JS

5 best Security Practices for JavaScript applications

codewithsloba.com

Swipe →

Minimize Attack Surface

Reduce the amount of code exposed to the client-side to limit potential vulnerabilities.

```
// Client-side (simplified example)
function submitData(data) {
  fetch('/api/submit', {
    method: 'POST',
    body: JSON.stringify(data)
  })
  .then(response => response.json())
  .then(responseData => {
    console.log("Data submitted successfully:", responseData);
  })
  .catch(error => {
    console.error("Error submitting data:", error);
  });
}
```

Move sensitive logic and data processing to the server-side whenever possible. Use server-side rendering for critical parts of your application.

Swipe →

Use Secure Random Numbers

Employ cryptographically secure random number generation for tasks like **session tokens or cryptography**.

```
const crypto = require('crypto');

function generateRandomToken() {
  return crypto.randomBytes(32).toString('hex');
}

const token = generateRandomToken();
console.log("Secure random token:", token);
```

Avoid using browser-based Math.random() for security-sensitive purposes. Utilize **libraries like crypto** (Node.js) or secure alternatives in your environment.

Swipe →

JS

Implement CSRF Protection

Prevent **unauthorized form submissions** using CSRF tokens.

```
// Client-side (simplified example)
const csrfToken =
document.getElementById('csrfToken').value;
```

Assuming hidden input with token

```
fetch('/submit-form', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'X-CSRF-TOKEN': csrfToken
  },
  body: JSON.stringify({ data: 'some data' })
})
// ... handle response
```

Generate a **unique CSRF token** for each user session. Include the token in forms and **verify it on the server-side** to prevent CSRF attacks.

Swipe →

Avoid Inline Event Handlers

Use separate functions for event listeners to **prevent XSS from injected code.**

```
<button id="myButton">Click me</button>

<script>
  const button = document.getElementById("myButton");
  button.addEventListener("click", handleClick);

  function handleClick() {
    console.log("Button clicked!");
  }
</script>
```

Instead of using onclick attributes in HTML, **attach event listeners** using JavaScript functions.

Swipe →

Use Content Security Policy (CSP)

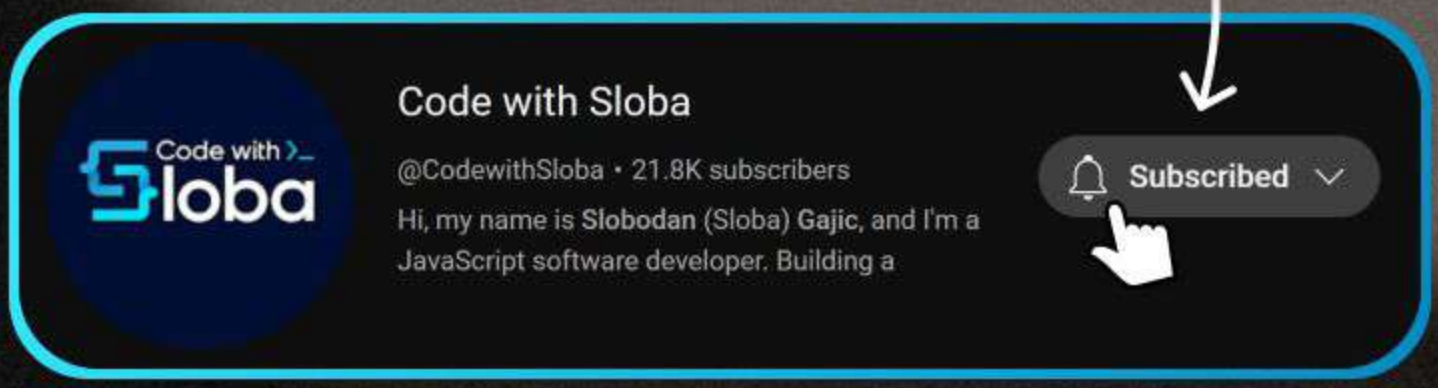
Define allowed sources for scripts, styles, images, etc., to mitigate XSS attacks.

```
const express = require('express');
const app = express();

app.use((req, res, next) => {
  res.setHeader("Content-Security-Policy",
"script-src 'self'");
  next();
});

// ... rest of your application code
```

Set a CSP header on your server to restrict where your application can load resources from.

Swipe →

Subscribe on YouTube to learn more



Code with Sloba

Educator • Senior JavaScript Developer • YouTuber