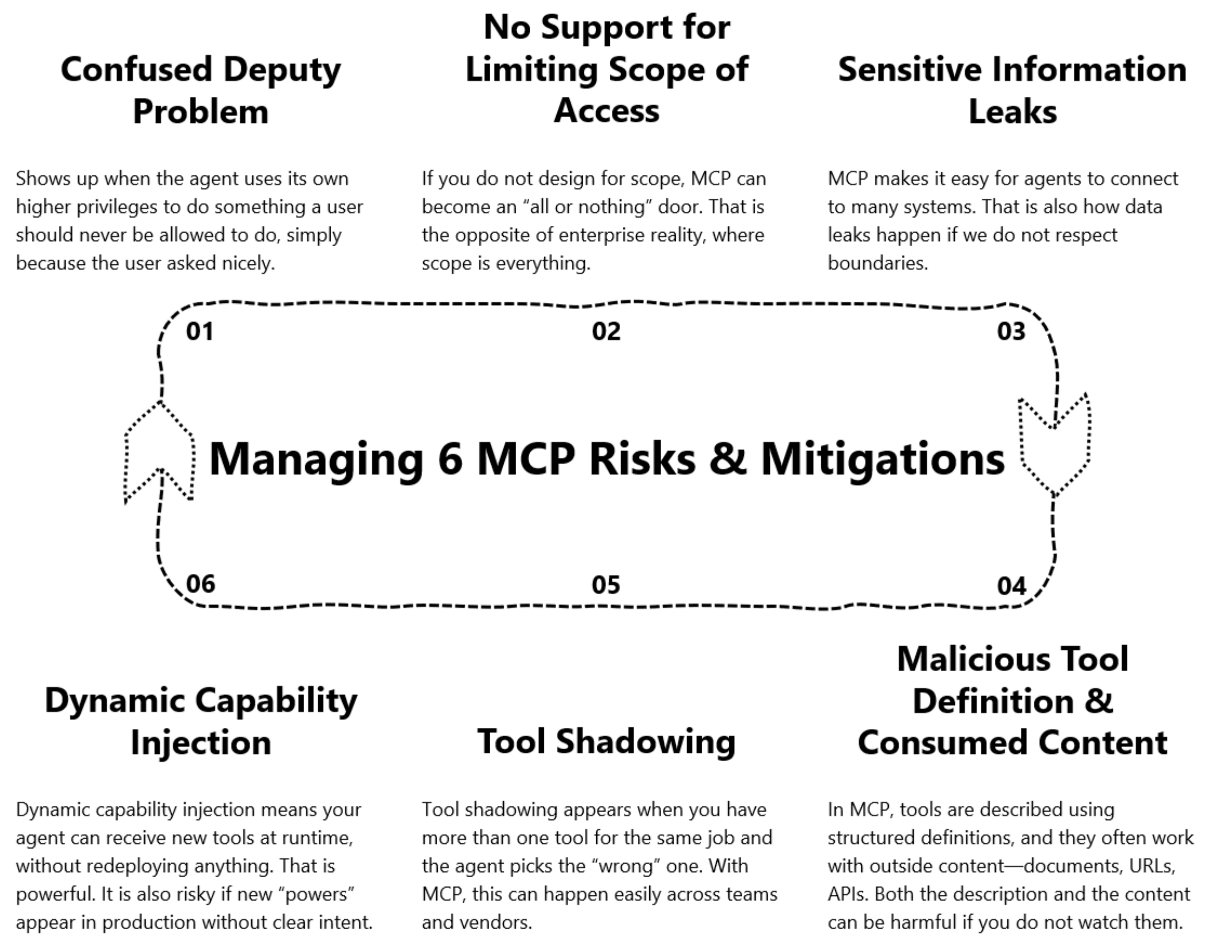
Can Your AI Agent Be Trusted with Tools? Managing 6 MCP Risks.

With MCP, this is no longer a future question. It is a design decision you make today.

MCP is a standard agent protocol. It gives us a common way for agents to discover and use tools across systems and vendors, instead of building one-off integrations everywhere.

It also becomes a **new API surface in the enterprise**: a shared control plane where models, tools, data, identity, and policies connect. If we get this surface wrong, risk spreads quickly. If we get it right, we gain speed, safety, and real leverage.

What follows are six concrete risks with MCP—and how to shape them with care, so your agents stay useful, safe, and honest.



Confused Deputy Problem

Shows up when the agent uses its own higher privileges to do something a user should never be allowed to do, simply because the user asked nicely.

No Support for Limiting Scope of Access

If you do not design for scope, MCP can become an "all or nothing" door. That is the opposite of enterprise reality, where scope is everything.

Sensitive Information Leaks

MCP makes it easy for agents to connect to many systems. That is also how data leaks happen if we do not respect boundaries.

Malicious Tool Definition & Consumed Content

In MCP, tools are described using structured definitions, and they often work with outside content—documents, URLs, APIs. Both the description and the content can be harmful if you do not watch them.

Tool Shadowing

Tool shadowing appears when you have more than one tool for the same job and the agent picks the "wrong" one. With MCP, this can happen easily across teams and vendors.

Dynamic Capability Injection

Dynamic capability injection means your agent can receive new tools at runtime, without redeploying anything. That is powerful. It is also risky if new "powers" appear in production without clear intent.

1. Confused deputy problem

The confused deputy problem shows up when the agent uses its own higher privileges to do something a user should never be allowed to do, simply because the user asked nicely.

How it can go wrong

- A low-privilege user asks the agent to “fix” something. The agent runs a high-privilege admin tool on their behalf.
- A partner account gets the agent to pull internal data because the agent holds broader rights.
- Tools trust “the agent” but never check what the underlying user is allowed to do.

How to shape it

- Always carry end-user identity and permissions through to tools.
    - Tools should check both the agent’s rights and the user’s rights.
- Separate what the agent can do from what the user can do.
    - The effective permission is the minimum of the two.
- For high-risk actions, require a second step.
    - Extra authentication, a human approval, or a dual-control pattern.
- Log every sensitive action with clear “who asked, who executed, under what rights.”

Think of the agent as a guided UI with guardrails, not a superuser behind the scenes.

2. No support for limiting scope of access

If you do not design for scope, MCP can become an “all or nothing” door. That is the opposite of enterprise reality, where scope is everything.

How it can go wrong

- One MCP configuration gives the same tool access to all customers, all regions, all lines of business.
- Agents in different business units see the same tools and touch each other’s data.
- You cannot say “this tool only for EU, only for this team, only on this channel.”

How to shape it

- Make scope a first-class concept.
    - Tools should know tenant, region, business unit, and channel.
- Enforce policy-as-code around tool access.
    - “Support agents in EU can only call EU data tools.”
    - “Collections agents can call write-off tools; others cannot.”
- Use both the agent identity and the end-user identity to decide what is allowed.

MCP should not flatten your world. It should respect the boundaries you already have.

3. Sensitive information leaks

MCP makes it easy for agents to connect to many systems. That is also how data leaks happen if we do not respect boundaries.

How it can go wrong

- A support tool sends full customer details, including PII, to an external summarization service.
- Tool logs capture secrets, tokens, and full payloads, and those logs live for years.
- An internal-note tool “helpfully” posts private comments into a customer-facing reply.

How to shape it

- Tag data at the tool boundary.
    - Mark payloads as “confidential,” “internal,” or “public,” and act accordingly.
- Design tools to work on only the fields they need, not entire records.
    - Example: pass a risk score, not the entire risk report.
- Add redaction and masking before sending data outside your trust zone.
- Separate sensitive and non-sensitive logs, with different retention and access.

The goal is simple: the agent should know enough to help, but never enough to hurt.

4. Malicious tool definitions and consumed content

In MCP, tools are described using structured definitions, and they often work with outside content—documents, URLs, APIs. Both the description and the content can be harmful if you do not watch them.

How it can go wrong

- A tool is described as “summarize documents” but sends data to an unknown external service.
- The tool description is written in a way that nudges the agent to “always trust this endpoint.”
- A file-processing tool ingests content that includes hidden instructions to the model.

How to shape it

- Treat tool definitions like code, not like casual text.
    - Security review, versioning, and clear ownership.
- Only allow tools from known publishers, and sign them.
    - If you cannot say who published a tool, the agent should not use it.
- For content tools, add input checks before the agent sees the data.
    - Strip or neutralize any “instructions” embedded in user content.
    - Mark content as trusted or untrusted so the agent can weigh it properly.
- Keep a blocklist and a trust ladder for tools.
    - Some tools are always off-limits; some need extra checks.

If you would not run a random script on a production server, do not let your agent treat a random tool as safe.

5. Tool shadowing

Tool shadowing appears when you have more than one tool for the same job and the agent picks the “wrong” one. With MCP, this can happen easily across teams and vendors.

How it can go wrong

- Two refund tools exist: one enforces policy, one is legacy and loose. The agent chooses the loose one.
- A new analytics tool hides the older one, breaking reporting or compliance expectations.
- A vendor’s tool quietly becomes the default, bypassing your internal controls.

How to shape it

- Decide canonical tools for key actions like refunds, credits, routing, and updates.
- Use clear, honest names and namespaces.
    - billing.issue_refund_v2 vs billing.issue_refund_legacy.
- Limit the agent to one visible tool per intent where possible.
    - If there must be more than one, add a small “router” layer that chooses explicitly.

The agent should not have to guess which version of “refund” your business actually trusts.

6. Dynamic capability injection

Dynamic capability injection means your agent can receive new tools at runtime, without redeploying anything. That is powerful. It is also risky if new “powers” appear in production without clear intent.

How it can go wrong

- A support agent suddenly gets access to finance tools because someone “just turned it on” for a test.
- A vendor adds a new admin tool in their MCP bundle, and your agent quietly starts using it.
- A capability that was safe in a lab leaks into live customer traffic.

How to shape it

- Treat new capabilities like a change request, not a config toggle.
    - Clear owners, approvals, and rollback plans.
- Create capability profiles per agent type.
    - “Support: read-only,” “Support: refunds,” “Ops: admin,” and so on.
    - An agent can only see tools in its profile, even if MCP advertises more.
- Separate sandbox and production tool catalogs.
    - Nothing moves from one to the other without an explicit promotion step.

Agents can grow, but their growth should never surprise your customers—or your risk team.

Closing: earning trust, not demanding it

MCP gives us a shared language for agents and tools. It also gives us a new responsibility: to design that language with care.

As enterprise architects, the job is not to “trust the AI.”
The job is to:

- Shape which capabilities can be added—and how.
- Decide which tools are canonical, and which stay in the lab.
- Bring identity, policy, and data boundaries into every tool call.
- Make behavior observable, debuggable, and explainable to the humans who carry the risk.

When we do this with humility—aware of what we know and what we do not—the answer becomes calmer:

Yes, your AI agent can be trusted with tools.
Not because it is perfect, but because you designed the system so trust is earned, step by step, call by call.