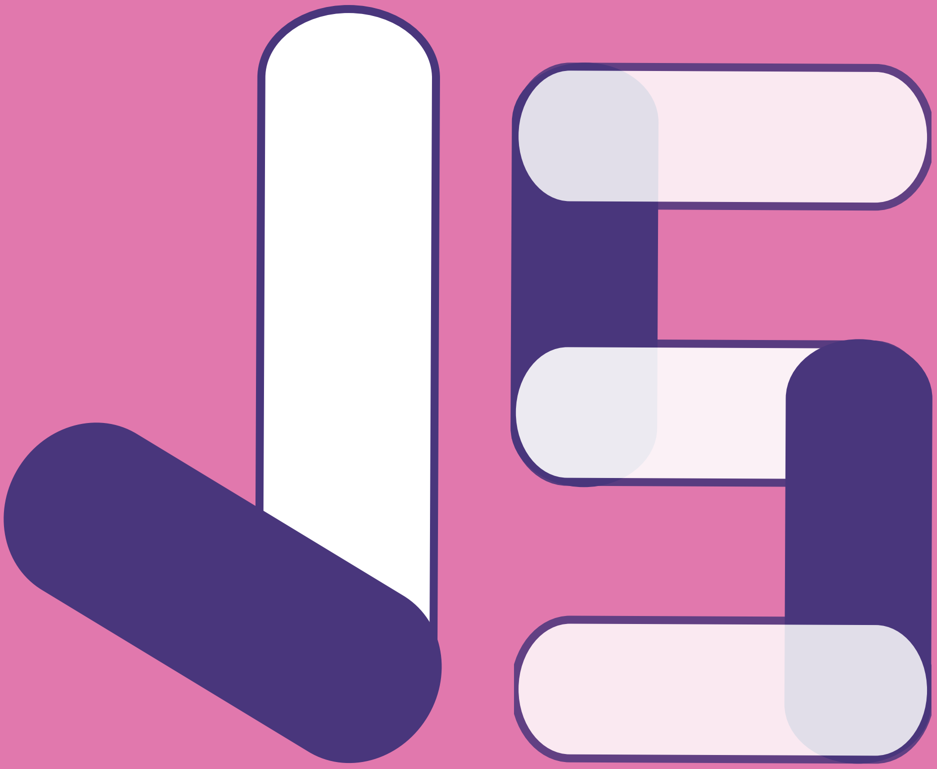
A Security Guide for JavaScript Developers



kiuwan
An Idera, Inc. Company

INDEX

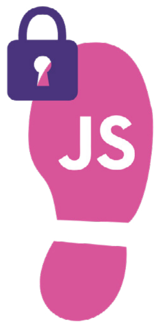
Overview

JavaScript developers have seen multiple issues, threats, and trends over the past year. Software security is a top priority across all industries since so much can go wrong. Programming errors and insecure user inputs, as well as a multitude of cyberattacks, can all sink a deployment if the developers do not employ strict security processes, penetration testing, and awareness of security issues involving JavaScript.

Regular, automated security scans can identify and remediate some vulnerabilities before damage occurs, but building in security as you go and keeping your applications updated goes a lot further in maintaining a shield between your code and malicious actors. The old adage, "An ounce of prevention is worth a pound of cure," applies.

This post looks at the elements that make up the JavaScript security footprint, highlighting some of the most common security issues in JavaScript projects. We talk about the JavaScript packages developers use and the impact of those packages on security, concentrating on the most vulnerable packages in JavaScript containers and the accompanying risks. Then we provide targeted recommendations to mitigate JavaScript security risks and produce secure code.

The JavaScript Project *Security* Footprint



JavaScript project security involves three risky elements in programming: direct and indirect dependencies and vulnerabilities.

- Direct dependencies are packages you include in your project, like package.json, which keeps a list of packages installed in your project. These build up rapidly.

- Indirect dependencies are packages that the project does not use directly but are used by a direct dependency. For example, if the application uses package 1 and package 1 uses package 2, then the application indirectly depends on package 2, which, if vulnerable, impacts the application.
- Vulnerabilities in JavaScript are most commonly found in vectors containing malicious script, tricking users into performing unintended actions, stealing a user's established session data or data from the browser's localStorage, and exploiting vulnerable source code in web applications.

Package.json is so popular because it can list dependencies, specify the version of a package your project can use with semantic versioning rules, and help you develop a reproducible, highly shareable build.

You can visualize your dependencies and try deduplication to simplify the dependency tree.

JavaScript is everywhere, in client-side scripting, server-side scripting, and device programming. It has many moving parts, including indirect dependencies that can hide vulnerabilities at a slight distance. Dependencies pile up quickly. For example, using Gatsby.js to build a blog adds **190,000 extra dependencies**, and there's no way to scan them all without an automated tool.

Attackers use these weaknesses to readily access security cameras, microphones, phone calls, and searches. Building in risk mitigation instead of layering it on after the fact helps immensely in developing a secure interactive web application.

The Most ***Common Security Issues*** in JavaScript Projects

Bad actors exploit any vulnerability they can find in any way they can. Code injections, scripting, and entry through insecure third-party applications are some of their favorites. The bad news is that more than one of these can be used in an attack.

SQL Injections

In SQL injection, data enters an application via an untrustworthy source and is used to construct a SQL query dynamically. It can result in data loss or corruption, denial of access, lack of accountability, and a complete host takeover.

Never trust user input provided in an SQL query. And don't assign values from the front-end to the database query without the appropriate care.

Encoded Scripting and Scripting Language Exploits

Encoded scripting is a method of supplying data featuring specific special characters for execution. The characters are used in the underlying JavaScript pages, and when the browser should display values or characters, instead, it mis-interprets them as part of the code.

Inserting or enabling coded scripting allows hackers to break out of text fields (like on forms) and provide additional browser-side codes that the hackers can trigger.

Cross-Site Forgery (CSRF)

Cross-site forgery, also known as session riding or one-click attacks, takes advantage of trusted users logged into the application. The attacker accesses the authorized user account using information from session cookies. Then the attacker takes action under the user's identity without them knowing or being involved.

Third-Party Security Vulnerabilities

Any time you use third-party code, packages, or applications, you run the risk that one or more of them are not as secure as they should be. They are corrupted indirect dependencies threatening the security of your application.

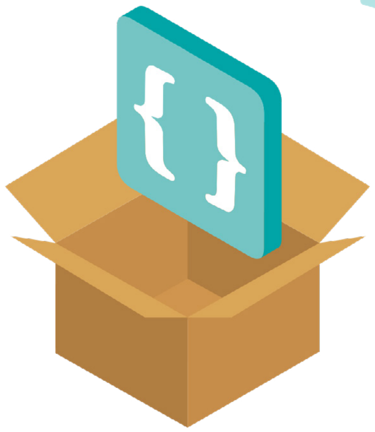
All third-party libraries and tools are vulnerable to JavaScript exploits. Some libraries and tools are created and monitored by large corporations that remediate issues and follow best practices for JavaScript security. However, libraries developed and maintained by independent parties, such as open-source code, don't have the same level of monitoring and maintenance due to a lack of resources for auditing and updating publicly available code.

Remote Code Execution

Hackers look at JavaScript like an unlocked candy store because it's so easy to send JavaScript code over a network connection. There is no built-in code to protect against hackers. Developers must rely on the security features in programming languages to shore up the defenses.

JavaScript Packages and *Vulnerabilities*

A package is a set of code made publicly available as a versioned downloadable library that can streamline and standardize development. Each package can contain a single file or many files of code to help you add functionality to your website. The package can contain a library (set of modules) or an executable. These packages usually reside on GitHub or NPM, which are not the same things.



- **GitHub** is a repository with a version control system. It's more like an online backup or host for your code.
- **NPM** is a package manager and package repository for JavaScript code. You use a command-line interface to install, update, and remove packages.

Libraries perform specific functions. For example:

- Data visualization. Chart.js, Apexcharts, Angolia Places
- DOM (Document Object Model) manipulation. jQuery, Umbrella JS
- Data handling. D3.js
- Database development. TaffyDB and ActiveRecord.js
- Forms. sForms, Live Validation, or qForms
- Animations. JSTweener or Anime.js
- Image effects. ImageFX or Reflection.js
- Fonts. Typeface.js
- Math and string functions. Date.js, Sylvester, or JavaScript URL library
- User interface and components.?ReactJS or Glimmer.js

As a package manager, NPM is often implicated in cybersecurity events. Hackers use it to launch attacks by placing malicious packages into the repository, hundreds of which are found and traced back to NPM monthly. These packages stole credentials and crypto and ran botnets and reconnaissance.

One popular package was found in hundreds of JavaScript projects now at risk.

The most popular JavaScript libraries include:

- jQuery
- React.js
- D3.js
- Underscore.js
- Lodash
- Algolia Places
- Anime.js
- Animate on Scroll (AOS)
- Chart.js
- Polymer
- Voca

These and many other libraries provide code for the whole gamut of behavior, from data visualization to user interface interactivity. No part of a project is without risk unless the developer builds it and keeps the code and libraries updated.

The Most *Vulnerable* JavaScript Containers

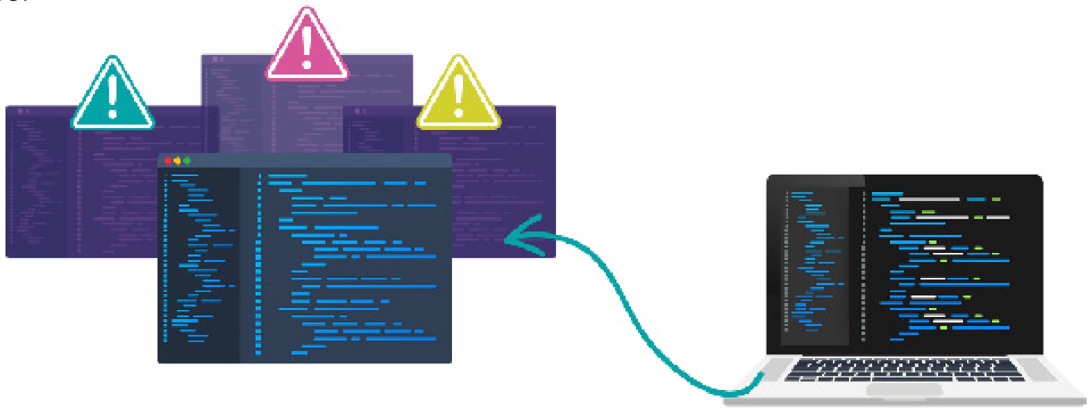
By far, the most popular library is jQuery, making it the favorite library to exploit. After that follow:

- jQuery UI
- Moment.js, Angular.js
- Handlebars
- Mustache, YUI 3
- jQuery mobile
- Knockout
- React

Unfortunately, React exploits are likely underreported.

What Are the *Risks of Vulnerabilities* in JavaScript Containers?

The most egregious issue is that if a website carries one known vulnerability, it probably carries others. Over half of vulnerable sites have multiple known security vulnerabilities. The majority of them only have one or two, but **nearly 10% of sites** carry libraries with four or more combined known security vulnerabilities.



For example, a vulnerability was discovered in the NPM package ua-parser-js that could allow hackers to execute remote code once installed on a target system. They use the vulnerability to detect browser, engine, OS, CPU, and device type and model information from user-agent data. Industries report the vulnerability as being actively exploited for malicious purposes.

Mitigating JavaScript Risks

Fortunately, many vulnerabilities are addressed within the library. Each library has at least one version available that is without known security vulnerabilities. The challenge is to identify them and get them into production. The use of automation would simplify and streamline the effort to keep packages up to date.

It also helps if developers receive training to build awareness and understanding of the security issues found within JavaScript packages.

JavaScript *Best* Practices

You can take advantage of several tools and processes to reduce and mitigate security vulnerabilities and their impact on your JavaScript projects.

Use a JavaScript linter.

Static code analysis tools called linters can automate checks for the following:

- Programmatic errors
- Code smells (characteristics in the source code that may indicate a deeper problem)
- Stylistic errors
- Known security exploits

The best-known linters are JSLint, JSHint, and ESLint. Alternatively, you can use the pluggable J avaScript linting functionality of Visual Code Studio and Atom.

Use a package manager to audit dependencies.

You can also use a package manager to audit dependencies. A package manager can track all the packages you have in use on a website and track, manage, and update dependencies. Examples include NPM, Yarn, and pnpm.

Package managers allow you to audit your packages for common JavaScript security issues.

Add SRI (subresource integrity) for external scripts.

Subresource integrity is built into most modern browsers. SRI uses a cryptographic hash to verify external script integrity. Use a command line-tool like Shasum or OpenSSL or generate a hash using a SRI Hash Generator.

Add the hash value generated for the external JavaScript file to the integrity attribute of the **<script>** or **<link>** element. Also, add the crossorigin=anonymous attributes to send a cross-origin request without credentials.

Avoid inline JavaScript and develop a content security policy.

Inline script tags can leave a website more exposed to cross-site scripting attacks. Mitigate the risk by **avoiding all scripts as external .js files**, including inline event handlers.

A content security policy (CSP) adds a security layer to the communication pathways between client and server. Add content security rules to your HTTP response header according to your CSP. Without inline scripts in your code, it's easier to set up an effective CSP, using script-src and default-src directives to block those scripts entirely. If malicious scripts attempt to execute, they automatically fail.

Validate user input.

HTML5 forms have form validation attributes built in. These allow you to analyze user data and return error messages without client-side JavaScript. Some examples include:

- **required**
- **min**
- **max**
- **type**

The pattern HTML attribute validates the value of an input using a regular expression, another technique to validate user input.

Also, most browsers support a Constraint Validation API so that you can perform custom input validation. The API extends the JavaScript interfaces that belong to various HTML form elements, like HTML InputElement, HTMLButtonElement, and HTMLSelectElement, all providing properties and techniques for comparing input validity against multiple constraints.

The API reports validity status as well as performs other actions.

Don't rely only on client-side validation. Malicious actors use some tools that bypass those validations and allow attacks directly on the server.

Escape or encode user input.

Encode or escape incoming or untrusted data to avoid cross-site scripting attacks. The method converts special characters that might pose a security threat into a safe form. You should always encode HTML entities when you receive them from an untrustworthy source.

- **Encoding** adds an extra character before a potentially risky character, like \ before quotation marks.
- **Escaping** converts a character into a safe, equivalent format like > places into the &gt; HTML string.

Some free escaping or encoding tools include JavaScript String Escaper and URL Encoder/Decoder. Don't use JavaScript methods and properties, like innerHTML, that return unescaped strings. Use the textContent property instead.

Use a CSRF token not stored in cookies.

Hackers exploit vulnerabilities in session cookies so they can present themselves as an authenticated user. You can prevent the exploit. Send a supplementary token with each HTTP request. A hacker cannot access a CSRF token because it isn't stored in cookies. You can add these tokens to the following:

- **Forms**
- **AJAX calls**
- **HTTP headers**
- **Hidden fields**
- **All state-changing requests, validating on the back end**

Transmit cookies securely.

Only transmit cookies using a secure protocol like HTTPS, which encrypts the data sent between client and server. Add the ;secure flag to the Document.cookie property that provides access to the document cookies.

Also, use the ;samesite flag to control cookie transmission in cross-site requests.

Minify, bundle, and obfuscate JavaScript code.

The harder it is for attackers to penetrate your code, the better. Using minify and bundle code tools like Webpack makes it difficult for hackers to understand and duplicate your scripts' logic and structure.

Obfuscation is a series of code transformations that turns simple JavaScript code into a modified version that is difficult to understand and reverse-engineer. In particular, you want to obfuscate any data in your code, so use obfuscation to hide variables, objects, and strings to make it hard for a hacker to understand the type of data that might be within the code.

Beyond concealing data, you can use obfuscation to hide layout and program control flow and include optimization methods to target the following:

- **Booleans**
- **Functions**
- **Identifiers**
- **Predicates**
- **Numbers**
- **Statements**
- **Regular expressions**

The most common obfuscation techniques for JavaScript include encoding, reordering, renaming, logic concealing, and splitting.

Use caution, however. Obfuscation alone is not a secure practice. Use it in addition, not instead of good security practices. The more layers you have to your security, the less likely an attacker can defeat it.

Your ***DevSecOps team should use*** a multi-pronged approach to prevent security issues in their source code. Auditing, testing, and dynamic scanning are all required to keep hackers at bay.

The Last Word in JavaScript *Security*

Security threats abound in the wild. Educating employees on prevention methods is essential to help your IT teams, including DevOps, system administrators, and development teams. It shows them how attacks happen and how to prevent them when designing web applications.

The OWASP Top 10 Security Risks for 2022 shows 94% of applications were tested for a form of broken access control, the top issue on this year's list, followed by cryptographic failures and injection. Security misconfigurations, insecure design, identification and authentication failures, server-side request forgery, and vulnerable and outdated components also made the list.

Before sending out your latest JavaScript application, provide protection using the techniques above.

Also, allow Kiuwan to help your DevSecOps team mitigate code security risks, improving your application security using our DevOps tools during software development. Your team can manage every aspect of the development process, from code analysis to governance, without exposing your JavaScript code to vulnerabilities or attacks.

Kick your authentication practices and security testing up a notch with Kiuwan's Code Security (SAST) and Insights (SCA) to mitigate security incidents and comply with a constantly expanding regulatory environment.

*YOU KNOW **CODE,** WE KNOW **SECUIRTY!***

GET IN TOUCH:

Headquarters
2950 N Loop Freeway W, Ste 700
Houston, TX 77092, USA

United States **+1 732 895 9870**
Asia-Pacific, Europe, Middle East and Africa **+44 1628 684407**
contact@kiuwan.com
Partnerships: **partners@kiuwan.com**