no-same-type-params: An ESLint Rule Against Silent Argument Swaps

Kabui, Charles

2026-02-17

Table of Contents

An ESLint rule that catches consecutive function parameters sharing the same type — the kind of bug TypeScript won't warn you about.

The Bug TypeScript Misses

TypeScript's type system will catch a `string` where a `number` is expected. But what about this?

```
function copyFile(source: string, destination: string, adapterId: string) {
  // ...
}

// Oops - swapped source and destination. TypeScript says nothing.
copyFile(destPath, sourcePath, adapterId);
```

All three arguments are `string`. TypeScript is satisfied. The code compiles. The bug ships.

This happens constantly in real codebases. File operations, coordinate systems, API calls — anywhere you have multiple parameters of the same type, you have a potential silent swap.

The Rule

`@mckabue/no-same-type-params` warns when two or more **consecutive** parameters share the same type annotation:

```
//  Warning: consecutive params 'source' and 'destination' share type 'string'
const copyFile = (source: string, destination: string) => {};

//  No warning - object parameter, impossible to swap
const copyFile = ({ source, destination }: CopyParams) => {};

//  No warning - different types between same types break the chain
const fn = (a: string, b: number, c: string) => {};
```

Why "Consecutive"?

The rule only flags consecutive same-type params, not all same-type params anywhere in the signature. Why?

Non-consecutive params with a different type between them are harder to accidentally swap. You'd have to skip over a completely different argument, which most editors and reviewers will catch. Consecutive same-type params are the real danger zone.

Installation

```
npm install @mckabue/no-same-type-params --save-dev
```

Configuration

ESLint Flat Config (recommended)

```
import noSameTypeParams from '@mckabue/no-same-type-params';
import * as tsParser from '@typescript-eslint/parser';

export default [
  {
    files: ['**/*.ts', '**/*.tsx'],
    languageOptions: { parser: tsParser },
    plugins: {
      '@mckabue': {
        rules: { 'no-same-type-params': noSameTypeParams },
      },
    },
    rules: {
      '@mckabue/no-same-type-params': 'warn',
    },
  },
];
```

What It Catches

```
//  Two consecutive strings
const move = (from: string, to: string) => {};

//  Three consecutive strings (2 warnings)
function transfer(source: string, dest: string, bucket: string) {}

//  Same array types
const merge = (a: string[], b: string[]) => {};

//  Same union types
const pick = (a: string | number, b: string | number) => {};

//  Default values with same types
const set = (key: string = 'k', value: string = 'v') => {};
```

What It Ignores

```
//  Different types
const fn = (a: string, b: number) => {};

//  Object parameter (the recommended fix)
const fn = (opts: { from: string; to: string }) => {};

//  Non-consecutive same types
const fn = (a: string, b: number, c: string) => {};

//  Untyped parameters
const fn = (a, b) => {};

//  Single parameter
const fn = (a: string) => {};
```

The Fix: Object Parameters

When the rule warns, the idiomatic fix is almost always an object parameter:

```
// Before: error-prone
function createUser(name: string, email: string, role: string) {}

// After: self-documenting, impossible to swap
function createUser({ name, email, role }: CreateUserParams) {}
```

Object parameters are:

- **Self-documenting** at call sites
- **Order-independent** — no swapping risk
- **Extensible** — add new fields without breaking callers
- **Grep-friendly** — `name:` is searchable, positional args aren't

Bonus Tip: Pair with `max-params`

For maximum effect, combine this rule with ESLint's built-in `max-params` rule:

```
rules: {
  '@mckabue/no-same-type-params': 'warn',
  'max-params': ['warn', { max: 3 }],
}
```

Together they form a one-two punch:

- **`max-params`** warns when a function has too many parameters (encouraging object params for complex signatures)
- **`no-same-type-params`** warns when consecutive params share the same type (catching swap risks even in short signatures)

A function with 2 string params triggers `no-same-type-params` but not `max-params`. A function with 4 distinct-type params triggers `max-params` but not `no-same-type-params`. Both push you toward the same idiomatic solution: **object parameters**.

Install: `npm install @mckabue/no-same-type-params --save-dev`

GitHub: https://github.com/mckabue/no-same-type-params **npm:** https://www.npmjs.com/package/@mckabue/no-same-type-params

---