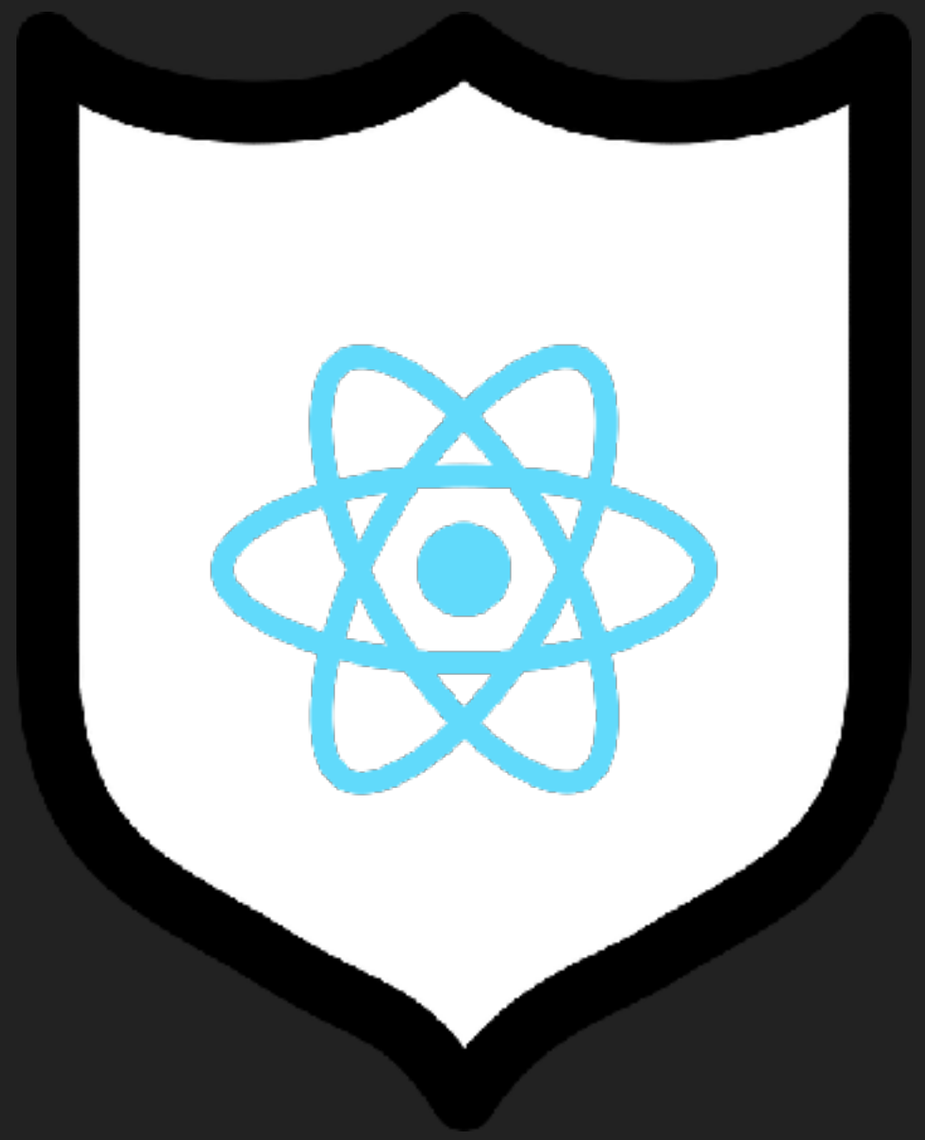


SHIELDS UP

SECURING REACT APPS

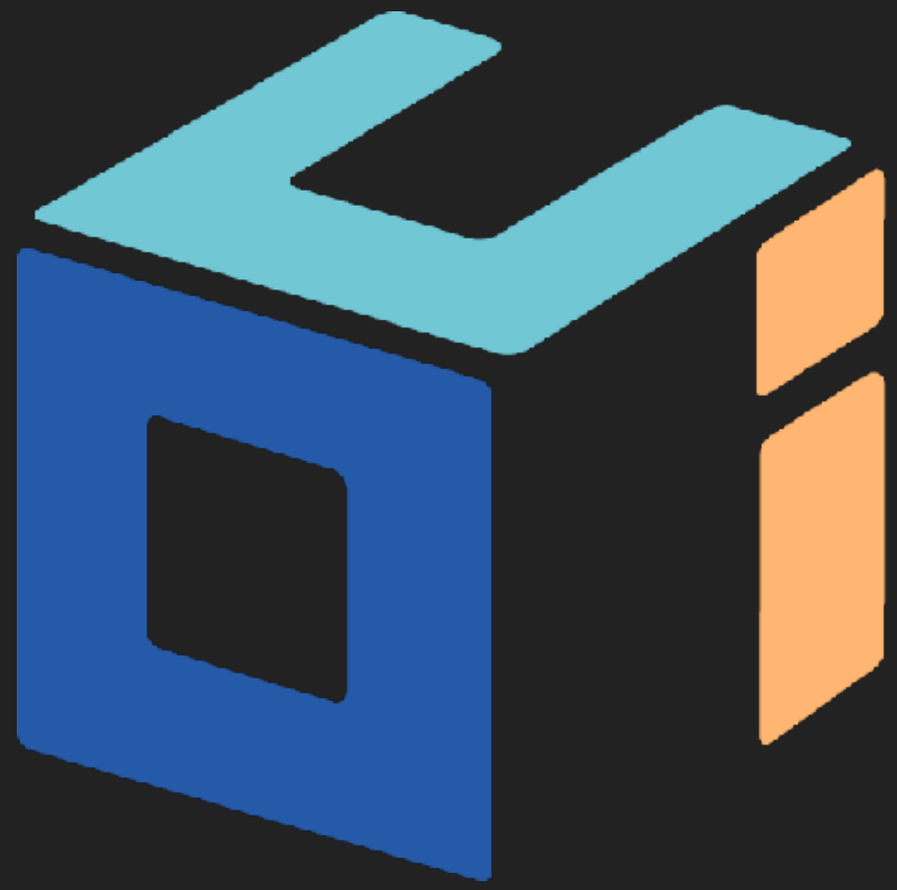
August 10, 2018

ZACHARY KLEIN

- Grails developer since 2010
- Frontend developer since 2015
- Joined OCI Grails team in 2015
- OSS contributor

ABOUT US

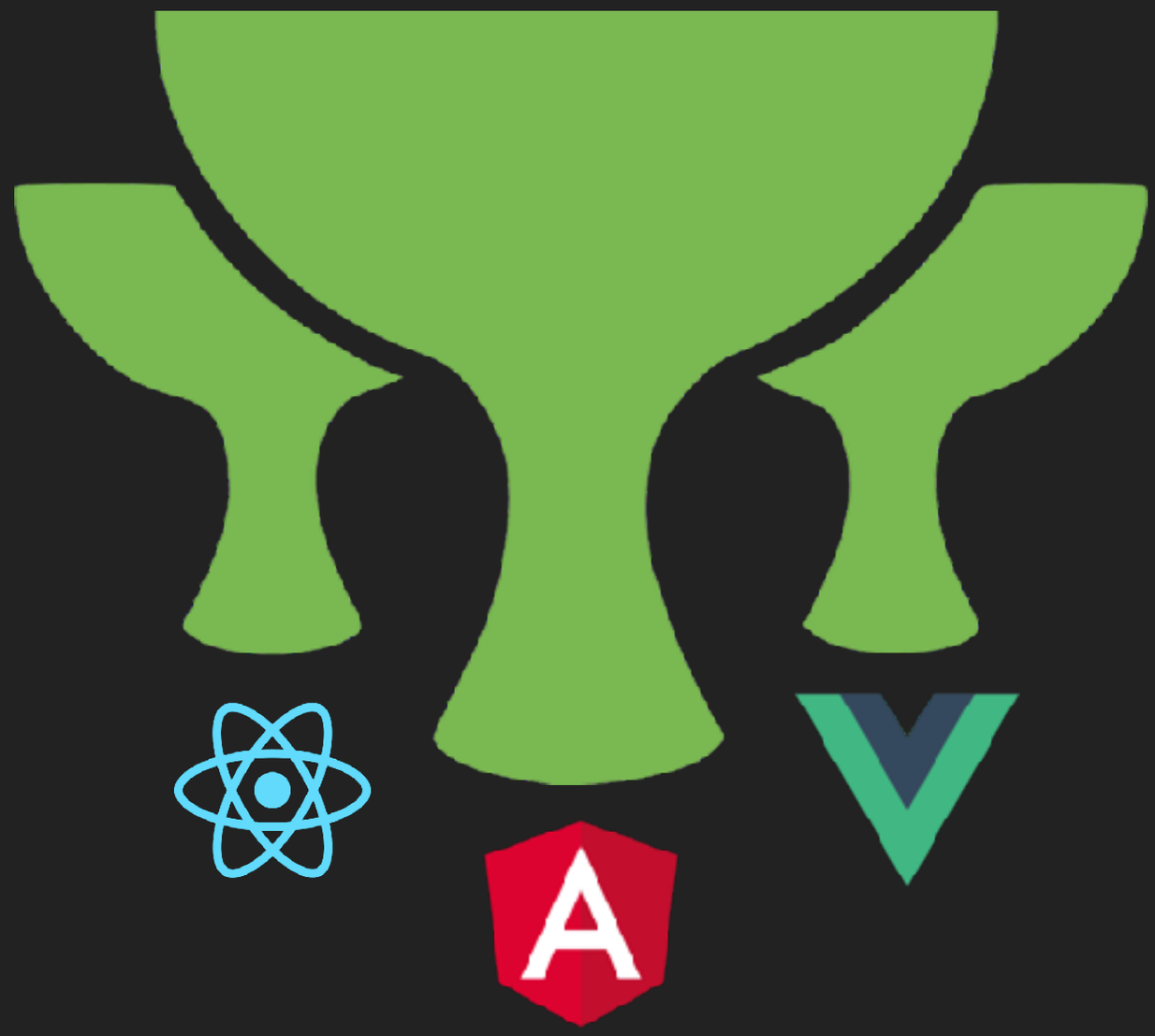
- Object Computing, Inc
  - Based in St Louis, MO
  - Consulting and training provider (24+ years experience)
- Corporate sponsor to the **Grails** & **Micronaut** frameworks and **Groovy** language
- https://objectcomputing.com



GRAILS

http://start.grails.org

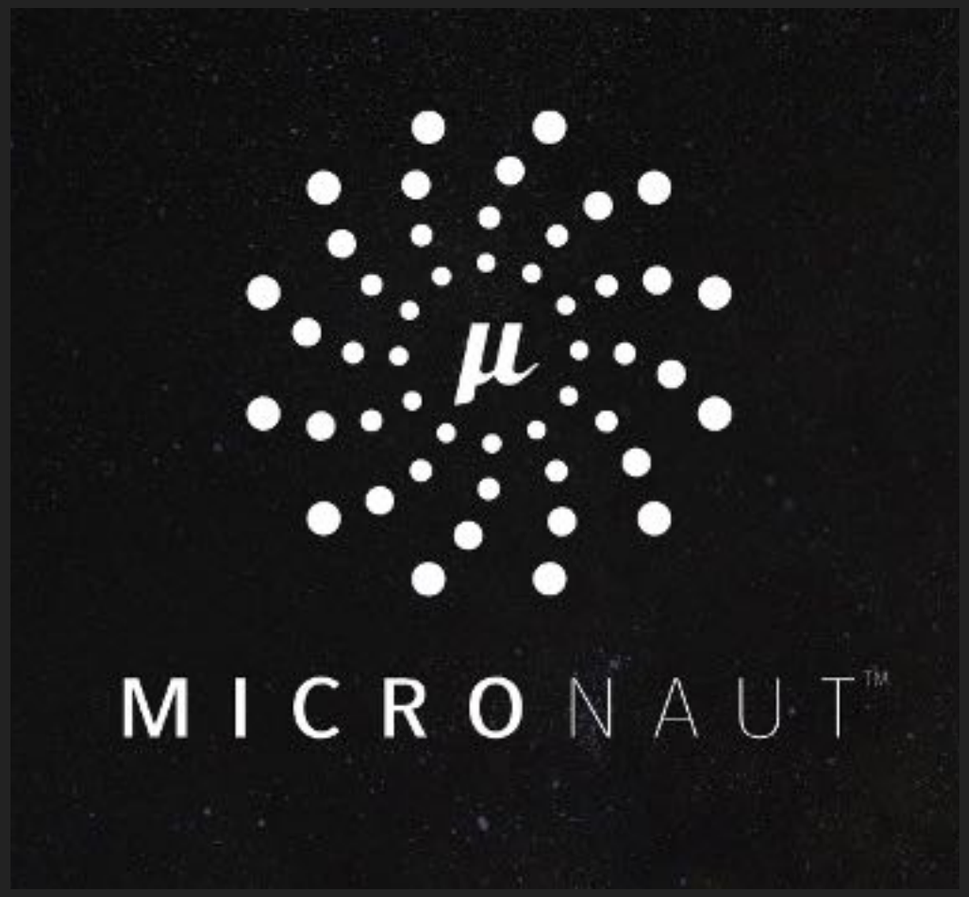
- Built over Spring Boot & Gradle
- Plugin system
- REST controllers, JSON views
- Single Page App support:
  - Angular
  - React
  - Vue.js



MICRONAUT

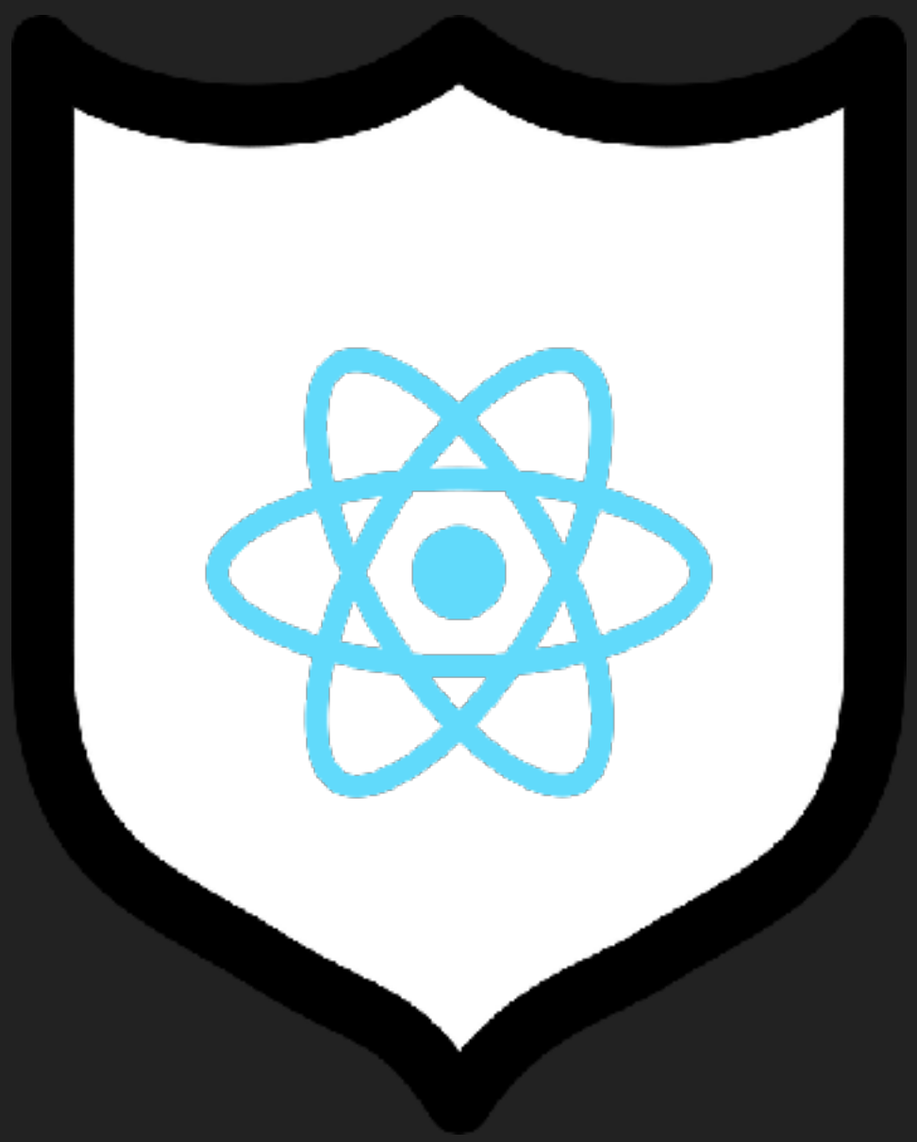
http://micronaut.io

- Full stack OSS JVM framework for microservices
- Subsecond startup time
- Uses as little as 7MB memory
- Reactive HTTP server & client
- “Natively” Cloud-Native
- Supports Java, Groovy, and Kotlin



Introduction/Sample Project: http://bit.ly/micronaut

SHIELDS UP - SECURING REACT APPS



RUN NPM AUDIT!

(or `yarn run audit`)

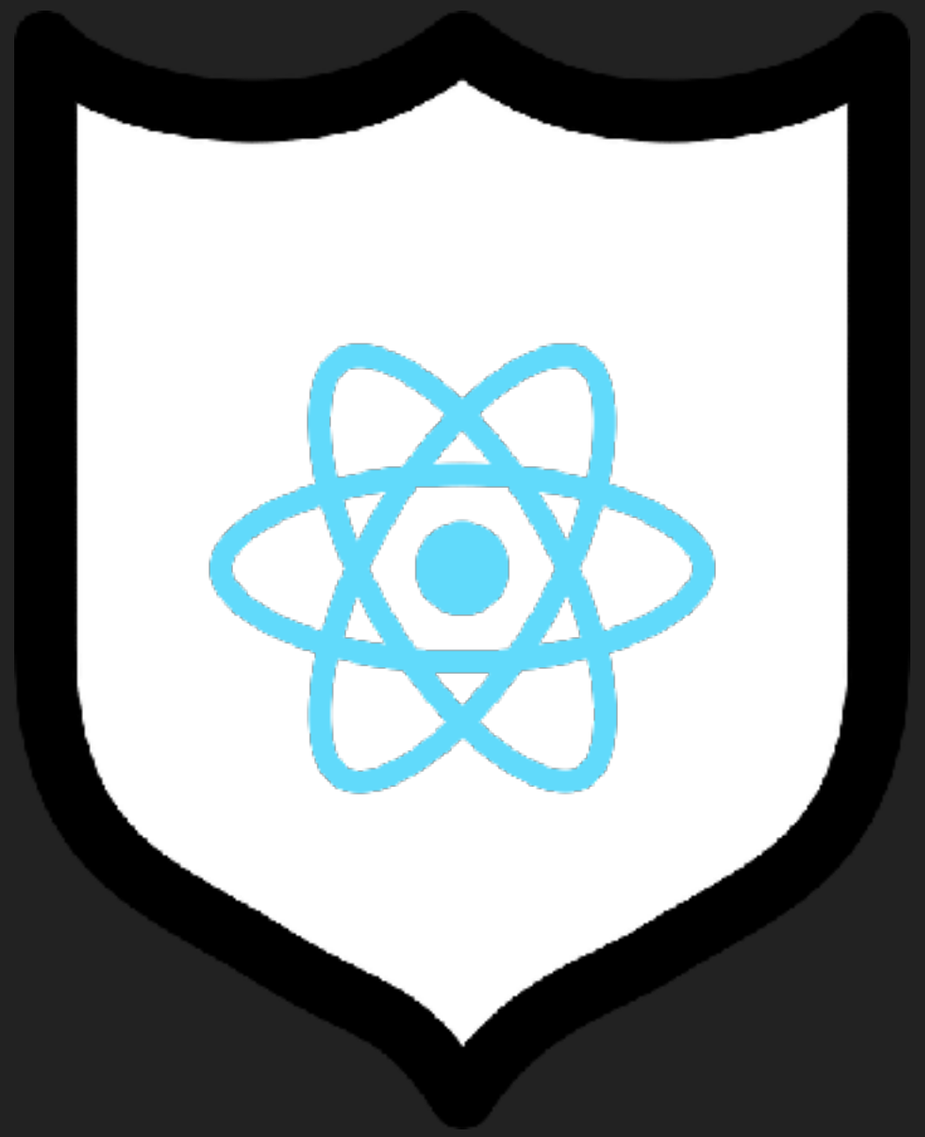
```
Zacharys-MacBook-Pro:client dev$ npm audit

                       === npm audit security report ===

# Run  npm install --save-dev react-scripts@1.1.4  to resolve 37 vulnerabilities
SEMVER WARNING: Recommended action is a potentially breaking change
```

High	Regular Expression Denial of Service
Package	tough-cookie
Dependency of	react-scripts [dev]
Path	react-scripts > webpack > watchpack > chokidar > fsevents > node-pre-gyp > request > tough-cookie
More info	https://nodesecurity.io/advisories/525

```
found 37 vulnerabilities (5 low, 27 moderate, 4 high, 1 critical) in 8239 scanned packages
  37 vulnerabilities require semver-major dependency updates.
```

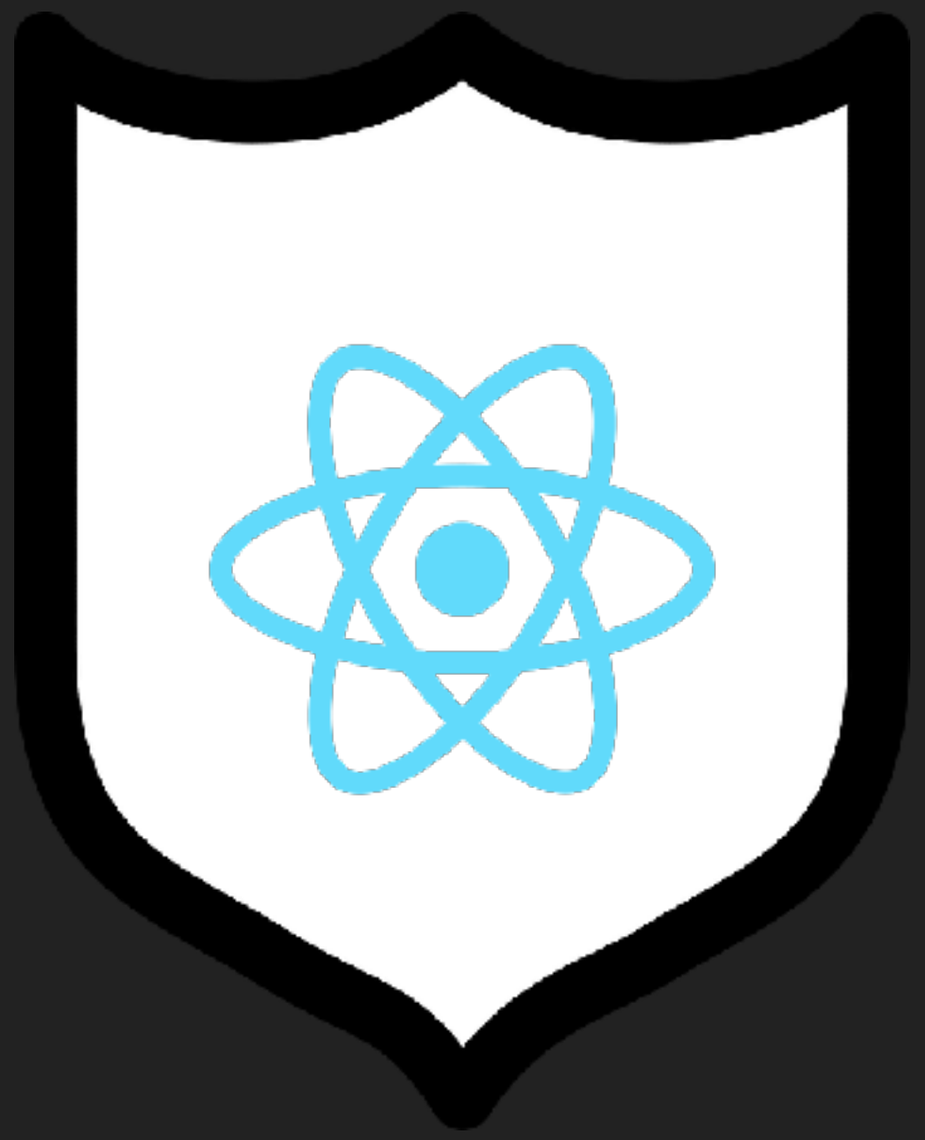
OVERVIEW

- Security in the world of SPAs
- Client-side security
  - Cross-site-scripting Prevention
  - Role-based Routing
- Server-side security
  - Stateless Authentication
  - Third-party authentication



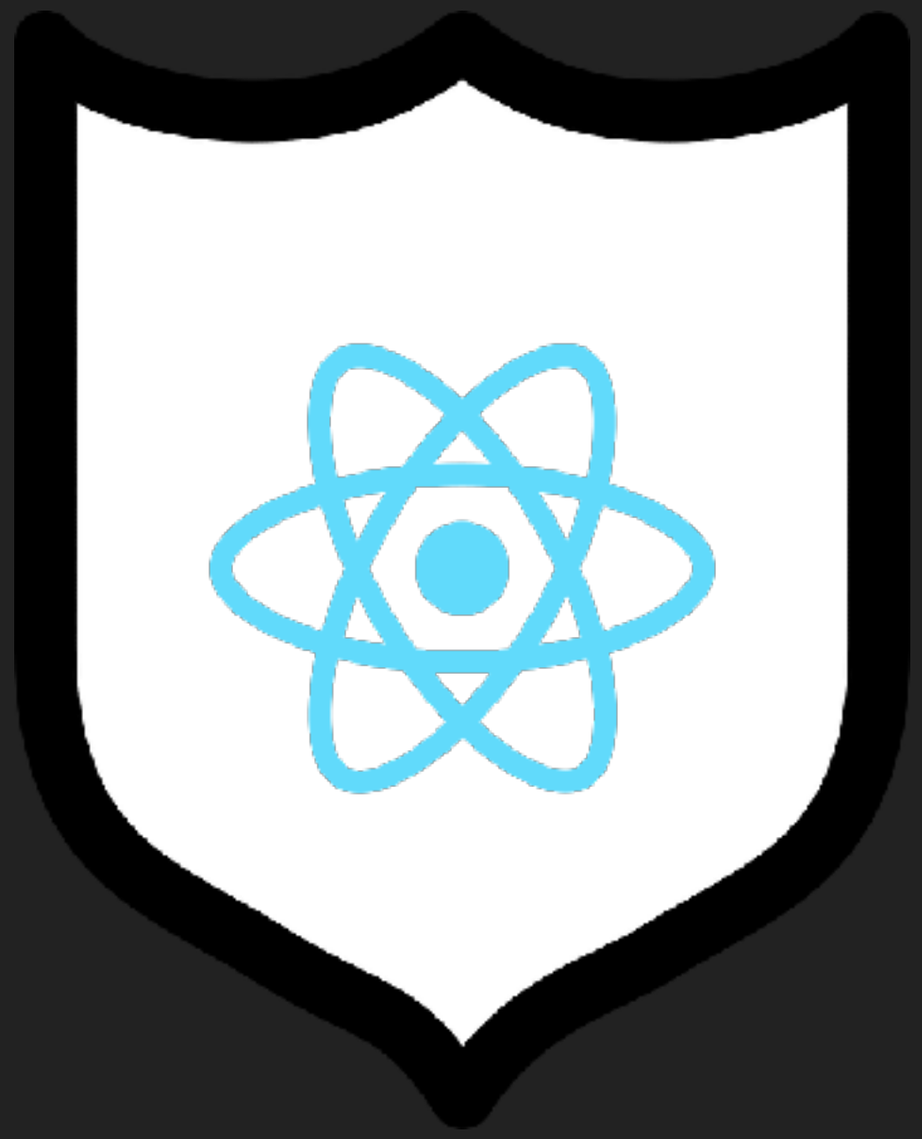
WHAT ARE WE PREVENTING?

- Unauthorized access to data
- Unauthorized access to UI
- Unexpected input



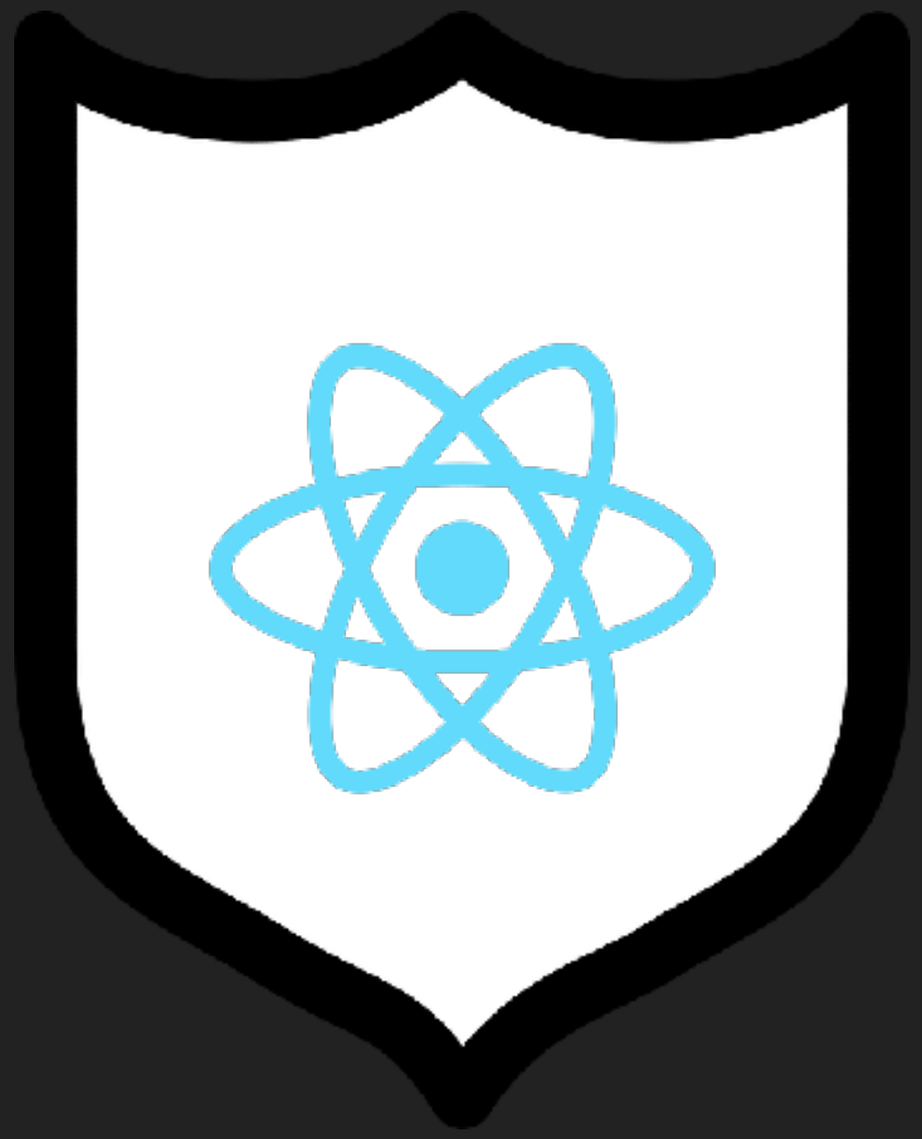
WHAT ARE WE PREVENTING?

- Unauthorized access to data
  - API authentication
  - Web Storage
- Unauthorized access to UI
- Unexpected input



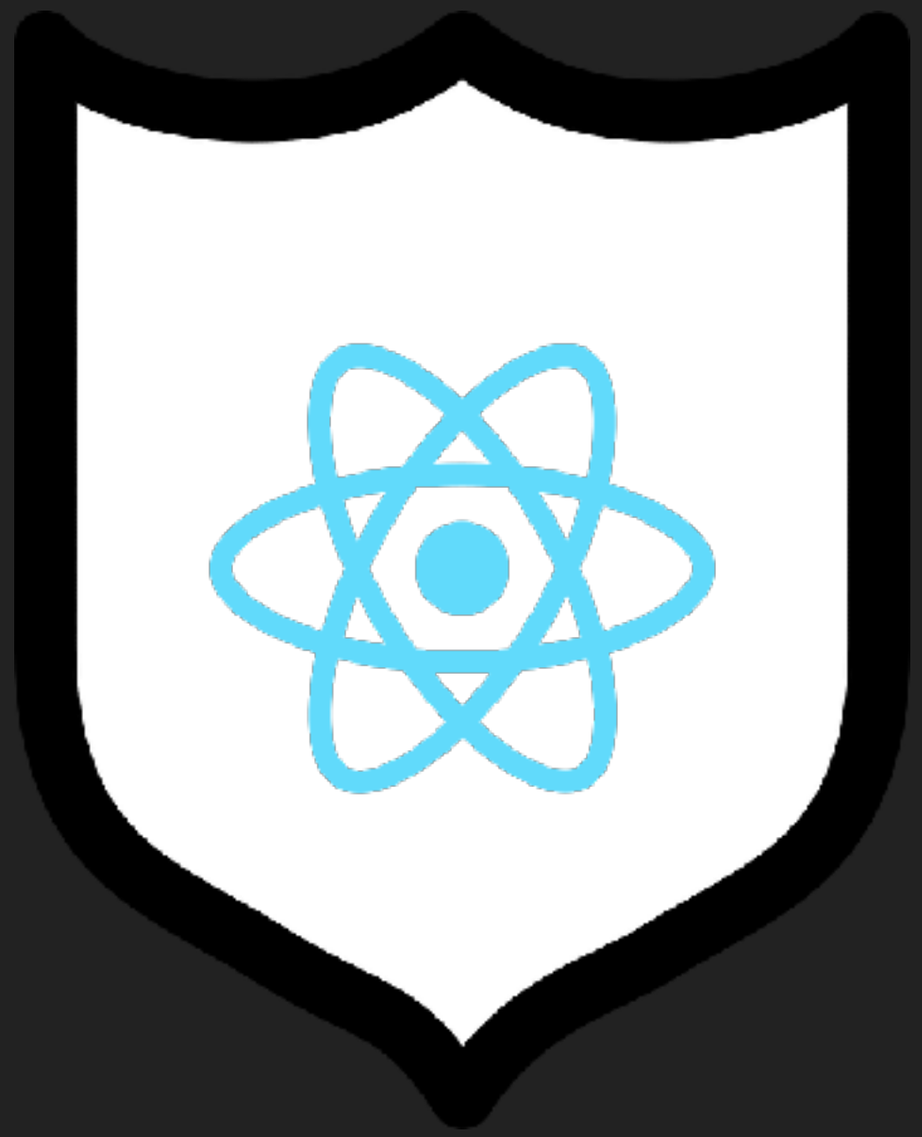
WHAT ARE WE PREVENTING?

- Unauthorized access to data
  - API authentication
  - Web Storage
- Unauthorized access to UI
  - Role-based Routing
- Unexpected input



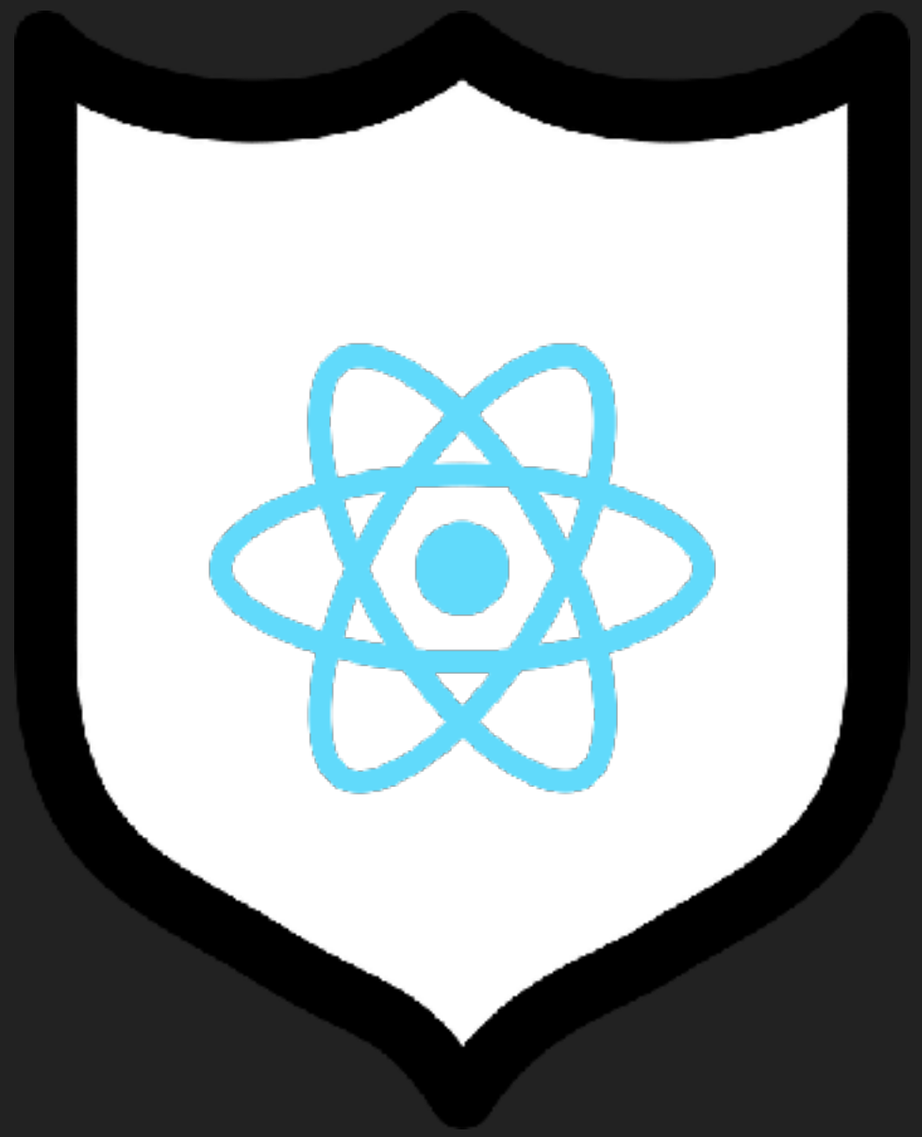
WHAT ARE WE PREVENTING?

- Unauthorized access to data
  - API authentication
  - Web Storage
- Unauthorized access to UI
  - Role-based Routing
- Unexpected input
  - Cross-site scripting (XSS)



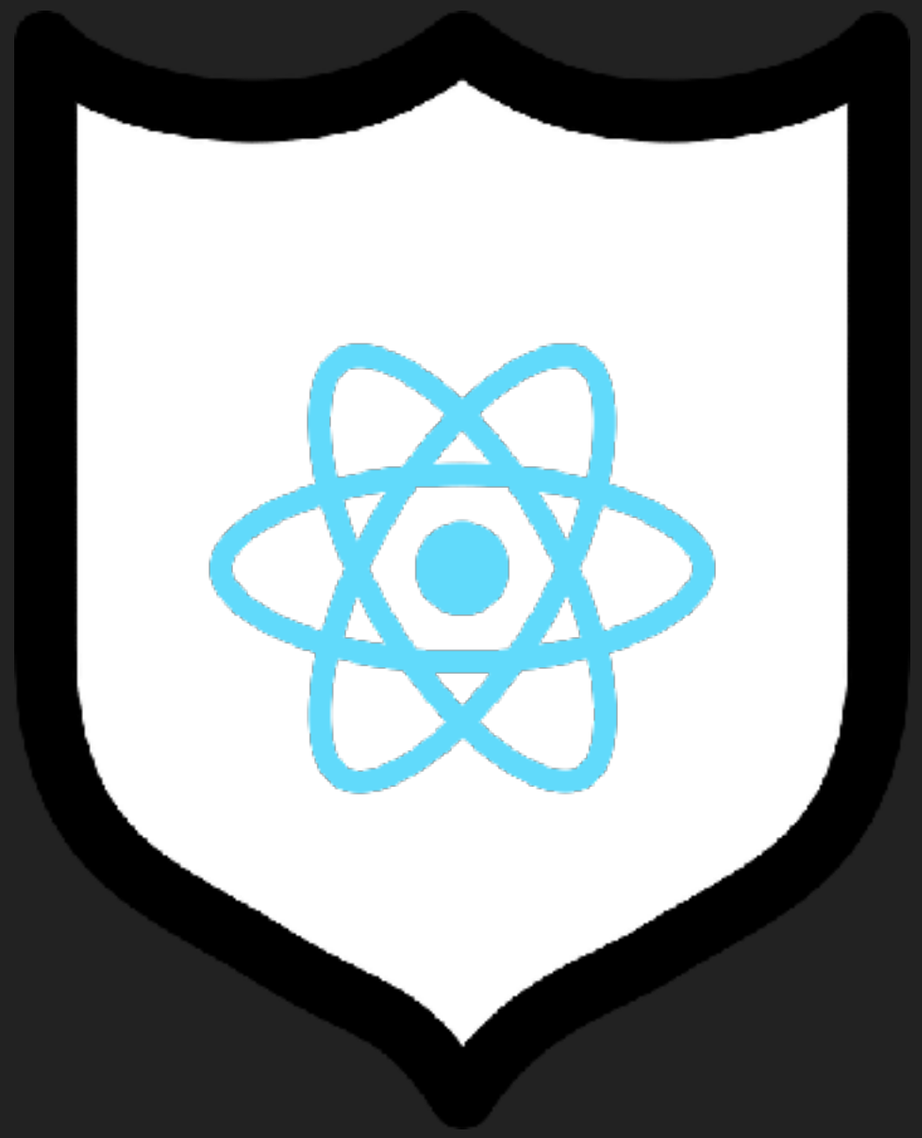
WHAT ARE WE PREVENTING?

- Unauthorized access to data
  - API authentication
  - Web Storage
- Unauthorized access to UI
  - Role-based Routing
- Unexpected input
  - Cross-site scripting (XSS)

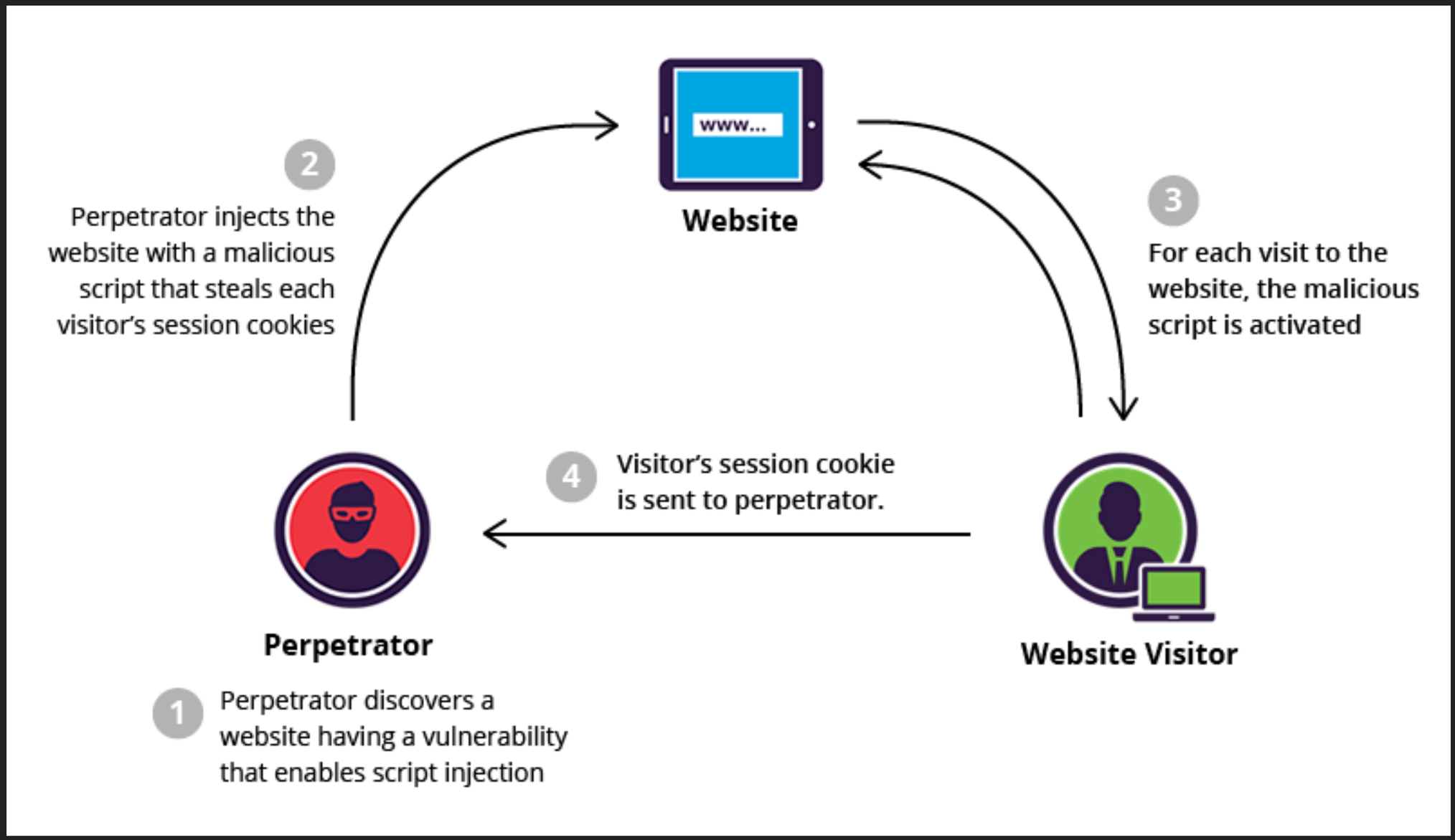


WHAT ARE WE PREVENTING?

- Unauthorized access to data
  - API authentication
  - Web Storage
- Unauthorized access to UI
  - Role-based Routing
- Unexpected input
  - Cross-site scripting (XSS)



CROSS SITE SCRIPTING



https://www.incapsula.com/web-application-security/cross-site-scripting-xss-attacks.html

CROSS SITE SCRIPTING & REACT

- React is XSS-safe by default - all content is escaped
- Potential loopholes: **`href`**, **`formaction`**
  - Server-side rendered content
  - **`dangerouslySetInnerHTML`**
  - Third-party JavaScript

WEB STORAGE

- Two options: Local Storage & Session Storage
- Allows web apps to store data local to the user's browser
- Storage is unique per origin (domain and protocol)
- Any JS running within the app can access the same storage
- Cannot be accessed by code executed from other domains

CROSS SITE SCRIPTING & WEB STORAGE

DEMO

COOKIES

- Cookies can be set with **secure** & **httpOnly** flags for storage - immune to XSS
- But… cookies are vulnerable to Cross-Site-Request-Forgery attacks
- Server can check for a CSRF Token to verify the request
- For Node Express: CSRF token generator: https://www.npmjs.com/package/csurf
- For Spring/Grails: https://docs.spring.io/spring-security/site/docs/current/reference/html/csrf.html

WEB STORAGE VS COOKIES?

https://stormpath.com/blog/where-to-store-your-jwts-cookies-vs-html5-web-storage

ROLE-BASED AUTHORIZATION & ROUTING

- Routing allows SPAs to expose “URLS” within the app
- Different content will be rendered based on the URL, query params, and other criteria
- Most popular JavaScript frameworks either include a router or a recommended third-party library
- React Router is the default for React
- Supports either hash history **or** HTML 5 browser history

http://myapp.com/#/my/route

http://myapp.com/my/route

ROLE-BASED AUTHORIZATION & ROUTING

```
const BasicExample = () => (
  <Router>
    <div>
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/about">About</Link></li>
        <li><Link to="/topics">Topics</Link></li>
      </ul>

      <hr/>

      <Route exact path="/" component={Home}/>
      <Route path="/about" component={About}/>
      <Route path="/topics" component={Topics}/>
    </div>
  </Router>
)
export default BasicExample
```

https://reacttraining.com/react-router/web/guides/quick-start

ROLE-BASED AUTHORIZATION & ROUTING

- Some content should only be available to certain users
- UI functionality and routes may still need to be hidden depending on a user's privileges (or roles)
- Note: **Data** still needs to be secured at the server!

ROLE-BASED AUTHORIZATION & ROUTING

- Simple Approach:
  - Store current user roles in state
  - Wrap routes/components in a container component
  - Container component accepts list of required roles & conditionally renders its **children**

ROLE-BASED AUTHORIZATION & ROUTING

```
import React from 'react'
import { array, bool } from 'prop-types'

const Authorized = {roles, allowedRoles, requireAll, children} => {

  const matchesRoles = required => {
    roles.indexOf(required) >= 0;
  }

  const show = requireAll ? allowedRoles.every(matchesRoles)
                          : allowedRoles.some(matchesRoles);

  return show ? children : null
}

Authorized.propTypes = {
  roles: array,
  allowedRoles: array,
  requireAll: bool
};

export default Authorized;
```

ROLE-BASED AUTHORIZATION & ROUTING

```
<Authorized allowedRoles={[‘ROLE_ADMIN', 'ROLE_ADMIN', 'ROLE_USER']}>
  <Profile user={user} />
</Authorized>
```

```
<Authorized allowedRoles={['ROLE_ADMIN', 'ROLE_MANAGER']}>
  <ManageEmployees />
</Authorized>

<Authorized allowedRoles={['ROLE_ADMIN', ‘ROLE_MANAGER']} requireAll={true}>
  <ManageDatabase />
</Authorized>
```

WITH REDUX

```
import React from 'react'
import {connect} from 'react-redux'

const Authorized = {roles, allowedRoles, requireAll, children} => {

  const matchesRoles = required => {
    roles.indexOf(required) >= 0;
  }

  const show = requireAll ? allowedRoles.every(matchesRoles)
                          : allowedRoles.some(matchesRoles);

  return show ? children : null
}


//..

const mapStateToProps = (state) => {
  return {
    allowedRoles: state.user.roles
  }
};

Authorized = connect(mapStateToProps, null)(RequiresRole);

export default Authorized;
```

ROLE-BASED AUTHORIZATION & ROUTING

- Advanced Approach
  - Encapsulate role evaluation in a **Higher Order Component**
- Slightly more complex to implement
- More reusable code

HIGHER ORDER COMPONENTS

- A Higher Order Component is "*… a function that takes a component and returns a new component.*" (React docs)
- Used to avoid cross-cutting concerns in components (e.g., accessing external data stores, such as Redux)

HIGHER ORDER COMPONENTS

```
// This function takes a component...
function withExternalLogic(WrappedComponent, inputFunction) {

  // ...and returns another component...
  return class ExternalLogic extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
        //set up some state
      };
    }

    componentDidMount() {
      //perform external logic by calling `inputFunction()`
    }

    render() {
      // ... and renders the wrapped component with the fresh data!
      return <WrappedComponent data={this.state.data} {...this.props} />;
    }
  };
}
```

```
const MyComponentWithExtLogic = withExternalLogic(
  MyComponent,
  (external) => external.doSomething()
);
```

https://reactjs.org/docs/higher-order-components.html

HIGHER ORDER COMPONENTS

```
import React, {Component} from 'react';

const Authorized = (allowedRoles, user) => {

   return (WrappedComponent) => {

     return class WithAuthorization extends Component {
       render() {
         const {role} = user;
         if (allowedRoles.includes(role)) {
           return <WrappedComponent {...this.props} />
         } else {
           return <h1>No page for you!</h1>
         }
       }
     }
   }
}

export default Authorized;
```

https://reactjs.org/docs/higher-order-components.html

https://hackernoon.com/role-based-authorization-in-react-c70bb7641db4

HIGHER ORDER COMPONENTS

```
import Authorized from "./Authorized";

render() {
  const {user, loggedIn} = this.state;
  const withAdminRole = Authorized(['admin'], user);
  const withUserRole = Authorized(['user', 'admin'], user);
  return (
    <Router>
          <Route path="/restricted" component={withUserRole(Restricted)}/>
          <Route path="/admin" component={withAdminRole(Admin)}/>
    </Router>
  );
```

https://reactjs.org/docs/higher-order-components.html

HIGHER ORDER COMPONENTS

```
import {withAdminRole, withUserRole} from “./AuthorizedRoles";

render() {

  Restricted = withUserRole(Restricted);
  Admin = withAdminRole(Admin);

  return (
    <Router>
          <Route path="/restricted" component={Restricted}/>
          <Route path="/admin" component={Admin}/>
    </Router>
  );
```

https://reactjs.org/docs/higher-order-components.html

API AUTHENTICATION

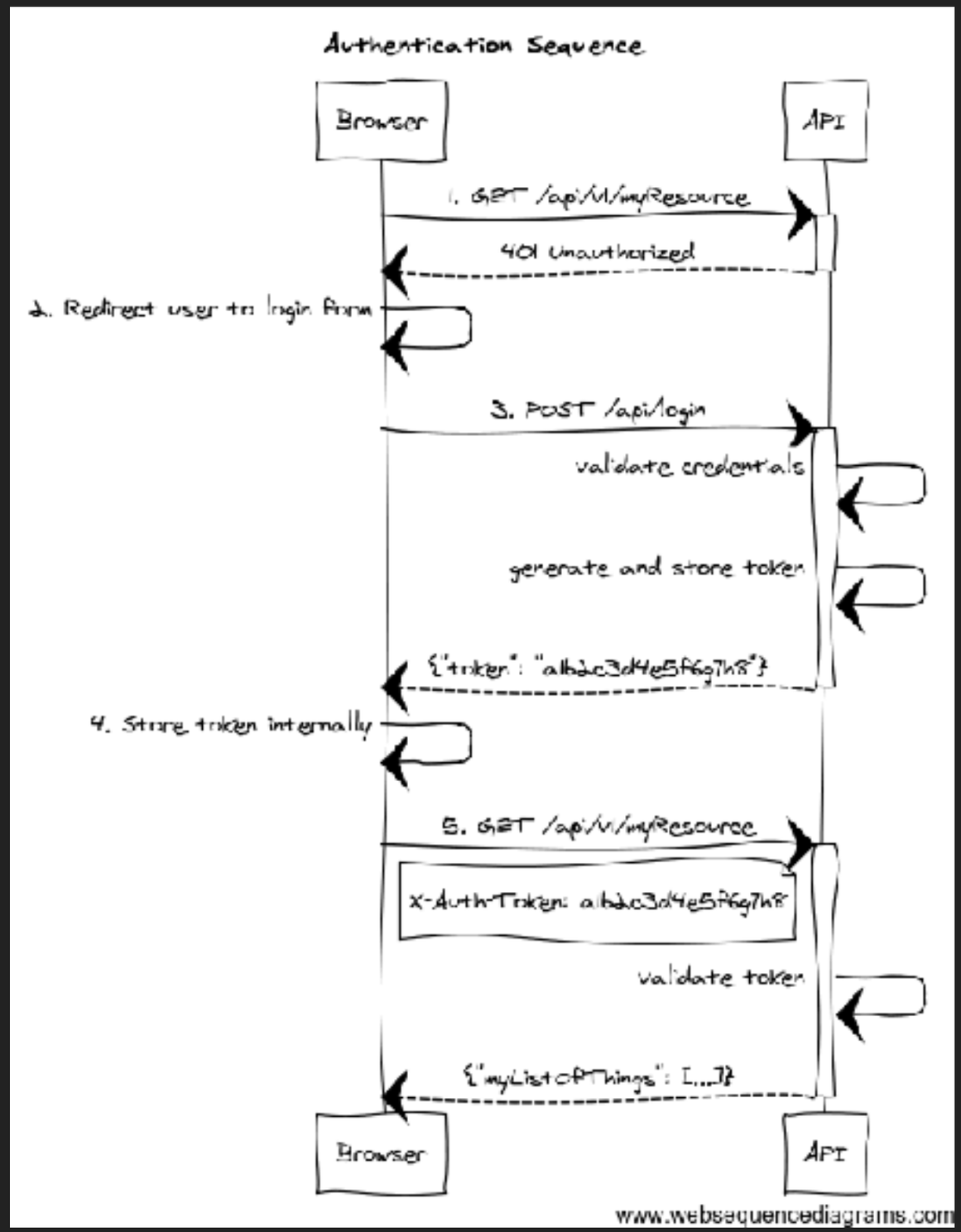
- Once data makes it to the UI, it's too late to secure
- Authorization needs to be checked at every endpoint
- In a microservice architecture, each service boundary should be secured
- Always use HTTPS

STATELESS AUTHENTICATION

- Stateless authentication typically relies on encrypting session details in a token
- Token is provided to client upon successful login
- Token can contain its own expiration date, user details, roles (scopes)
- Client includes token in API requests
- Server can verify client's identity/authorization via token

STATELESS AUTHENTICATION

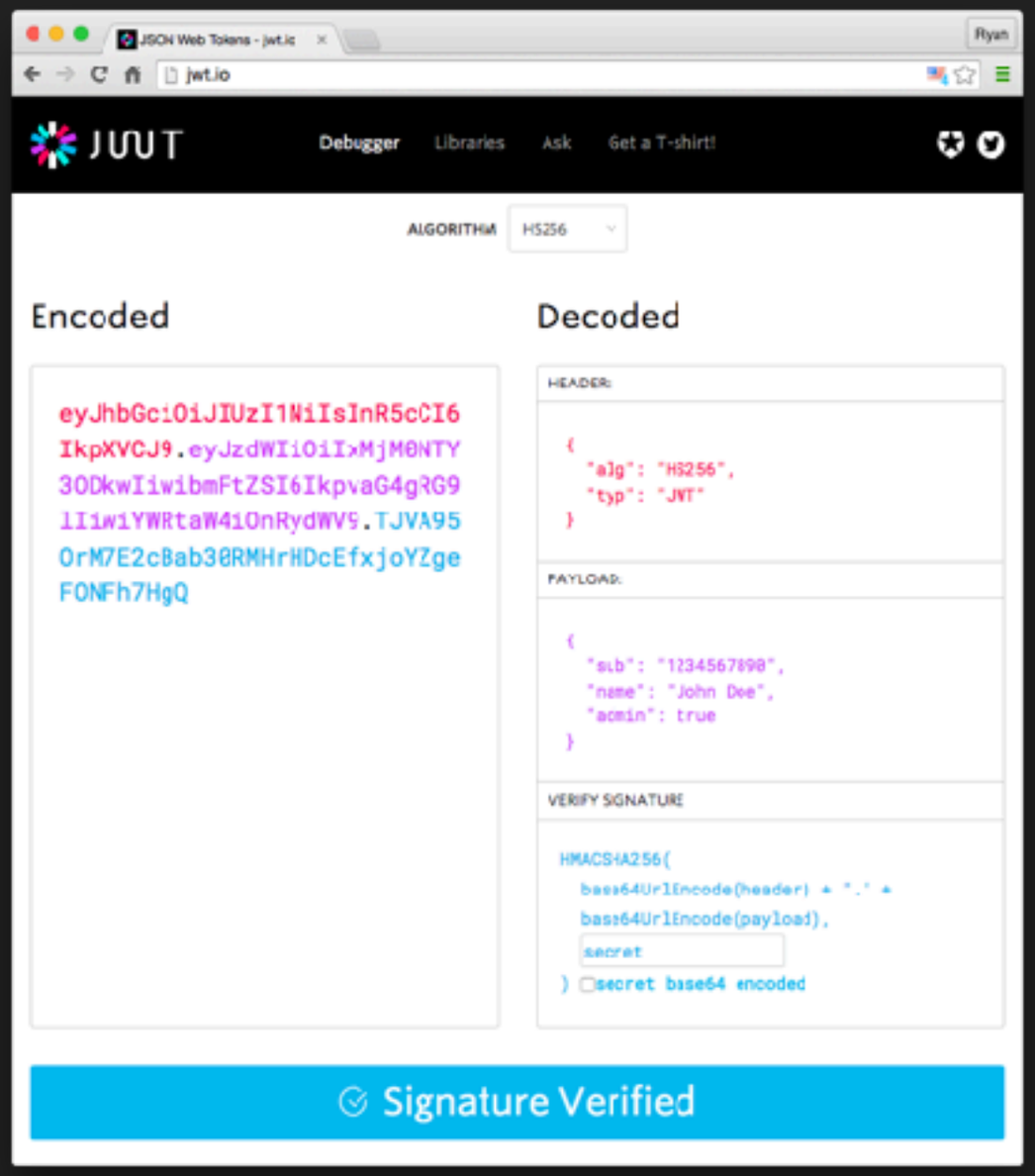
- Unauthorized request is made to API
- Responds with 401
- Client POSTs to login endpoint
- Responds with authorization token
- Token included in subsequent request
- Responds with resource



JSON WEB TOKENS

- https://jwt.io
- Open, industry-standard method for representing claims securely between two parties
- Typically consist of a header, payload, and signature

JSON WEB TOKENS



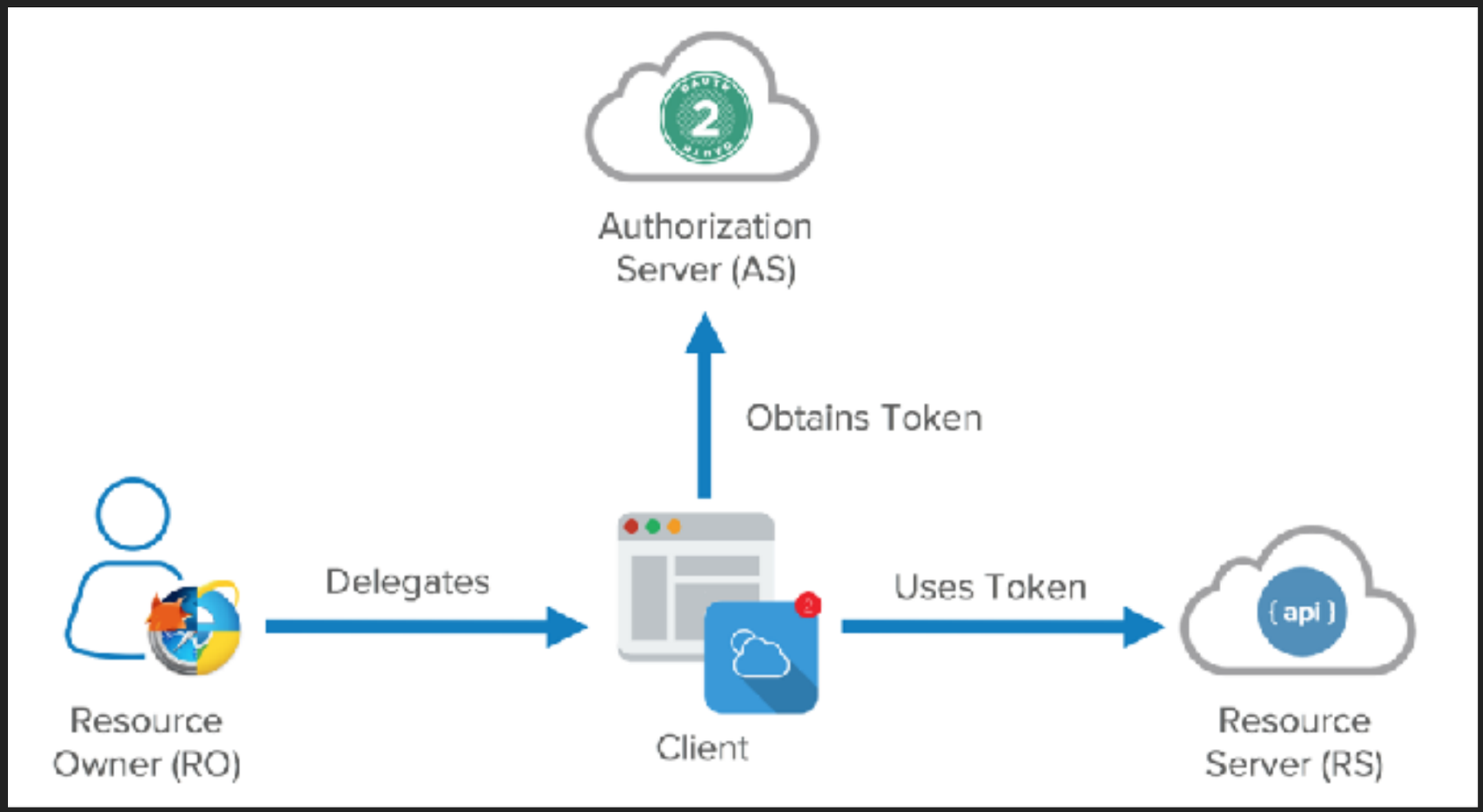
https://jwt.io/introduction/

STATELESS AUTH WITH JWT

DEMO

OAUTH2 & THIRD-PARTY AUTHENTICATION

- OAuth2 is a standard for authentication and secured resource access w/o sharing credentials



https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth

OAUTH2 & THIRD-PARTY AUTHENTICATION

- Many third party authentication providers available
- Doesn't usually make sense to stand up your own
- https://hackernoon.com/authentication-as-a-service-an-honest-review-of-auth0-315277abcba1
- https://hackernoon.com/dev-rant-stop-reinventing-user-auth-1193b138772

AUTH0

- https://auth0.com/docs/quickstart/spa/react/01-login
- https://auth0.com/blog/reactjs-authentication-tutorial

OKTA

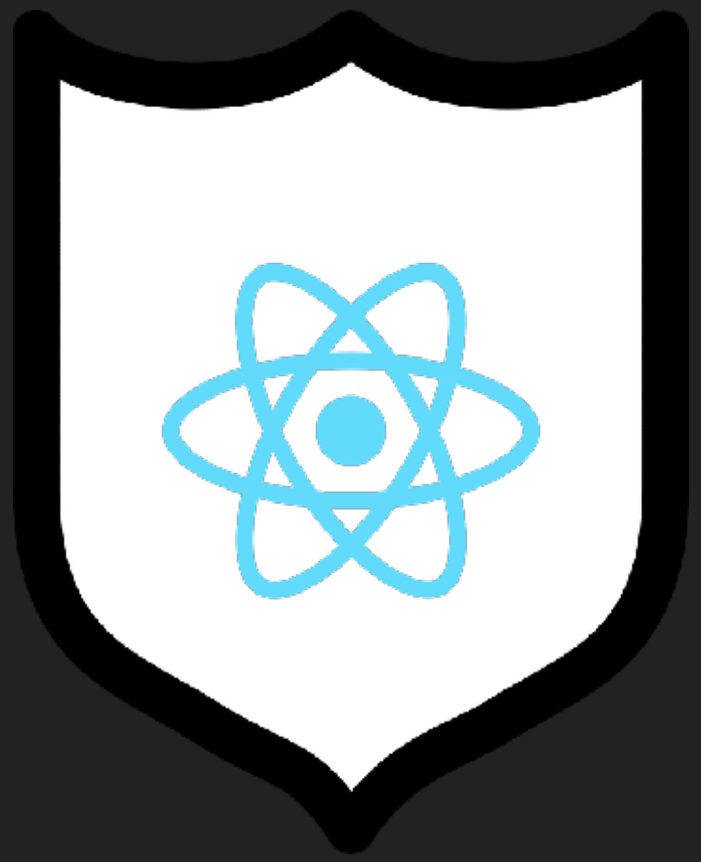
- https://developer.okta.com/blog/2017/03/30/react-okta-sign-in-widget
- https://developer.okta.com/blog/2017/12/06/bootiful-development-with-spring-boot-and-react

SUMMARY

- React is largely very safe for client-side attacks
- Always validate user input
- Plan routing strategies
- Secure your API first
- Don’t re-invent the wheel - leverage existing platforms for authentication/authorization
- Keep checking behind your back

LINKS

- http://www.jamesward.com/2013/05/13/securing-single-page-apps-and-rest-services
- http://guides.grails.org/react-spring-security/guide
- https://hackernoon.com/authentication-as-a-service-an-honest-review-of-auth0-315277abcba1
- https://hackernoon.com/dev-rant-stop-reinventing-user-auth-1193b138772
- https://medium.com/dailyjs/exploiting-script-injection-flaws-in-reactjs-883fb1fe36c1
- https://stormpath.com/blog/where-to-store-your-jwts-cookies-vs-html5-web-storage



THANK YOU

Twitter: **@ZacharyAKlein**. Github: **@ZacharyKlein**. Email: kleinz@objectcomputing.com

https://github.com/ZacharyKlein/shields-up-securing-react-apps