🔒 ZodSheriff

Enforce rigorous safety and validation on Zod schemas generated by LLMs.

ZodSheriff is a tool designed to ensure that Zod-based schema definitions created (or augmented) by Large Language Models (LLMs) are safe, valid, and production-ready. By applying a set of careful validations and transformations, ZodSheriff helps you trust and safely execute schemas that might otherwise be risky or prone to injection attacks.

Key Use Case: You have code (TypeScript/JavaScript) that includes `zod` schemas, some or all of which may be generated by an LLM. Before using them in your application, you want to ensure they are safe: no suspicious imports, no malicious code paths, no overly deep object expansions, etc. ZodSheriff provides a second line of defense—preventing suspicious patterns and cleaning the code into a minimal, validated form.

🎯 Features

- **Strict Schema Validation:** Ensures that each schema declaration follows safety rules—only `const` declarations, no dangerous imports, and each schema must have a proper initializer.
- **Allowed Methods Only:** Limits the set of Zod methods to a known safe subset, preventing unexpected runtime code paths generated by LLMs.
- **Depth & Size Limits:** Enforces configurable maximum depths for objects and chains, maximum node counts, and other resource controls to prevent malicious or resource-exhausting schemas.
- **Regex & Function Checks:** Validates that refinements and transforms use safe functions and that regex patterns are not catastrophic.
- **Automatic Cleanup & Export:** Removes invalid nodes, disallowed imports, or suspicious code constructs while preserving as much of the original schema as possible. If desired, it can auto-export cleaned schemas, ensuring you always have a tidy, minimal, and safe output.

🚀 Quick Start

You can run ZodSheriff as a CLI tool to validate and clean a given schema code:

```
npx zodsheriff schema.ts

# or if installed globally:
zodsheriff schema.ts
```

By default, it:

- Validates that `z` is properly imported from `zod`.
- Ensures only allowed Zod methods are used.
- Strips out unsafe code.
- Enforces configuration limits (e.g., max chain depth).
- Produces a cleaned version of the schema code if it can.

If the schema passes validation, you'll see a success message and the cleaned code. If not, a list of issues will be printed.

⚙️ Configuration

ZodSheriff comes with multiple preset configurations (e.g., `extremelySafeConfig`, `mediumConfig`, `relaxedConfig`) and a function to create or override these settings. These configs let you tune:

- **Timeouts & Node Limits:** Avoids infinite loops or extremely large schemas.
- **Depth Controls:** Restrict how deeply nested objects or method chains can be.
- **Property Safeties:** Deny unsafe property names (`__proto__`, `constructor`) or suspicious prefixes.
- **Regex Checks:** Rejects catastrophic or suspicious regex patterns.
- **Function Validation:** Disallows async or generator functions in schema refinements.

```
import { validateZodSchema, extremelySafeConfig } from 'zodsheriff';

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

const result = await validateZodSchema(code, extremelySafeConfig);
console.log(result.isValid);      // true/false
console.log(result.cleanedCode); // Cleaned schema code if valid
console.log(result.issues);      // Array of reported issues
```

🌱 Design Considerations

ZodSheriff has been architected with a deliberate balance between safety and developer convenience. Here are some key design choices:

1. **Refuse Suspicious Code by Default:**
   The tool blocks computed object properties, unsafe imports, and disallowed schema methods. Rather than prompting for user approval, ZodSheriff errs on the side of safety. If it's not on the allowed list, it's removed.
2. **Preserve as Much as Possible:**
   While removing invalid parts, ZodSheriff tries to keep the remaining structure intact. It doesn't just fail entirely; it attempts to produce a cleaned, minimally-destructive output. This allows you to see which parts of the LLM-generated schema were accepted and which were stripped out.
3. **Keep Comments & Formatting Sensibly:**
   The goal is to produce a cleaned, valid schema that is still human-readable. Comments and some formatting details are kept whenever possible, so you can understand what the original schema intended.

4. **Turn All Schemas into Exported Consts:**
   Schemas that pass validation but aren't exported will be wrapped in an export statement. This ensures that the final result is usable directly without additional manual changes, facilitating a seamless pipeline from LLM -> ZodSheriff -> Production Code.
5. **Ignore Bad Paths:**
   If a particular node (e.g., an unsafe property or a malicious regex) is detected, ZodSheriff removes it rather than attempting complex repairs. This straightforward removal policy reduces complexity and ensures no partial unsafe logic remains.
6. **Balance Safety and Usability:**
   Some might find the rules too strict or too lenient. For example, template expressions or loops might be disallowed in some modes. The configuration system allows you to pick your comfort level. Still, by default, ZodSheriff chooses safety over convenience.

🤖 Why Safe Execution Matters

LLMs can generate code that "looks" safe but might embed malicious constructs or hidden infinite loops. Even if the schema is just defining data shapes, a cleverly constructed schema could cause performance issues (e.g., catastrophic backtracking in regex) or resource exhaustion (e.g., excessively large object graphs).

By passing every LLM-generated schema through ZodSheriff, you gain confidence that:

- Imported dependencies are legitimate.
- The schema methods are from the known safe Zod API.
- No suspicious runtime code (like `eval`, `constructor` properties) sneaks in.

🔮 Future Improvements & Roadmap

- **More Granular Configuration:**
  Currently, configurations allow broad strokes. Future versions may allow fine-grained tuning of allowed methods, property names, or argument validators.
- **Interactive Mode:**
  Provide a CLI mode where you can see what's removed and possibly whitelist certain patterns interactively.
- **Performance & Parallelism Enhancements:**
  For extremely large codebases, further optimization and parallel validation could speed up processing.
- **More Sophisticated Heuristics:**
  Move beyond a known set of rules to heuristic-based detection (e.g., suspicious naming patterns or suspicious code-like strings within strings).

ZodSheriff is a step towards making LLM-generated code safer by default. As LLMs evolve, so must our safety measures.

📦 Installation

```
bun install zodsheriff
```

Or using `npm`:

```
npm install zodsheriff
```

Example CLI Usage

```
zodsheriff schema.ts --config extremelySafe
```

This will:

- Parse `schema.ts`
- Validate imports, node counts, nesting depth, and allowed methods
- Print issues if any
- If valid, produce a cleaned and auto-exported schema

ZodSheriff helps you transform the uncertain into the trustworthy, ensuring that your LLM-generated schemas don't compromise your codebase's safety or integrity.