ASGARDEO by WSO2

Mastering Secure Login Mechanisms for React

Hello!



Brion Silva

Technical Lead

brion@wso2.com

About Me 👀

- Works at WSO2
- Focuses on the Identity & Access Management Products
- One of the core maintainers the Asgardeo JavaScript based SDKs & the WSO2 Design System
- Co-Host of JavaScript Colombo meetup group
- ❤️ Loves everything Front-End!

Today’s Lineup

- Security in React 🔐
  - XSS in React
- Adding Login to a React App 🛡️ With Demo
  - What is Login
  - Why use an IdP (Identity Provider)
  - How would you implement the Login?
  - Why Asgardeo
- How does a XSS attack usually happens 🎯 With Demo
  - How can we protect app users



Security in React 🔐

What makes React so popular? 🤔

⚡ 1. Component-Based Architecture

React's modular approach allows developers to create **reusable UI components**, making development faster and more efficient.

🔥 2. Virtual DOM for High Performance

Efficiently updates only the necessary parts of the UI.

🛠️ 3. Strong Developer Ecosystem

A vast community, rich documentation, and extensive third-party libraries.

🌍 4. Cross-Platform Capabilities

With **React Native**, developers can build mobile apps using the same principles as React web apps, increasing code reusability.

📦 5. Rich Tooling

Tools like React Developer Tools and powerful state management libraries make React a great choice.

🔐 6. Built-in Security

React **automatically escapes user input** to prevent **Cross-Site Scripting (XSS) attacks**.

💡 7. Backed by Meta (Facebook)

React's continuous development and improvements are ensured by Meta (formerly Facebook), keeping it up to date with modern web standards.

“React escapes all content inside JSX by default, preventing malicious script execution (XSS).”

Lets see this in action 🙌

React code

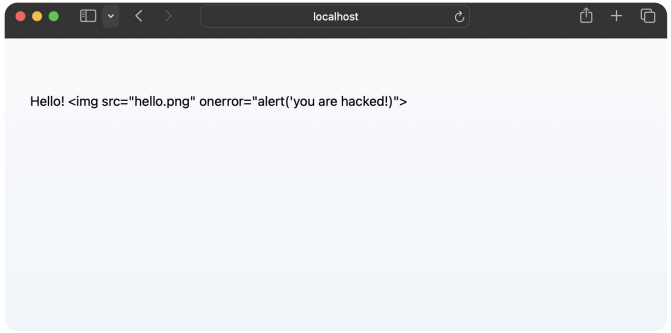
```
export function Greeting() {
  const searchParams = new URLSearchParams(window.location.search);
  const greeting = searchParams.get('greeting') || 'Hello';

  return <h1>{greeting}</h1>;
}
```

Malicious code

```
<URL>?greeting=Hello! <img src='hello.png' onerror='alert("you are hacked!")/>
```

Output



Problematic Implementations: dangerouslySetInnerHTML

React code

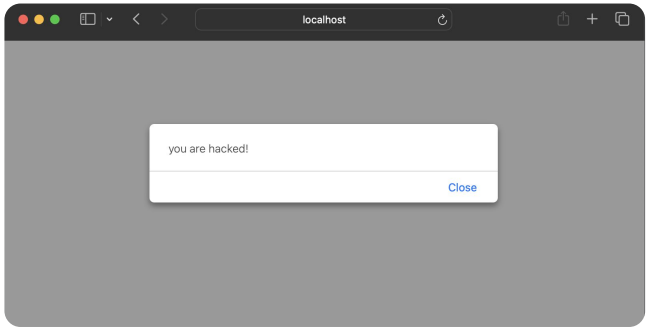
```
export function Greeting() {
  const searchParams = new URLSearchParams(window.location.search);
  const greeting = searchParams.get('greeting') || '<b>Hello</b>';

  return <h1 dangerouslySetInnerHTML={{__html: greeting}} />;
}
```

Malicious code

```
<URL>?greeting=Hello! <img src='hello.png' onerror='alert("you are hacked!")/>
```

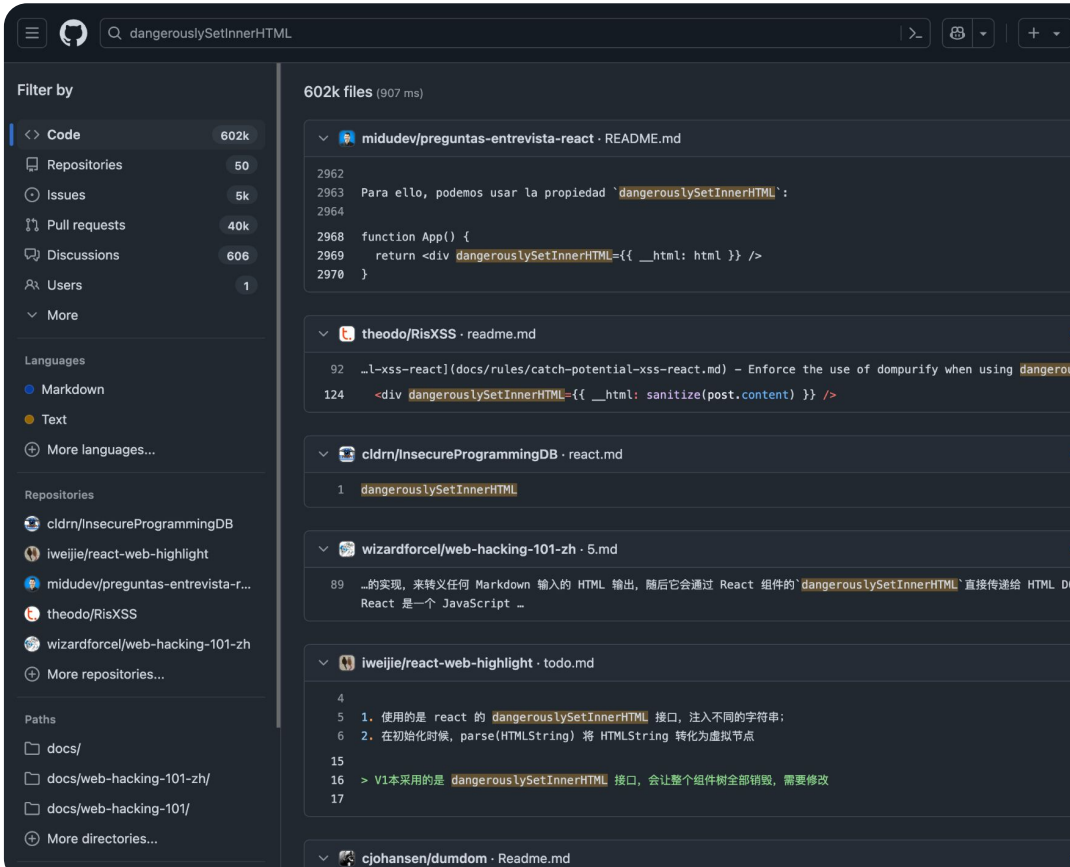
Output



Who does this? 😅

More than half a million files on **GitHub** live life **dangerously**

because….YOLO



Not just dangerouslySetInnerHTML

There are other ways to bypass React's escaping

- Trusting HTML Parsers (ex: html-react-parser)



- Accessing the DOM directly with `**refs**`, `**findDOMNode**`, `**document.getElementById**` and performing `**innerHTML**`.

- Directly rendering externally passed URLs.
  - **javascript:** is a valid URI scheme.

How to avoid XSS

- Try not to bypass React's data binding
- Always use dangerouslySetInnerHTML with sanitization (DOMPurify is a really good library)
- Don't trust parsers.
- Don't manipulate the DOM directly.
- Don't use insecure URLs directly without validations.
- Use content security policies (CSPs)

Scared ?



Adding Login to a React App 🛡️

What is Login? 🔐

“Login refers to the process where a user identifies themselves to an application in order to gain access to personalized features, resources, or data. During login, the system verifies the user's identity, often using credentials like a username and password.”

Some IAM Jargon & concepts that’ll be used in the upcoming slides

- **OAuth 2.0/OIDC**: Protocols for secure authentication and authorization.
- **Scopes:** Defines what kind of access is requested or granted.
- **Authorization**: Ensures that the user is who they claim to be.
- **Authentication**: The process of verifying the identity of a user.
- **Tokens**: Credentials that represent the user's authenticated state (e.g., JWT - JSON Web Tokens).
- **Session**: A temporary period when a user can interact with the app without needing to log in repeatedly.
- **SSO (Single Sign-On)**: Allows users to authenticate once and gain access to multiple services or apps.
- **Identity Provider (IdP)**: A service that manages user identities and handles authentication (e.g., Asgardeo, Auth0).

Let’s take a sample scenario

New Startup Idea 💡

🚀 Introducing CodeFit.ai

CodeFit.ai is a **fitness tracking web app for developers**, helping them stay active with AI-powered workout recommendations based on their coding habits.

Key Requirements

- **AI Agent** to coach and assist the users.
- Landing page to **register**.
- Without login, they can’t access the **AI chat** or **saved progress**.
- ⭐ Provides users with coupon codes to purchase subscriptions.

GO TO PROD 💥

01 of May 2025

How would you implement the Login?

1. 

   Use an IdP (Identity Provider) 🔒

2. 

   Use a Managed Auth Service 🔥

3. 

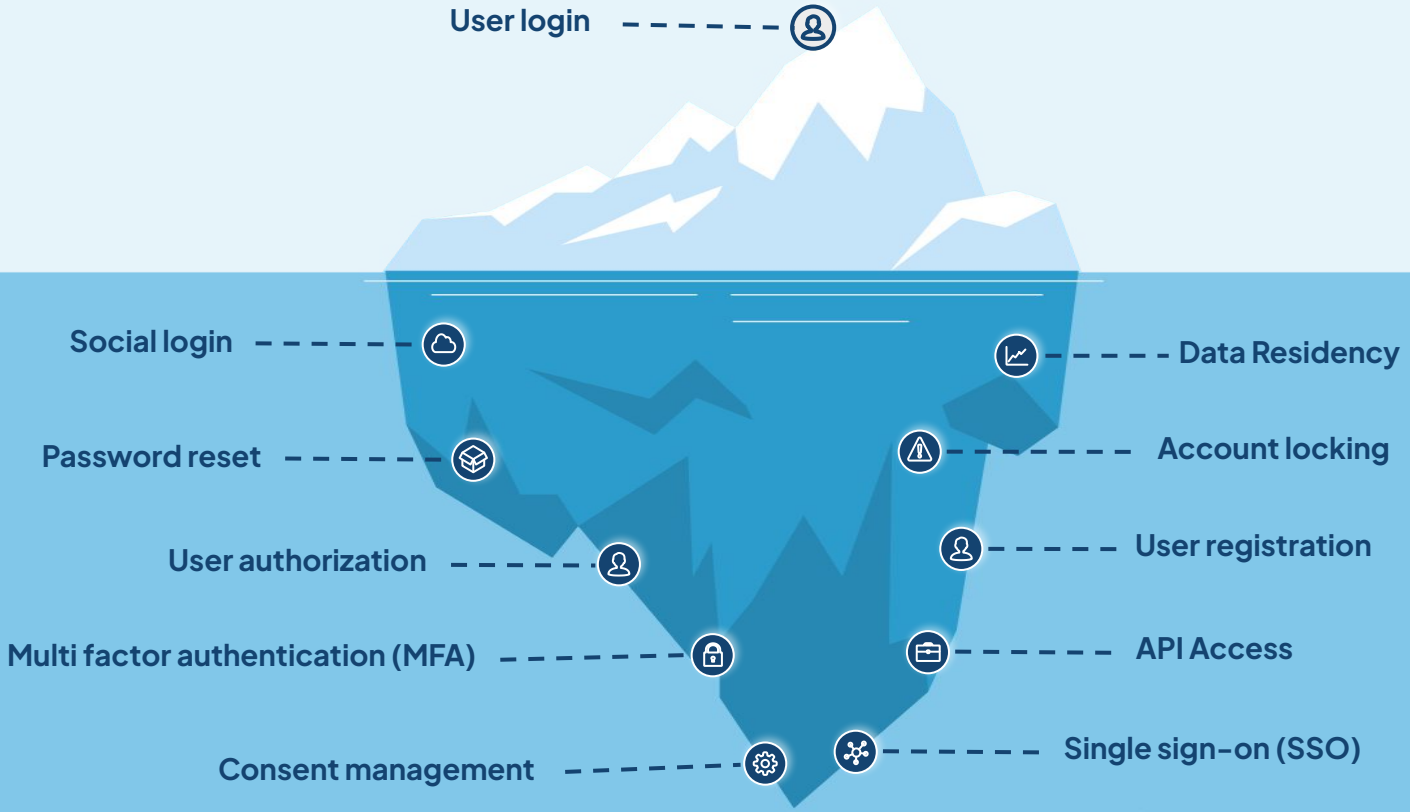
   Save usernames & passwords in the DB and call it a day 🗑️

4. 

   Implement everything from Scratch (DIY) 💥

quick poll.

Why use an IdP (Identity Provider) 🔓



Popular IdPs (Identity Providers) 🔓





















Why use Asgardeo

⭐ Seamless Authentication & Security
- Supports **OIDC, OAuth 2.0, and SAML** for secure user authentication.
- Built-in **Multi-Factor Authentication (MFA)**, password policies, and adaptive authentication.

⭐ Pretty generous free plan



Check here 👉

- **Up to $10,000** credits
- **Free $2,100** credits for **WSO2 Support**
- 1:1 time with a technical expert

⭐ Developer-Friendly open-source SDKs & APIs

⭐ Faster Go-To-Market
- Low Code UIs & AI support ✨

⭐ Ideal for Enterprises & Startups

Which SDK to use?

Is there a `react-auth`

Options 👇

Asgardeo Auth React SDK 🔐 (OIDC-based authentication)

Firebase Auth 🔥 (Google's identity solution)

Auth0 React SDK 🛡️ (OAuth & social logins)

Why use Asgardeo React SDK



⭐ Opensource (http://github.com/asgardeo)

⭐ Standard SDK

⭐ Enhanced token storage mechanisms

⭐ Built by React Developers

⭐ Good developer community

⭐ Internally used by WSO2 products

Why use Asgardeo React SDK

⭐ Vendor Agnostic

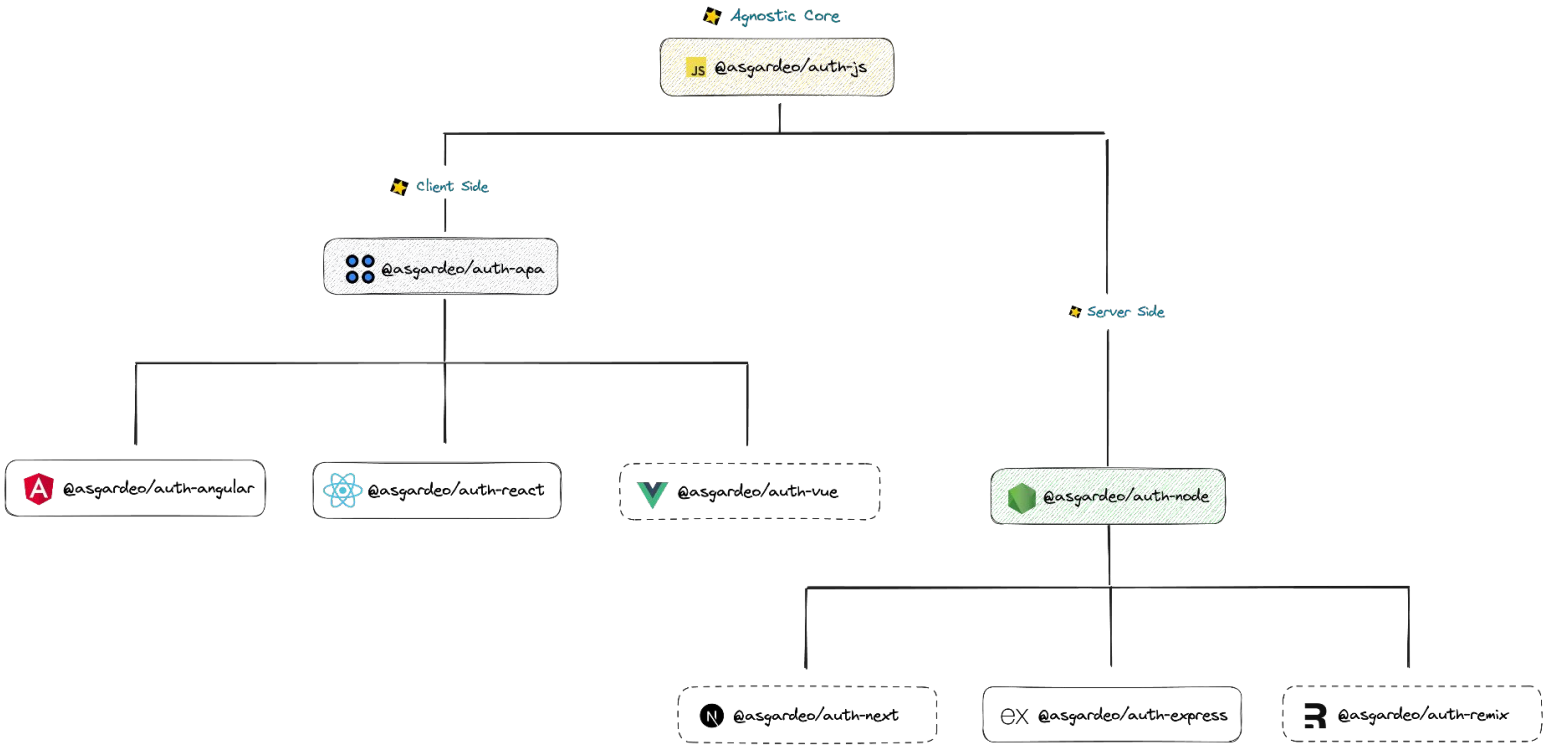
Supports OIDC Discovery

```tsx
import { AuthProvider } from '@asgardeo/auth-react';

function App() {
  return (
    <AuthProvider
      config={{
        clientID: '<your-client-id>',
        wellKnownEndpoint:
          'https://<your-tenant>.zitadel.cloud/.well-known/openid-configuration',
        scope: ['openid', 'profile'],
        signInRedirectURL: 'http://localhost:5173',
        signOutRedirectURL: 'http://localhost:5173',
      }}
    >
      <YourApp />
    </AuthProvider>
  );
}

export default App;
```

Architecture of Asgardeo JavaScript SDKs



DEMO TIME 👨‍💻

Sign Up to Asgardeo 🔗

Follow the React Quickstart Guide 🔗

How does a XSS usually happens 🎯

How does a XSS usually happens

1️⃣Reflected XSS (User Clicks on a Malicious Link) 🎯

2️⃣Stored XSS (Malicious Data is Saved) 💾

3️⃣DOM-Based XSS (Manipulating the Frontend) 🏗️



Reflected XSS

The app echoes the input from the URL directly into the HTML response.

Malicious URL

```
https://vulnerable-site.com/search?q=<script>alert('Hacked!')</script>
```

Server Response

```
<p>You searched for: <script>alert('Hacked!')</script></p>
```

Stored XSS

Attacker saves a harmful script in a comment, post, etc. It's stored and shown to future users.

Malicious Entry

```
Great post! <script>alert('I stole your session!')</script>
```

Server Response

```
<div class="comment">
  Great post! <script>alert('I stole your session!')</script>
</div>
```

DOM-Based XSS

Reads data from location.href or document.location, and injects it into the DOM without sanitization.

Malicious URL

```
http://localhost:5173/chat?message=<img src="x" onerror="alert('Hacked!')">
```

Server Response

```
return (
  <div>
    <h2>User Message:</h2>
    {/* 🧯 Dangerous: Direct injection into innerHTML */}
    <div dangerouslySetInnerHTML={{ __html: message }} />
  </div>
);
```

DEMO TIME 💥

Final Takeaways

- ✅ Always follow React best practices — let React handle the DOM safely.
- 🧼 Sanitize and validate any untrusted input before rendering.
- 🔐 Try to use an already established IdP for identity & access management.
- 🔒 Use secure storage for sensitive information (e.g., tokens).
- 🔍 Regularly audit dependencies and update vulnerable packages.
- 🛡️ Use a Content Security Policy (CSP) to mitigate XSS risks.
- 🧪 Implement security testing (e.g., unit, integration, and penetration tests).

Sign up to Asgardeo / Grab the Slides / Connect with Me

https://linktr.ee/brionmario



Question Time!

References and further reading

✨ Simplify React App Login

A simple breakdown of integrating login in React using Asgardeo.

📘 React Complete Guide by Asgardeo

Official complete guide to secure your React apps.

🧱 The Foundation of Secure Authentication

Understand the core principles behind secure login systems.

🧩 Asgardeo React SDK (GitHub)

Explore the open-source SDK powering secure authentication.

🛡️ OWASP Top 10

The top ten most critical web application security risks.

📄 Asgardeo Official Docs

Start integrating Asgardeo into your JavaScript apps with ease.

Thanks!

ASGARDEO*

by WSO2

Identity Management Simplified

wso2.com/asgardeo