MCP Dev Summit Mumbai

Your MCP Server Is an Attacker's Dream

A Security Playbook From Real-World Assessments

AKASH MAHAJAN

Founder CEO • 2x Tech Book Author • 2x Company Builder

Now building a Solo Founder Software Factory

15+

MCP Servers with *Critical* Authorization Bugs

MCP Dev Summit Mumbai

Shipping Fast, Skipping Security, YOLO

MCP Dev Summit Mumbai

1. Missing object level authorization (IDOR)
2. Unauthenticated callers read the whole catalog
3. RBAC present but ignored

The Lethal Trifecta

1

Access to private data

Overprivileged tools expose data the server never needed to touch.

2

Exposure to untrusted content

Tool poisoning & injected content the LLM treats as trusted commands.

3

Ability to exfiltrate

SSRF via tool args & transport misconfig open an outbound channel.

Survivable alone. Lethal in combination. — *concept coined by Simon Willison (2025)*

PART ONE

How the Trifecta Shows Up in Prod MCP Servers

Tool Poisoning - UNTRUSTED CONTENT GETS IN

How the attack works

- Attacker plants instruction text in data a read tool returns
- Planted text on a page of a docs/wiki platform
- The read tool returns it verbatim
- Agent fires a comment-creation mutation tool
- The model treats returned data as instructions

Why it slips past code review

- Both tools are working as designed.
- The bug is in the read → write interaction
- Reviewers fixate on poisoned tool descriptions; the dangerous variant is poisoned tool

What the fix looks like

- ✓ Treat tool output as untrusted data, never instructions.
- ✓ Put a confirmation gate on state-changing calls driven by read output
- ✓ Don’t echo user-authored content verbatim into agent context.

Permission Escalation – PRIVILEGE & DATA ACCESS

How the attack works

- On a personal-CRM MCP server, any logged-in user who supplies a victim's *contact_id* can read, modify and create their objects
- Cross-account *update_contact* / *create_note* returned HTTP 200.
- On an enterprise ITSM platform, an employee key set *requester: <org-admin>* and impersonated the admin.

Why it slips past code review

- The server checks are you logged in? and stops
- Ownership checks are assumed upstream.
- Identity-override fields (*requester*, *owner*, *assignee*) read as business parameters, not auth boundaries.

What the fix looks like

- ✓ Enforce object-level authz on **every** tool taking a resource ID
- ✓ Strip client-supplied identity fields server-side
- ✓ Filter *tools/list* by role.

SSRF via Tool Arguments – THE EXFIL PATH

How the attack works

- A tool makes server-side request to an attacker URL with a stored credential into that outbound request.
- The server injected the secret as a header to the attacker's endpoint.
- The secret never appeared in any MCP response; it was captured purely by controlling the destination.

Why it slips past code review

- SSRF-by-design
- The tool exists to make outbound calls, so it reads as a feature.
- The trust assumption is only our client calls this, no host egress filtering makes it worse

What the fix looks like

- ✓ Destination allowlist (block non-integration / non-RFC1918 endpoints)
- ✓ Bind each credential to its destination pattern
- ✓ Scope keys so a chatbot key can't reach the HTTP-proxy tool.

Chaining all three to steal internal credentials

A REDACTED platform

1. Tools list returns 79 tools to every key, regardless of role. In-spite of RBAC.
2. A credential-list tool returns 100+ stored credentials metadata
3. One HTTP tool advertised as any URL, auth handled by the platform, no destination allowlist.
4. Call it with an attacker URL + a CREDENTIAL_REFERENCE, the platform resolves the secret server-side into the outbound headers.
5. Attacker endpoint reflects the injected credential. The secret never traversed an MCP response. Loop all IDs.
6. One chatbot-grade key drains every stored integration credential in the tenant.

THE ASSESSMENT PLAYBOOK

Assess Any Server Before Prod

MCP Dev Summit Mumbai

7 Steps Security Review for your MCP Server

1. Map the surface. Inventory every tool, schema, resource & annotation.
   - ❑ Do it unauthenticated first. What leaks before login?
2. Classify tool power. Read-only / state-changing / destructive
   - ❑ Flag danger fields (URL, path, code, query, identity).
3. Enumerate injection surfaces. Tool descriptions and the data that tools return.
4. Build the confused-deputy graph. Map every read → write pair
   - ❑ SSRF-source → credential-sink = high, search→search = noise.
5. Test authz with ≥2 principals. Same tool, cross-user / cross-org / cross-role.
   - ❑ Positive + negative controls catch IDOR.
6. Active-safe probing. SSRF canary callbacks, credential-ref exfil, requester spoof
   - ❑ Operator-owned resources only, with cleanup.
7. Promote evidence → issues. Every claim backed by request/response logs
   - ❑ Verify false- vs true-positive against raw schemas.

The Pre-Prod Checklist

Pre-prod checklist item	Defends leg
tools/list rejects unauthenticated callers (no schema leak pre-auth)	**Untrusted**
tools/list filtered by caller role (low-priv keys don't see admin tools)	**Privilege**
Every tool taking a resource ID enforces object-level authz (no IDOR)	**Privilege**
Client-supplied identity fields (requester/owner/assignee) ignored server-side	**Privilege**
MCP keys scope-restricted per tool (chatbot key ≠ HTTP-proxy / credential access)	**Privilege**
Tool descriptions free of behavioral directives (“proceed anyway”)	**Untrusted**
Tool output / user content not returned verbatim into agent context	**Untrusted**
State-changing calls triggered by read output require a confirmation gate	**Untrusted**
Outbound-request tools enforce a destination allowlist (block SSRF)	**Exfil**
Stored credentials bound to a destination; not usable with arbitrary URLs	**Exfil**
No host-level egress to arbitrary external URLs (egress filtering)	**Exfil**
Cloud metadata / localhost / internal ranges blocked from request tools	**Exfil**
Audit log + alert on credential-ref use against an unmatched destination	**Exfil**
Bearer-token lifetimes sane (not multi-year)	**Privilege**
Mass-read tools (list users / credentials) rate-limited + minimal fields	**Privilege**

Cut One Leg, Kill the Attack

The exploit needs all three legs. Remove any one and the attack collapses.

UNTRUSTED CONTENT

Break this leg

Treat tool metadata & external content as tainted. Pin and review tool definitions. Isolate descriptions from the instruction context.

PRIVILEGE

Break this leg

Scope every tool to least privilege. No standing broad data access. Separate read from write; require approval for sensitive scopes.

EXFIL PATH

Break this leg

Allow-list outbound destinations. Validate URLs in tool args. Enforce transport auth. Block arbitrary HTTP from tools.

Hall of Fame vs. Hall of Shame

✓ Got it right

Pattern: real authz boundaries, SSRF allowlists, no pre-auth leak.

- **GitHub** cross-org access denied; org-scoped fine-grained PATs.
- **Supabase** bidirectional cross-principal access denied; boundary enforced.
- **Linear** read/destructive hints on all 46 tools; SSRF domain allowlist.
- **MotherDuck** no promotable vulns; destructive hint set on the SQL tool.
- **PostHog** required auth; unauth assessment blocked at transport.

✕ Got it wrong

Pattern: auth without authz, open catalogs, SSRF with no allowlist.

- **Enterprise ITSM platform** SSRF + credential proxy = tenant-wide secret leak. Critical.
- **Personal-CRM MCP** cross-account IDOR + 38 tools to unauthenticated callers. Critical.
- **Spec/collab platform** cross-user IDOR on 4 paths incl. destructive finalize. High x4.
- **Docs/wiki platform** tool-output injection drove a mutation tool. Medium.
- **Testimonials SaaS** read-tool output can drive a state change. Medium.

Doing Auth Right Doesn't Always Save You

Two servers got authentication exactly right, and still shipped a critical bug.

A DevTool Collab platform

- Enforced authentication perfectly
- 401 on unauthenticated calls, clean Oauth
- Then shipped four High-severity cross-user IDORs.
- Authenticated, but no ownership checks. **Auth ≠ authz.**

REDACTED platform

- Built a deliberately safe credentials API that never returns secret values
- Then added an HTTP-proxy tool that leaks those same secrets outbound.
- A safe API + a permissive tool = an unsafe catalog.

When tools work together, how data traverses across the tool boundaries can make you vulnerable

What to Take Back to Your Team

1. The trifecta is the lens. Any MCP server with private data, untrusted input, and an outbound path is one prompt away from compromise.
2. The mistakes are predictable. Tool poisoning, overprivileged definitions, SSRF via arguments, and transport misconfig show up again and again.
3. Code review may not catch it. These slip past because tool metadata reads like docs, not executable input. You need an assessment, not a skim.

Thank you.

Your MCP Server Is an Attacker’s Dream

A Security Playbook From Real-World Assessments

Akash Mahajan