










npm v0.0.1

A robust validator for Zod schemas, designed to ensure the safety and integrity of code, especially when generated by LLMs.

[Features](#) • [Quick Start](#) • [Installation](#) • [Usage](#) • [Design Decisions](#) • [Future Work](#)

## 🌟 Key Features

-  **Safeguard LLM-Generated Schemas:** Automatically detects and prevents common vulnerabilities that can arise when using LLMs to generate Zod schemas, ensuring your application remains secure.
-  **Automatic Code Cleanup:** Removes invalid code and ensures your schemas conform to the required structure by automatically exporting valid schemas as constants and cleaning up extraneous code.
-  **Resource Control:** Implements configurable limits on execution time, node count, and nesting depth, protecting your application from denial-of-service issues and runaway processes.
-  **Detailed Issue Reporting:** Provides clear, actionable reports on any issues found in your schemas, including line numbers, column numbers, and suggestions for fixing them.
-  **Highly Configurable:** Offers a range of configuration options, from relaxed to extremely safe presets, allowing you to tailor the validation process to your specific needs.
-  **Fast and Efficient:** Designed to quickly validate your Zod schemas without sacrificing performance, using caching and optional parallel processing.
-  **Strict Zod Import Enforcement:** Ensures that your schema definitions properly import `z` from `zod`, preventing common errors and enhancing code maintainability.

## 🚀 Quick Start

```
# Validate a schema file using the default 'relaxed' config
bunx zodsheriff schema.ts

# Validate a schema from stdin using the medium config
cat schema.ts | bunx zodsheriff --stdin --config medium

# Validate a schema from the clipboard with the extremely safe config
bunx zodsheriff --clipboard --config extremelySafe

# Output only the cleaned schema code
bunx zodsheriff schema.ts --clean-only
```

## 📦 Installation

### As a CLI Tool

```
npm install -g zodsheriff # or bun add -g zodsheriff
```

## As a Package

```
npm install zodsheriff # or bun add zodsheriff
```

## Usage

### Command Line Interface

```
zodsheriff <input-file> [options]
```

Options:

- `<input-file>`: Path to the TypeScript file containing Zod schema definitions.
- `--stdin`: Read schema from standard input.
- `--clipboard`: Read schema from the system clipboard.
- `--config <level>`: Set validation config: `extremelySafe`, `medium`, or `relaxed` (default: `relaxed`).
- `--clean-only`: Output only the cleaned schema code (no validation reports).
- `--json`: Output result in JSON format.
- `--help`: Show this help message.

## As a Module

```
import { validateZodSchema, mediumConfig } from "zodsheriff";

async function validateMySchema() {
  const schemaCode = `
    import { z } from 'zod';
    export const userSchema = z.object({
      name: z.string(),
      age: z.number()
    });
  `;

  const result = await validateZodSchema(schemaCode, mediumConfig);

  if (result.isValid) {
    console.log("Schema is valid!", result.cleanedCode);
  } else {
    console.error("Schema is invalid!", result.issues);
  }
}

validateMySchema();
```

# Design Decisions

---

ZodSheriff is designed to provide a balance between safety and usability, especially when dealing with code generated by LLMs. Here's a breakdown of key design considerations and the reasoning behind them:

## 1. Schema Extraction and Transformation

- **Focus on `const` Declarations:** We explicitly target `const` declarations for schema definitions. This encourages immutability, which is a security best practice. Schemas are expected to be immutable once declared.
- **Auto-Export Valid Schemas:** To ensure the validated schemas can be used, ZodSheriff automatically wraps valid schema declarations in `export const` statements. This eliminates the need for manual exporting, which is often forgotten, especially with LLM generated code.
- **Preserving Code Comments:** We carefully preserve comments throughout the code transformation process. This ensures that important developer notes and LLM-generated explanations are not lost during validation.

## 2. AST Traversal and Node Handling

- **Ignoring Invalid Paths:** ZodSheriff uses a targeted traversal approach, only visiting nodes that are relevant to Zod schema definitions. Invalid paths and other constructs are ignored by design, ensuring that we focus on the core schema logic.
- **Explicitly Allowed Statements:** We define a strict whitelist of allowed statement types (`ImportDeclaration`, `ExportNamedDeclaration`, `VariableDeclaration`, `ExportDefaultDeclaration`). Any other statements trigger a validation error, ensuring that the code contains only what is necessary to define a Zod schema.
- **Strict Import Enforcement:** We require that the Zod import must use the name `z`. This prevents common errors and ensures that the code is easily understood.

## 3. Method Chain and Argument Validation

- **Limited Method Whitelists:** We use whitelists for Zod methods (`allowedZodMethods`, `allowedChainMethods`). This allows for a high degree of safety by limiting the attack surface. Any unknown or potentially unsafe methods are flagged.
- **Function Validation:** For methods like `refine` and `transform`, we validate the function bodies to ensure they are not asynchronous or generators, further reducing the risk of unintended side effects.
- **Regex Safety:** Regular expressions are checked using the `safe-regex` library to prevent catastrophic backtracking and denial-of-service attacks. We also limit the length of regex patterns to prevent excessively long patterns.
- **Argument Count Validation:** We validate the number of arguments passed to Zod methods, ensuring that they match the expected usage, which helps catch common errors.

## 4. Resource Management

- **Configurable Limits:** We enforce limits on node count, execution time, object depth, and chain depth, using a `ResourceManager`. This protects against resource exhaustion and infinite loops.

- **Aggressive Timeout Checks:** We perform aggressive timeout checks (at 90% of the limit) to help catch performance issues before they become critical.
- **Caching:** Validation results are cached to prevent redundant processing. Caching is enabled by default, but can be turned off.

## 5. Balancing Safety and Convenience

- **Configurable Presets:** ZodSheriff comes with three configuration presets (`extremelySafe`, `medium`, and `relaxed`). This allows users to choose the level of safety that best suits their needs.
- **Flexibility:** You can also override individual config settings to fine-tune the validation process.
- **Detailed Reporting:** Even when invalid schemas are found, ZodSheriff provides detailed error reports to make it easy to understand and correct the issues.

## 6. No Runtime Code Injection

- ZodSheriff **does not** attempt to automatically fix your schemas by injecting code. We believe that it is safer to explicitly report errors and leave it to the user to correct the code.

## Future Work

---

Here's a roadmap of potential enhancements and future work:

- **More Granular Function Body Analysis:** Implement more detailed analysis of function bodies, including static analysis to detect potentially unsafe operations.
- **Runtime Checks:** Develop a system for adding runtime checks to Zod schemas to catch errors that cannot be detected during static analysis.
- **Customizable Method Lists:** Allow users to extend the allowed method lists with their own custom methods.
- **Improved Error Messages:** Provide even more specific and helpful error messages to guide users in fixing their schemas.
- **Worker Threads:** Implement worker threads for CPU-bound tasks to improve performance with larger schemas.
- **Support for More Zod Features:** Expand the validation process to cover more advanced Zod features, such as custom types and schema composition.
- **Automatic Fixes:** As a future opt-in feature, consider providing the option for automatic fixes for simple issues with user confirmation.

We're continually working to improve ZodSheriff. Your feedback is always welcome.

Created by [Hrishi Olickel](#) • Support ZodSheriff by starring our [GitHub repository](#)