: 225, height: 225, display: 'block', margin: '0 auto' }}
alt="Logo" />
                </Col>
              </Row>

              { /* web form, instructing uses on how to signuup to Chameleon newsletter */}
              <Form className="d-flex flex-column align-items-center text-center" >
                <p style={{ fontSize: 20, fontWeight: 'bold' }}>Sign up to the Chameleon newsletter!</p>
                <p>Enter your email address in the field below to hear about our latest innovations.</p>
                <Form.Control
                  style={{
                    height: 40,
                    borderColor: 'black',
                    borderWidth: 1,
                    width: 200,
                    margin: 10,
                    paddingLeft: 10,
                  }}
                  type="email"
                  required
                  value={this.state.email}
                  onChange={this.setEmail}
                />

                { /* Button used to subscribe to newsletter, which triggers handleSubmit function onClick */}
                <Button
                  type="submit"
                  style={{
                    backgroundColor: 'green',
                    padding: 10,
                    borderRadius: 5,
                    margin: 10,
                  }}
                  onClick={this.subscribe}
                >
                  Subscribe
                </Button>

                { /* Display toast state after user selects subscribe button */}
                {this.state.showToast && <p>{this.state.toastMessage}</p>}

              </Form>
            </Container>
          </div>
        </Col>
      </div>
    );
  }
}

export default newsSignupForm;
```

2. Purpose of the Code

The provided code presents a React component designed to facilitate user sign-up for the Chameleon company newsletter. The form includes an email input field and a subscription button, utilizing a POST request to a server endpoint for subscription handling. The purpose of this code is to enhance user engagement by enabling them to easily subscribe to the Chameleon newsletter service. The component incorporates visual elements such as the Chameleon logo, providing a branded and user-friendly experience.

3. Code Structure and Organization

The code maintains a clear and organized structure, utilizing React Bootstrap components for the form layout. It follows a modular approach, encapsulating related functionalities within the component. The structure includes well-defined sections for the Chameleon logo, a descriptive text, the email input field, and the subscription button. Styling is applied using inline styles and Bootstrap classes, ensuring a cohesive and visually appealing presentation.

4. Use of Data Structures

The data structure in this code is relatively straightforward, focusing on managing the state for the email input and toast messages. The state object includes properties for email, showToast, and toastMessage. The email input value is updated through the setEmail function, and the subscribe function utilizes the email state to send a POST request to the server. The code effectively utilizes JSON for data exchange with the server, adhering to best practices for handling form data.

5. Commenting and Documentation

While the code is well-structured, it lacks comprehensive comments that could provide further clarity on specific functionalities or the purpose of certain sections. Adding comments to describe the purpose of functions, the server interaction, and the overall flow of the code would enhance its readability and aid future maintenance.

6. Coding Standards Adherence

The code adheres to React coding standards and integrates Bootstrap components for consistent styling. The use of inline styles is minimal, and the code maintains a clean and readable appearance. Adherence to coding standards ensures consistency and facilitates collaboration among developers.

7. Code Complexity

The code complexity is relatively low, focusing on essential form functionalities and server communication. However, there is room for improvement in terms of handling more complex scenarios, such as additional form validations and user feedback. Adding comments and documentation can assist developers in understanding the code's logic more effectively.

8. Potential Problem Areas

The potential problem areas in the submitted code highlight critical cybersecurity and data integrity considerations. While the code is well-written, the addition of client-side email format checking stands out as a critical improvement to protect against potential injection attacks or data manipulation. Implementing a strong validation mechanism on the client side guarantees that users enter properly formatted email addresses, resulting in a more secure and user-friendly experience.

It is critical to craft error messages that strike a balance between providing contextual information for users and suppressing specific facts that may aid attackers. Vague error messages may inadvertently reveal sensitive information, whereas informative but non-disclosing notifications enable users to understand and resolve subscription difficulties correctly. Furthermore, emphasising the significance of strong server-side validation underscores the need for a multi-layered approach to data integrity, preventing hostile actors from circumventing client-side checks and transmitting unauthorised or damaging material directly to the server. Overall, the proposed modifications lead to a more robust and secure newsletter registration form, consistent with best cybersecurity practices in web application development.

9. Security and Performance Checks

The provided code for the new signup form component demonstrates a commitment to handling user data securely through the utilization of HTTPS and the incorporation of a server-side component to manage newsletter subscriptions. However, there are areas where additional security measures can be implemented to further enhance the resilience of the form.

Client-Side Validation: While the form uses server-side validation to handle subscription requests, incorporating client-side validation, particularly for the email entry, can improve the user experience by preventing incorrectly structured data from being sent. Implementing JavaScript-based validation for email formats assures data integrity and eliminates the possibility of delivering incorrect data to the server.

Enhancement of Error Messages: The error messages issued in the catch block during subscription attempts suggest a possible area for development. A better user experience can be achieved by creating more user-friendly error messages that provide precise guidance on the nature of subscription failures. It is critical to strike a balance between informativeness and security to ensure that users receive valuable feedback without exposing vital facts that dangerous actors could exploit.

10. Proposed changes to Code

```
// Import necessary modules and components
import React, { Component } from 'react';
import { Button, Form, Col, Container, Row } from 'react-bootstrap';
import Chameleon_Logo from "./images/Chameleon_Logo.png";

// Define the NewsSignupForm component
class NewsSignupForm extends Component {
 constructor(props) {
  super(props);
 }

 setEmail = (event) => {
  this.setState({ email: event.target.value });
 }

 subscribe = (event) => {
  event.preventDefault();

  // Validate email format
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (!emailRegex.test(this.state.email)) {
   this.setState({
    showToast: true,
    toastMessage: 'Invalid email format. Please enter a valid email address.',
   });
   return;
  }

  fetch('', {
   method: 'POST',
   headers: {
    'Content-Type': 'application/json',
   },
   body: JSON.stringify({ address: this.state.email }),
  })
   .then(response => response.json())
   .then(data => {
    console.log(data);
    this.setState({
     showToast: true,
     toastMessage: 'Subscription to the Chameleon newsletter successful!',
    });
   })
   .catch((error) => {
    console.error('Error:', error);
    this.setState({
     showToast: true,
     toastMessage: 'Subscription to Chameleon newsletter failed. Please retry shortly.',
    });
```

```
      });
  }

  render() {
    return (
      <div style={{ display: 'flex', justifyContent: 'center', alignItems: 'center', height: '100vh', width: '40%', margin:
'20px auto' }}>
        <Col>
          <div style={{ backgroundColor: 'white', padding: 20, border: '1px solid black', height: '100%' }}>
            <Container className="d-flex flex-column align-items-center">
              <Row className="text-center">
                <Col style={{ padding: 10 }}>
                      <img src={Chameleon_Logo} style={{ width: 225, height: 225, display: 'block', margin: '0 auto' }}
alt="Logo" />
                </Col>
              </Row>

              <Form className="d-flex flex-column align-items-center text-center" >
                <p style={{ fontSize: 20, fontWeight: 'bold' }}>Sign up to the Chameleon newsletter!</p>
                <p>Enter your email address below to hear about our latest innovations.</p>
                <Form.Control
                  style={{
                    height: 40,
                    borderColor: 'black',
                    borderWidth: 1,
                    width: 200,
                    margin: 10,
                    paddingLeft: 10,
                  }}
                  type="email"
                  required
                  value={this.state.email}
                  onChange={this.setEmail}
                />

                <Button
                  type="submit"
                  style={{
                    backgroundColor: 'green',
                    padding: 10,
                    borderRadius: 5,
                    margin: 10,
                  }}
                  onClick={this.subscribe}
                >
                  Subscribe
                </Button>

                {this.state.showToast && <p>{this.state.toastMessage}</p>}
              </Form>
            </Container>
          </div>
      </Col>
    </div>
  );
 }
}

export default NewsSignupForm;
```

11. Findings

The code accomplishes its goal of building a newsletter signup form. It's well-structured, follows coding standards, and uses Bootstrap components for styling. However, it lacks explicit comments that would provide more information about the code's capabilities. HTTPS usage is a security issue, but further validation tests, particularly on the server side, are advised. It is worth noting that in the live environment it is non-functioning. Additionally, user experience can be improved by performance enhancements such as client-side email format validation and asynchronous asset loading. The code handles potential subscription failures, however, error messaging may be improved for greater user help. The code serves its purpose in general, but integrating comments and addressing security and performance concerns would improve its overall quality and maintainability.