# JavaScript Security Best Practices


RAJA MSR

**Full Stack Engineer**


rajamsr.com

# 01 Bulletproof Your Code: JavaScript Input Validation

Learn how to validate and sanitize user inputs effectively to prevent common vulnerabilities like Cross-Site Scripting (XSS) attacks.

```javascript
// Validate user input (e.g., email)
function isValidEmail(email) {
    const emailRegex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
    return emailRegex.test(email);
}
```

## Common JavaScript Input Validation Techniques

### Regular Expressions
Widely used for pattern matching and validation.

### Conditional Statements
Utilize if statements to check input against predefined conditions.

### HTML5 Input Types and Attributes
HTML5 introduces input types and attributes like required, pattern, min, and max

### Third-party Libraries
Libraries like Validate.js, jQuery Validation Plugin, or FormValidation offer pre-built validation functionalities.

### Challenges and Considerations
Ensure validation techniques work consistently across different browsers.

# 02 Say No to eval(): Secure JavaScript Execution

Explore safer alternatives to the eval() function, which can lead to code injection vulnerabilities.

```
// Avoid using eval() for dynamic code execution
const dynamicCode = 'console.log("Hello, World!");';
const saferFunction = new Function(dynamicCode);
saferFunction();
```

- JavaScript eval() executes arbitrary code, making applications vulnerable to code injection attacks.
- Attackers can exploit this to execute malicious scripts, leading to data theft or unauthorized system access.

# 03 Fortify Your Web Apps: Cross-Site Scripting (XSS) Defense

Discover how to implement content security policies (CSP) to prevent unauthorized script execution.

```
<meta http-equiv="Content-Security-Policy" content="script-src 'self'">
```

| Key Directives and Their Usage | |
|---|---|
| **default-src** | Specifies the default sources for loading content, such as scripts, stylesheets, images, fonts, and frames. |
| **script-src** | Controls the sources from which scripts can be executed. |
| **style-src** | Defines the allowed sources for loading stylesheets. |
| **img-src** | Specifies the sources from which images can be loaded. |
| **connect-src** | Determines the valid sources for network connections, such as AJAX requests, WebSocket connections |

# 04 CSRF-Proof Your Forms: JavaScript Anti-CSRF Measures

Learn how to generate and validate anti-CSRF tokens to protect against Cross-Site Request Forgery (CSRF) attacks.

```
const csrf = require('csurf');
app.use(csrf());
```

| Generate CSRF Token | Include CSRF Token in Requests | Validate CSRF Token on the Server |
| --- | --- | --- |
| Generate a unique CSRF token for each session. | Include the CSRF token in each request sent to the server. | Validate the CSRF token on the server-side for each incoming request. |

05

# DOMPurify: Your Shield Against Malicious Inputs

Dive into DOMPurify, a library that sanitizes user inputs and neutralizes potential code threats.

```
// Example user input (potentially unsafe)
var userInput = '<img src="javascript:alert(\'XSS Attack!\')">';

// Render potentially unsafe HTML
document.getElementById('unsafe-html').innerHTML = userInput;

// Use DOMPurify to sanitize the HTML
var safeHtml = DOMPurify.sanitize(userInput);

// Render safe HTML
document.getElementById('safe-html').innerHTML = safeHtml;
```

# 06 JSON.parse() vs. eval(): Battle of the Dynamic Parsers

Compare the security benefits of using JSON.parse() over eval() for parsing JSON data.

```
var jsonString = '{"name": "John", "age": 30}';
var parsedObject = JSON.parse(jsonString);
console.log(parsedObject);
// Output: { name: 'John', age: 30 }
```

**JSON.parse()**

Safer than eval() because it only evaluates JSON strings and does not execute arbitrary JavaScript code. This makes it immune to code injection attacks.

```
var codeString = '2 + 2';
var result = eval(codeString);
console.log(result);
// Output: 4
```

**eval()**

It should be avoided whenever possible. If untrusted data is passed to eval(), it can execute malicious code, leading to security vulnerabilities such as XSS attacks.

# 07 Secure Dependencies: Audit Your JavaScript Packages

Explore tools like npm audit to keep your project dependencies up-to-date and secure.

Run this command to analze your project's dependencies and display any security vulnerabilities found.

*npm audit*

To automatically install compatible updates to resolve the vulnerabilities, you can run:

*npm audit fix*

# 08 Safeguarding Your Scripts: Subresource Integrity (SRI)

Learn how to add SRI hashes to external scripts for enhanced security.

- SRI is implemented by adding the **integrity** attribute to the **<script>** or **<link>** tag that references the external resource.
- The integrity attribute value consists of the hash algorithm (e.g., **sha256**), followed by a hyphen, and then the base64-encoded hash of the resource's content.

```
<body>
  <!-- Example of an external script with SRI hash -->
  <script src="https://example.com/external-script.js"
  integrity="sha384-XmV/M0cucqEbc+HlJjuIwDl7GHwbNhZeNvDJfTBI/1Ot5W+2t//ArlBv6nqKI+a5"
  crossorigin="anonymous"></script>

  <!-- Explanation:
       - src: URL of the external script
       - integrity: SRI hash generated for the script content
       - crossorigin: Specifies how the browser should handle the
     request if it is loaded from a different origin -->
</body>
```

**Integrity Verification**

**Hash-Based Integrity Check**

**Protection Against Code Injection Attacks**

**Multiple Hashes**

# 09 JavaScript Linting: Your First Line of Defense

Embrace linting tools to catch potential security issues early in your code.

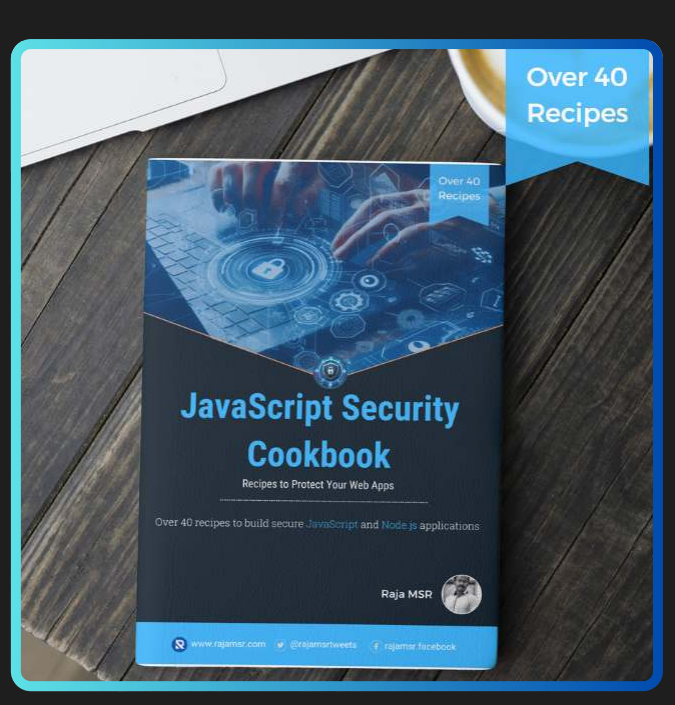
```
// my-script.js
function calculateTotal(price, quantity) {
    let total = price * quantity;
    return total;
}

let itemPrice = 10;
let itemCount = 5;
let totalPrice = calculateTotal(itemPrice, itemCount);

console.log(totalPrice);

// Run eslint
npx eslint my-script.js
```

In this example, linting might detect that the itemPrice variable is declared but never used, prompting you to either remove it or use it appropriately.

# Download “JavaScript Security Cookbook” for FREE!

Security measures are vital when writing code in JavaScript and Node JS. Don't let hackers ruin your awesome work! 😎

- 40+ Recipes
- Infographics
- Source code
- 63 Pages eBook (PDF)




**Download**

https://onlinemsr.gumroad.com/l/javascript-security-cookbook

# Thank You!

What’s your favorite tool or technique for securing JavaScript applications? Share your experiences below!👇

rajamsr

rajamsrtweets

rajamsr.facebook

www.rajamsr.com

I would appreciate it if you could repost this!