

Code Review

Here we will see a basic login signup page code for a bottle company and review its vulnerabilities and provide recommendations for secure practices



The Code

```
const users = [
  { username: 'admin', password: 'password123' },
  { username: 'manager', password: 'bottleco2023' }
];
function login(username, password) {
  for (let user of users) {
    if (user.username === username && user.password === password) {
      console.log('Login successful');
      return true;
    }
  }
  console.log('Login failed');
  return false;
}
let username = document.getElementById('username').value;
let password = document.getElementById('password').value;
if (login(username, password)) {
  window.location = 'admin-dashboard.html';
} else {
  alert('Invalid credentials');
}
```

Understanding the vulnerabilities



```
const users = [  
  { username: 'admin', password: 'password123' },  
  { username: 'manager', password: 'bottleco2023' }  
];
```

Plaintext Passwords

- Passwords are stored in plain text, visible to anyone who can access this code.
- If a malicious actor gains access to this data, they immediately know user passwords.
- Example scenario: Imagine an employee with access to the codebase. They could easily view and misuse these passwords.

Improved Code

Store the hashedPassword instead of the plain text password.

```
const bcrypt = require('bcrypt');  
const hashedPassword = await bcrypt.hash('password123', 10);
```

```
function login(username, password) {  
  for (let user of users) {  
    if (user.username === username && user.password === password) {  
      console.log('Login successful');  
      return true;  
    }  
  }  
  console.log('Login failed');  
  return false;  
}
```

Client-side Authentication

- Authentication logic is entirely in the client-side JavaScript.
- Easily bypassed by modifying the JavaScript in the browser.

Improved Code

- Using POST method for the vulnerability

```
const response = await fetch('/api/login', {  
  method: 'POST',  
  headers: { 'Content-Type': 'application/json' },  
  body: JSON.stringify({ username, password }),  
});  
app.post('/api/login', async (req, res) => {  
});
```

```
if (user.username === username && user.password === password) {  
    console.log('Login successful');  
    return true;  
}
```

Timing Attacks

- The function returns immediately upon finding a match.
- The time taken to return can leak information about valid usernames.


```
const timingSafeEqual = require('crypto').timingSafeEqual;
const usernameMatch = timingSafeEqual(Buffer.from(user.username), Buffer.from(username));
const passwordMatch = timingSafeEqual(Buffer.from(user.password), Buffer.from(password));
if (usernameMatch && passwordMatch) {
}
```

Improved Code

- Use constant-time comparison

Lack of Input Validation

- No validation is performed on the input.
- Could lead to injection attacks or unexpected behavior.

```
let username = document.getElementById('username').value;  
let password = document.getElementById('password').value;  
  
if (login(username, password)) {
```

Improved Code

```
const username = document.getElementById('username').value.trim();
const password = document.getElementById('password').value;

if (!username || !password) {
  alert('Please enter both username and password');
  return;
}

if (username.length > 50 || password.length > 100) {
  alert('Username or password too long');
  return;
}
```

Direct Object References

```
if (login(username, password)) {  
    window.location = 'admin-dashboard.html';  
}
```

- Directly references a page name in the client-side code.
- Could allow an attacker to enumerate or access restricted pages.

Improved Code

```
if (response.ok) {  
  const { redirectUrl } = await response.json();  
  window.location.href = redirectUrl;  
}  
if (authenticatedUser.role === 'admin') {  
  res.json({ redirectUrl: '/admin-dashboard' });  
} else {  
  res.json({ redirectUrl: '/user-dashboard' });  
}
```

- Server returns a token or session ID to the server side.

Final Improved Secure Code

```
const bcrypt = require('bcrypt');
async function login(username, password) {
  try {
    const user = await fetchUserFromDatabase(username);
    if (!user) {
      console.log('Login failed');
      return false;
    }
    const match = await bcrypt.compare(password, user.hashPassword);
    if (match) {
      console.log('Login successful');
      return true;
    } else {
      console.log('Login failed');
      return false;
    }
  } catch (error) {
    console.error('Login error:', error);
    return false;
  }
}

document.getElementById('loginForm').addEventListener('submit', async (e) => {
  e.preventDefault();
  const username = document.getElementById('username').value;
  const password = document.getElementById('password').value;
  if (!username || !password) {
    alert('Please enter both username and password');
    return;
  }
  const response = await fetch('/api/login', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ username, password }),
  });
  if (response.ok) {
    window.location.href = '/admin-dashboard';
  } else {
    alert('Invalid credentials');
  }
});
```

Secure Coding Practices



Thank You