cerbos

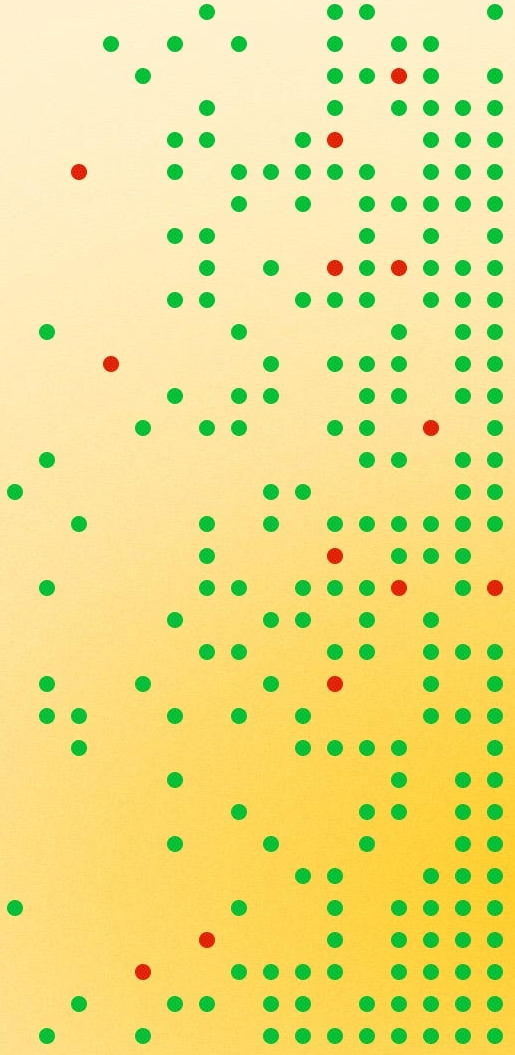
Authorization in MCP systems

Getting it right from the start

MCP Dev Summit Seoul | 13 August 2026

Aram Andreasyan

Director of Solutions Engineering, Cerbos



cerbos

Four companies, one bug

Ten years, four industries, and the same conversation every time.

Experian	Talend	Qubit and Coveo	Cerbos
Credit data	Data integration	Ecommerce search	Developer tooling

Four industries. The same bug in every one.

cerbos

We solved who you are, we never solved what you can do

One half of the problem got finished. The other half never did.

Authentication

Solved. You do not write your own login in 2026.

Authorization

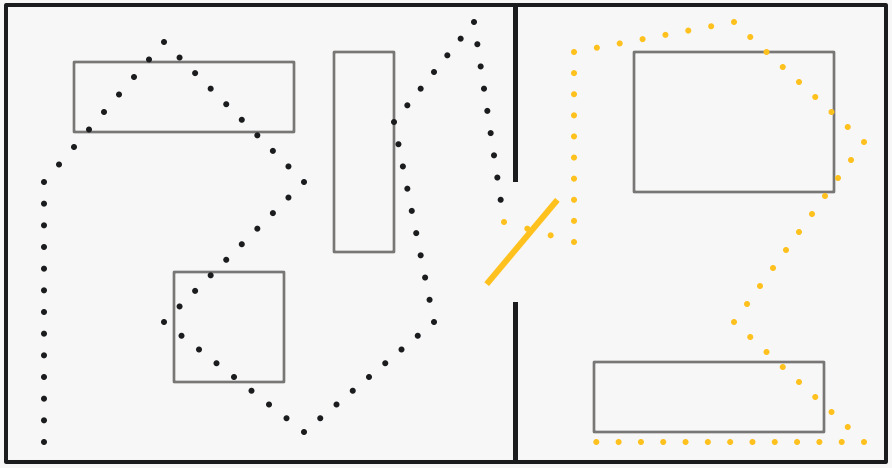
Still handwritten, in every service, slightly differently.

For thirty years that was fine, because the traffic was human. A person who cannot get in gives up and goes to lunch.

cerbos

It does not clean the floor

It maps the house by collision, and so does an agent with tools.



the door you left open, and the path that finds it

Twenty minutes, a complete map

Every wall it hits becomes information. It only stays harmless because you closed the bathroom door.

Your permissions are the house

The agent calls a tool, takes the denial, changes the arguments, and calls again. It has no concept of good enough.

A person gives up in ninety seconds.

cerbos

Same misconfiguration. Same missing check.
Same line of code you wrote two years ago.

Same bug. Very different Tuesday.

cerbos

Two decisions, and most teams only make one

Authentication settles before the protocol opens. Then you decide twice.

`OAuth 2.1`	**Who is this. Settled before the first request.**
`tools/list`	**Which tools does this principal get to see** `the protocol makes this one mandatory`
`tools/call`	**May it do this, with these arguments** `most teams implement only this one`

Discovery is skipped. Invocation is checked by name, not by argument.

cerbos

The safest deny is the tool the model never knew existed

And a smaller catalog makes the agent better at its job.

Everything the server exposes

```
list_tickets
search_kb
order_lookup
note_add
macro_apply
audit_read
refund_issue
refund_bulk
invoice_void
payment_capture
export_customers
user_impersonate
kb_delete
order_cancel
```

→ tools/list

What this principal can call

```
list_tickets
search_kb
order_lookup
note_add
```

The model cannot pick what it never saw

There is no denial to handle, because there is no attempt.

The context window got smaller

Fewer tools in context, and the agent chooses better.

cerbos

In MCP the resource is inside the arguments

The tool name tells you the shape of the operation, not what it touches.

REST

```
GET /customers/12345
```

the resource is in the path, and any gateway can read it

MCP

```
{ "method": "tools/call",
  "params": {
    "name": "get_customer",
    "arguments": { "id": "12345" }
  } }
```

Parameter level authorization is the only level at which the decision means anything.

cerbos

Nobody re authorizes a live token

The token is checked on every request. What it allows is never checked again.

No re authorization step
No revocation signal to the server
Token TTL is the only clock

MCP delegates to OAuth 2.1 and RFC 9728. Token lifetime is whatever your identity provider decided.

Unless you decide per call.

cerbos

Four things teams do today

Named without judgment, because most of us have shipped at least two.

01 **Authorization in the system prompt**
Written as an instruction the model is asked to follow.

02 **Trust the token, check nothing after**
A valid token is treated as a current answer.

03 **Check the tool name, never the arguments**
The control cannot see what the request actually touches.

04 **The same check copied into three services**
Three copies agree on the day you write them and never again.

cerbos

Why the prompt is not a control

The most common thing I see, and the one that worries me most.

A prompt is a suggestion

It is requested, not enforced. Testing it means running the model.

Not enforced
Not independently testable
Not versioned separately
No record of what was decided

system prompt

```
You are only allowed to issue
refunds under 50 dollars and only
for tickets assigned to the user.
```

It works most of the time

Which reads exactly like a control that is working. Until the model finds a path where following the instruction and finishing the task are not compatible.

And then it completes the task.

cerbos

Shape before scope

Getting it right from the start is about the shape, not the scope.

01

Externalized

The decision does not live inside the code that enforces it.

Where it lives

02

Evaluated at runtime

Per request, with current information, not resolved an hour ago.

When it decides

03

Centralized

One place the rules are written. Many places they are enforced.

Where it is written

04

Auditable

Every decision leaves a record of who asked and which policy answered.

What it leaves behind

Get the shape right and the scope can be tiny.

Get it wrong and no amount of scope saves you, because every change costs you a deployment and an argument.

cerbos

Three dimensions on your own server

Fill in the cells you know. The blank ones are a map of what you have not decided yet.

		Support agent	Triage agent	Admin
list_tickets	discover	✓	✓	✓
list_tickets	call	✓	✓	✓
refund_issue	discover	✓	×	✓
refund_issue	call	✓	×	✓
export_customers	call			✓
kb_publish	call			✓

for wider use cases: https://www.cerbos.dev/blog/mapping-business-requirements-to-authorization-policy

cerbos

Pick one tool, ship it, iterate

Crawl, then walk, then run.

Not one server. Not one application. One tool.

Take its checks out of the handler, put them in policy, and change nothing else about your system.

01

Roles, one tool

Start with roles, because they are easy and they get you further than you expect.

Crawl

→ 02

Attributes, the arguments

Add attributes the moment roles stop being able to express the rule.

Walk

→ 03

The rest of the catalog

Only once the shape has survived contact with real traffic.

Run

production starts at step one, not step three

Write tests for the policy before you wire it to anything.

cerbos

The author of the tool declares how it should be judged

x-authzen-mapping, declared by the tool and delivered in tools/list.

x-authzen-mapping, simplified

```
"inputSchema": {
  "properties": { "id": { "type": "string" } },
  "x-authzen-mapping": {
    "subject":  "$user",
    "action":   "read",
    "resource": {
      "type": "customer",
      "id":   "$params.arguments.id"
    },
    "context":  { "agent": "$agent" }
  }
}
```

There is already a standard

AuthZEN at the OpenID Foundation. The Authorization API is published. The MCP profile, COAZ, was approved as a working group draft in June 2026.

It arrives at discovery time

The declaration travels in the tools/list response, so the enforcement point learns how to authorize a tool without any configuration.

Declared interest. Our co founder and chief product officer co chairs the working group and co authored the MCP binding.

cerbos

The tool with no authorization

It takes an amount and a ticket. When an agent calls it, money moves.

refund_issue

```
// no authorization anywhere in here
server.tool("refund_issue", {
  amount_cents: z.number(),
  ticket_id:    z.string(),
}, async ({ amount_cents, ticket_id }) => {
  const r = await payments.refund(
    ticket_id, amount_cents
  );
  return { ok: true, refund_id: r.id };
});
```

Reasonable on day one

One internal user, and you are still finding out whether the tool is useful. Adding a policy engine before you know that is procrastination, not discipline.

The problem is that it has nowhere to go

Every rule you add from here lands inside this handler, and inside the gateway, and inside the API behind it.

cerbos

Step one, it disappears from the catalog

A decision on discovery, and nothing else in the system changes.

Support agent connects

```
list_tickets
search_kb
order_lookup
note_add
refund_issue
```

Triage agent connects

```
list_tickets
search_kb
order_lookup
note_add
```

The model cannot choose a tool it never saw

There is no denial to handle, because there is no attempt.

The context window got smaller

And the agent got better at the job you actually gave it.

- **The refund handler has not changed. Same code as the last slide.**

cerbos

Step two, the rule reads what the model actually asked for

One engine's syntax. Look at the structure, not the keywords, because the shape is the same in Cedar or OPA.

```
resourcePolicy:
  resource: "mcp_tool"
  rules:
    - actions: ["call"]
      roles: ["support_agent"]
      effect: EFFECT_ALLOW
      condition:
        match:
          all:
            of:
              - expr: R.attr.arguments.amount_cents ≤ P.attr.refund_limit
              - expr: R.attr.arguments.ticket_id == P.attr.assigned_ticket
      output:
        when:
          conditionNotMet: "over limit or wrong ticket"
```

```
// in refund_issue, before payments.refund
const d = await cerbos.checkResource({ principal, resource });
if (!d.isAllowed("call")) return denied(d);
```

Request

principal: alice (support_agent)
limit: 5000
ticket: T-4471
asked: 12000, T-9002

Decision

⊗ Deny

12000 over the 5000 limit, and T-9002 is not T-4471

One line in the handler. Every rule after this is a policy change.

cerbos

The rules you cannot write inside the handler

Two things the policy layer does that a tool handler cannot.

External context

```
the handler knows
  amount, ticket, caller
the policy engine also fetches
  refunds_today  41,400
  daily_cap      50,000
```

Tested in isolation

```
given   alice, 12000, T-9002
expect  DENY
runs in CI. no model.
no MCP server. no network.
```

Twenty small refunds, also denied

The handler cannot see the running total for today. The policy engine asks for it at decision time.

A policy is a test file

Run the rules in CI without booting the model or the MCP server.

- **This is the rule that stops the vacuum trying the smaller door.**

cerbos

Denial is a signal, not a wall

A 403 to a human is an error page. A 403 to an agent is input.

Empty denial

```
{
  "error": {
    "code": -32003,
    "message": "Access denied"
  }
}
```

the agent retries with different arguments

A denial the agent can act on

```
{
  "error": {
    "code": -32003,
    "message": "Access denied",
    "data": {
      "reason": "amount 12000 over limit 5000",
      "policy": "mcp_tool:default",
      "remediation": "ask a human approver"
    }
  }
}
```

it escalates, explains, or stops

A good denial tells the agent which door to try instead, or that there is not one.

cerbos

Three things to take away

01 **Decide the shape before the scope**
Externalized, runtime, centralized, auditable.

02 **Pick one tool. Ship it. Iterate.**
Do not try to fill the whole grid before anything reaches production.

03 **Read the COAZ working draft**
It is short, it is still a draft, and you can shape it before it is final.

`openid.net/wg/authzen`

`linkedin.com/in/aramandreasyan`

I would rather hear what broke in your system than tell you about mine.