React.js Security Guide: Threats, Vulnerabilities, and Ways to Fix Them

Carefully built and well-functioning security systems in web applications help businesses to stand strong and establish trusting relationships with their customers. Security systems prevent sensitive data leaks, help companies maintain their reputation and minimize losses.

Unfortunately, some businesses overlook the importance of security in their apps and run into long-lasting negative consequences. According to Check Point Software's Security Report, "87% of organizations have experienced an attempted exploit of an already-known, existing vulnerability" in 2020.

Many companies try to minimize security weaknesses. For this, they search for reliable IT companies with extensive app development expertise and use the most effective and time-tested technologies for building their web applications. One of these technologies is React.js.

In this article, we explain why to choose React.js for building your web application and how to secure React apps.

React.js Popularity and Why to Choose It

React.js is the second most popular JavaScript framework for web development according to StackOverflow's latest developer survey. It's a powerful solution for building dynamic and interactive user interfaces. With React.js web developers can create large web applications with fast performance and simple scalability.

The main features that make React.js a preferred technology for web development are:

- reusable components for consistent app look and facilitated app development;
- virtual DOM which allows fast web page rendering and improves app performance;
- high abstraction layer that makes app development simple even for React.js beginners;
- Flux – a Facebook architecture for simple data flow management in React.js apps;
- Redux – a state container that organizes React.js apps, making them consistent and easy to test solutions;
- a wide range of development tools, extensions, and compatible libraries.

Stackshare states that React.js has 168.2K stars on GitHub with over 33.8K GitHub forks. More than 9600 companies use React.js in their tech stacks, including Uber, Facebook, Netflix, Airbnb, Pinterest, Instagram, and many others.

Why is it Important to Follow React.js Security

Any business web app involves extensive data exchange and connection to various sources. This helps businesses compete in the market and effectively provide services to their customers. On the downside, this high connectivity makes web apps prone to a wide range of security flaws and data breaches.

When building their web apps with React.js, web developers have to keep in mind that React.js has no default security settings. Therefore, they need to know how to handle the most widespread security issues that may appear in web applications.

The most common React.js vulnerabilities include:

- dangerous URL schemes;
- broken authentication;
- Server-side rendering;
- SQL Injections;
- Zip Slip;
- Cross-Site Scripting (XSS).

React.js Security Vulnerabilities and Solutions

Let’s have a look at the most common React.js vulnerabilities and best practices to prevent them.

Dangerous URL Schemes

Links to other resources become dangerous when hackers add malicious code that starts with JavaScript to URLs. When a user clicks on a link, they activate the script in a browser. React.js app security doesn’t prevent the use of links without “HTTP:” or “HTTPS:” protocols and has no features to prevent potential threats.

To avoid JavaScript in their links, web developers can:

- make links start with whitelisted protocol and display HTML entities on a browser;
- eliminate URL input from the users e.g. use a YouTube video ID instead of its link;
- implement third-party tools to sanitize all the input links.

Broken Authentication

The insecure connection between the web client and the server-side leads to broken authentication and user authorization issues. Hackers can interfere with the authentication and authorization processes and spoil user account data, passwords, session tokens, and other entities.

The most widespread React.js security risk factors related to broken authentication include:

- exposing session IDs in the URL;
- simple or easy-to-predict login credentials;
- session IDs and passwords transmitted with unencrypted connections;
- session fixation attacks;
- sessions that don't get invalidated after a user logs out, and others.

To protect the HTTP basic authentication protocols, web developers need to:

- determine if the domain "WWW" header has a real attribute which helps to avoid mismatches in user IDs and their passwords;
- use proper authentication methods, e.g. make sure that a web app responds with a 401 status error page in case of failed authentication;
- implement multi-factor authentication;
- introduce cloud-native authentication, e.g. Google Cloud Identity Platform or Azure Active Directory;
- implement password checks for strengths and weaknesses.

Server-Side Rendering

Many web apps utilize server-side rendering when they display their web pages and content to the users. Server-side rendering has many advantages. It improves app performance, makes webpage loading faster, and ensures consistent SEO performance. Although, this type of page rendering can involve some security challenges.

When rendering an initial state of a web page with Redux, web developers can generate a document variable from a JSON string looking like this:

```
<script>window.__STATE__ = ${JSON.stringify({ data })}</script>
```

The JSON.stringify() can be a risky method as it converts any given data into a string and displays it on a web page. As a result, attackers can insert some malicious code inside the JSON string and eventually take control of a web app or its data.

To tackle this vulnerability, web developers need to:

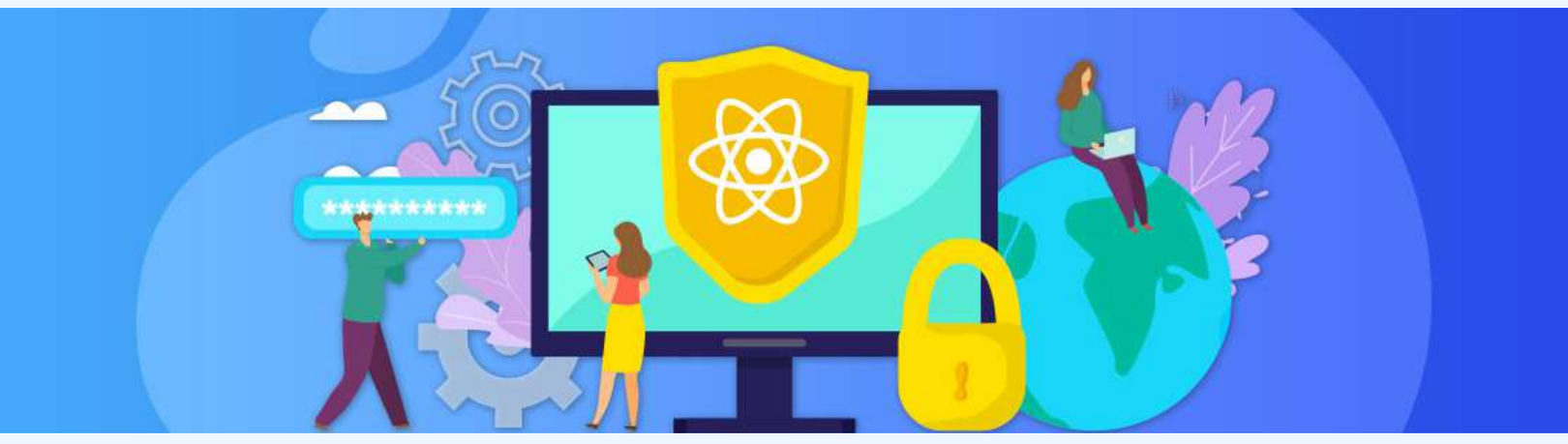
- often perform code reviews and check the data that appears in JSON.stringify();
- utilize serialize-JavaScript NPM module to avoid rendering JSON.

SQL Injections

This type of attack is aimed at the app's databases. Attackers inject malicious SQL code into a database and receive access to the stored information. By gaining the admin credentials they can edit, delete, or create new records.

There are various types of SQL injections such as error-based, time-based, and logical-error-based, and React apps are vulnerable to all of them. Therefore to minimize the number of SQL attacks, web developers can:

- implement whitelists to filter all kinds of inputs;
- apply the principle of least privilege to all the accounts when a web app can use only one statement (SELECT, UPDATE, INSERT, or DELETE) for certain operations;
- assign the database roles to different accounts;
- use vulnerability scanners;
- validate all API functions according to their API schemas.



Zip Slip

Zip Slip vulnerability happens when app users submit reduced in size zip files in React.js apps. When a web app decompresses such files it restores them to their initial size. Some of these files can include some hacker codes that provide attackers with access to the system.

Attackers can use this React.js vulnerability and overwrite the files responsible for app configurations and information storage. With Zip Slip, they can break into the app's system on the client or server-side layers.

To eliminate the possibility of Zip Slip hacker attack, web developers can:

- make sure that all the files in their web app have standard names and no special characters;
- generate new names for the zip files uploaded in the system.

Cross-Site Scripting

Cross-site scripting (XSS) is a serious vulnerability that appears when attackers manage to trick a web app into launching a JavaScript code. Cross-site scripting can be divided into two forms:

- Reflected cross-site scripting

It happens when a web app receives malicious JavaScript code from a trustworthy source in the form of an HTTP request. And this code is processed by the app's browser.

As a result, the malicious script gains access to cookies, session tokens, or other sensitive data from the browser. These scripts can also rewrite the HTML page content or execute unwanted actions that a user can perform.

- Stored cross-site scripting

Stored XSS comes from an untrusted source. A web app stores malicious content on a server or a database for later. When a user requests the stored data the script starts working on the server-side generating data and sending it to the user.

To defend their React.js-based apps from XSS attacks web developers can:

- disable or avoid writing code where attackers can potentially insert instructions for running malicious scripts, e.g. exclude in HTML elements like <script>, <object>, <embed>, and <link>;
- implement snippet libraries such as ES7 React, Redux, and others;
- use {} for default data binding – this way the React framework will escape values automatically;
- utilize Web Application Firewall in the app's code;
- use special tools to scan built apps for XSS vulnerabilities.

Summary on React.js Security

Many businesses with a worldwide reputation search for React development companies to build their web solutions. React.js is a robust and effective framework that helps to create fast, high performance, and rich in features web applications.

However, like any software development framework, React.js isn't resistant to hacking 100%. Yet, web developers can introduce some measures to reduce the number of malicious attacks to a minimum.

Here is a quick summary of the most useful practices web developers can follow to build effective security systems in their React.js application:

- use whitelists to filter all the app inputs and perform frequent React code audits for potential vulnerabilities;
- inspect app code and features for the possibility to insert malicious parts of code, like URLs or HTML elements;
- improve app protection with vulnerability scanners, serialize-JavaScript NPM module, and Web Application Firewall, and others;
- utilize proper authentication methods and technologies;
- inspect databases for possible SQL injections and properly assign roles to various accounts;
- validate all API functions according to their API schemas;
- rename downloaded zip files;
- use {} for default data binding to prevent XSS attacks.

Feel free to contact us and ask any questions: info@scand.com