



Delegated Authorization for AI Agents

Build an Agent with Fine-Grained Permissions

Sohan Maheshwar

@AuthZed

AAIF Ambassador

Throwback Thursday #tbt

close

 **Circles**

You share different things with different people. So sharing the right stuff with the right people shouldn't be a hassle. Circles makes it easy to put your friends from Saturday night in one circle, your parents in another, and your boss in a circle by himself - just like real life.

 Lisa Stiemman	 Steve Rura	 Anthony Cafaro	 Emily Stiebel
 Sam Stiebel	 Natalie Hammel	 Sara Rowghani	 Alex Chen

Click and drag people into circles

Ski Crew
0

San Diego peeps
0

Nana & Grandpa
0

Access Control in Google Products



You need access

Request access, or switch to an account with access.

[Learn more](#)

- ☐ Viewer
- ☐ Commenter
- ☒ Editor

Message (optional)

Request access



Access Control in ChatGPT Apps

ChatGPT

New chat

Search chats

Library

Apps

Codex

More

Projects

New project

Apps

Chat with your favorite apps in ChatGPT

reference

box

Box (Legacy)

Search and reference your documents

Google Contacts

Reference saved contact details

Notion (Legacy)

Search and reference your Notion pages

Gmail

Find and reference emails from your inbox

Outlook Email

Search and reference your Outlook email

Mobbin

Find UI & UX design references



4

OWASP Top Ten Web Apps Security Risks

- #1 Broken Access Control (2021 and 2025)
- 100% of apps tested had broken access control issues

OWASP Top 10:2025 RC1

Welcome Page

Current Release

OWASP Top 10:2021 (Current Release) >

Release Candidate

OWASP Top 10:2025 (Release Candidate) v

Introduction

About OWASP

What are Application Security Risks?

Establishing a Modern Application Security Program

Top 10:2025 List v

A01 Broken Access Control

A02 Security Misconfiguration

A03 Software Supply Chain Failures

A04 Cryptographic Failures

A01:2025 Broken Access Control



Background.

Maintaining its position at #1 in the Top Ten, 100% of the applications tested were found to have some form of broken access control. Notable CWEs included are *CWE-200: Exposure of Sensitive Information to an Unauthorized Actor*, *CWE-201: Exposure of Sensitive Information Through Sent Data*, *CWE-918 Server-Side Request Forgery (SSRF)*, and *CWE-352: Cross-Site Request Forgery (CSRF)*. This category has the highest number of occurrences in the contributed data, and second highest number of related CVEs.

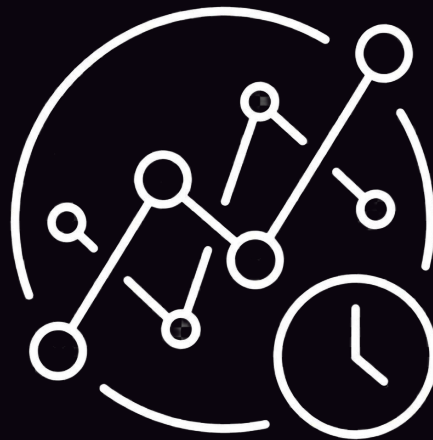
Score table.

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Max Coverage	Avg Coverage	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences
40	20.15%	3.74%	100.00%	42.93%	7.04	3.84	1,839,701

owasp.org/Top10/2025/

Ambient Context

- N users $\times$ M agents $\times$ O actions
- Evolving graph of **relationships**, **states**, and **entities** (users, agents, resources)
- Not fully contained in tokens (JWTs), ACLs, or request payloads.
- Too complex for **coarse-grained** techniques such as RBAC



Primer on Authorization

Access Control Types

- Access Control Lists
- Role-Based Access Control
- Attribute-Based Access Control



Google Zanzibar

- Globally distributed authorization system
- >10 million client queries per second
- From the developer's perspective - it's an API
- ReBAC

The screenshot shows a web browser displaying the 'Zanzibar: Google's Consistent, Global Authorization System' paper. The browser's address bar shows 'zanzibar.tech'. The page header includes the title and a link to 'AuthZed'. The main content area features the title, authors (Ruoming Pang, Ramón Cáceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner*, Jeffrey L. Korn, Abhishek Parmar, Christina D. Richards, Mengzhi Wang), and their affiliations (Google, LLC; Humu, Inc.; Carbon, Inc.). The abstract discusses the design, implementation, and deployment of Zanzibar, a global system for storing and evaluating access control lists. It mentions that Zanzibar provides a uniform data model and configuration language for expressing a wide range of access control policies from hundreds of client services at Google, including Calendar, Cloud, Drive, Maps, Photos, and YouTube. The paper also notes that Zanzibar scales to trillions of access control lists and millions of authorization requests per second to support services used by billions of people. The introduction section is visible at the bottom of the page.

Zanzibar: Google's Consistent, Global Authorization System

Ruoming Pang, Ramón Cáceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner*, Jeffrey L. Korn, Abhishek Parmar, Christina D. Richards, Mengzhi Wang

Google, LLC; Humu, Inc.; Carbon, Inc.
{rpang, caceres}@google.com

Abstract

Determining whether online users are authorized to access digital objects is central to preserving privacy. This paper presents the design, implementation, and deployment of Zanzibar, a global system for storing and evaluating access control lists. Zanzibar provides a uniform data model and configuration language for expressing a wide range of access control policies from hundreds of client services at Google, including Calendar, Cloud, Drive, Maps, Photos, and YouTube. Its authorization decisions respect causal ordering of user actions and thus provide external consistency amid changes to access control lists and object contents. Zanzibar scales to trillions of access control lists and millions of authorization requests per second to support services used by billions of people. It has maintained 95th-percentile latency of less than 10 milliseconds and availability of greater than 99.999% over 3 years of production use.

1 Introduction

semantics and user experience across applications. Second, it makes it easier for applications to interoperate, for example, to coordinate access control when an object from one application embeds an object from another application. Third, useful common infrastructure can be built on top of a unified access control system, in particular, a search index that respects access control and works across applications. Finally, as we show below, authorization poses unique challenges involving data consistency and scalability. It tackle them once across applica

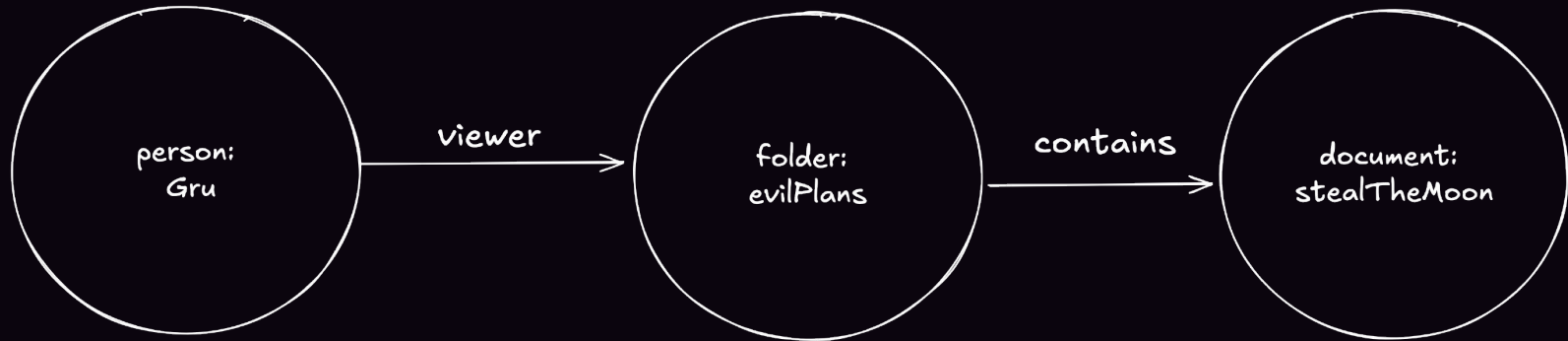
We have the following goals

- *Correctness*: It must ensure decisions to respect user in
- *Flexibility*: It must support policies as required by both applications.
- *Low latency*: It must respond quickly because authorization checks are often in the critical path of user interactions. Low latency at the tail is particularly important for serving search results, which often require

At no point in the paper is the full list of operators outlined. This is one of the reasons why when people throw around the term "Zanzibar spec" it's not necessarily concrete enough to consider.

www.zanzibar.tech

Relationship Based Access Control



User

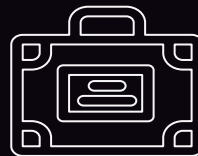
- Represents a natural person or service account



userid: 10957135

Object

- Represents a non-user entity
- Unique identifier prefixed with an object type



**folder:
evilPlans**



**doc:
secret-doc-id**

Relation

- The way one object can be related to another object or user
- Examples
 - **member** of a group
 - **editor** of a document
 - **uploader** of a video

Relation Tuple

- Datum which expresses a single relationship

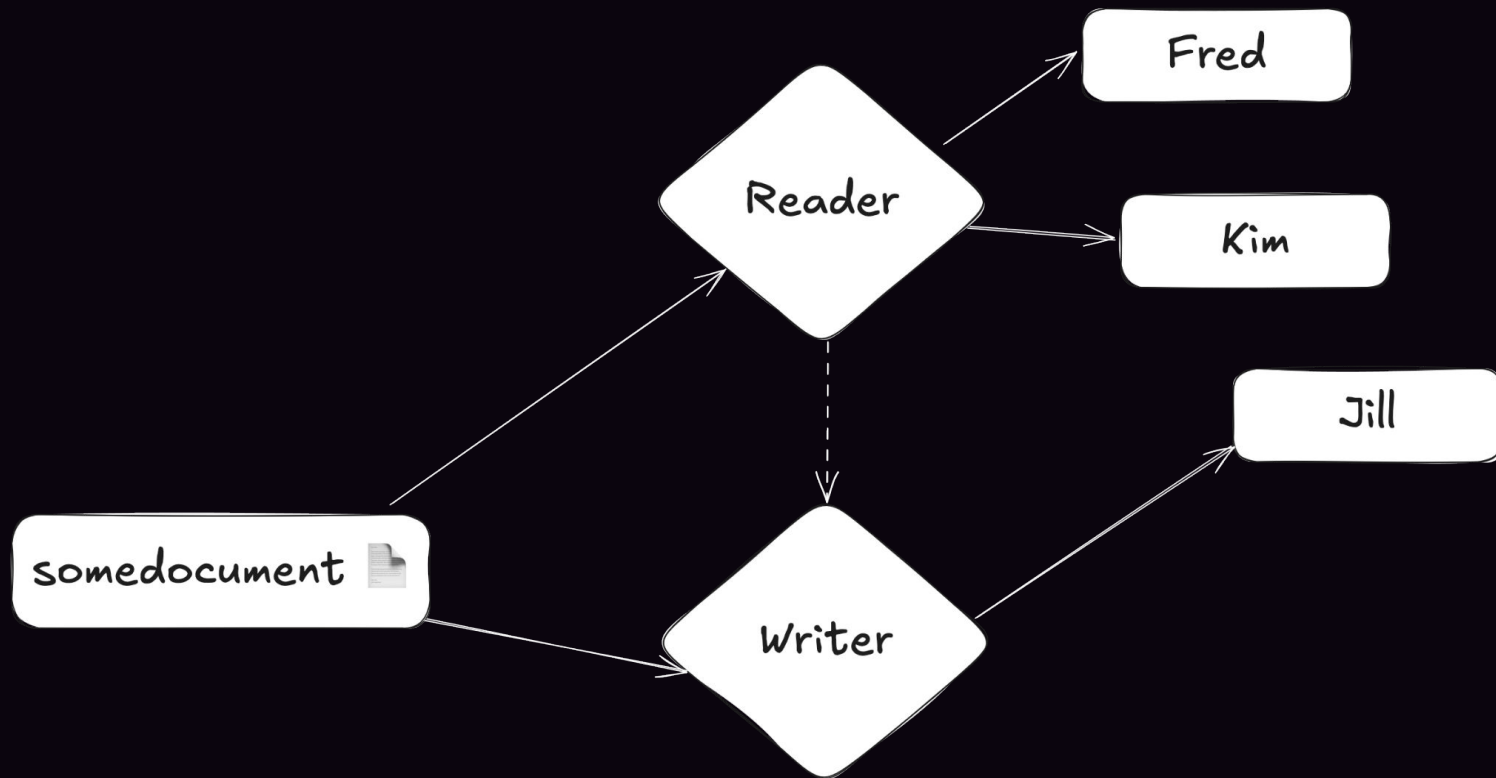
document:123#owner@user:3



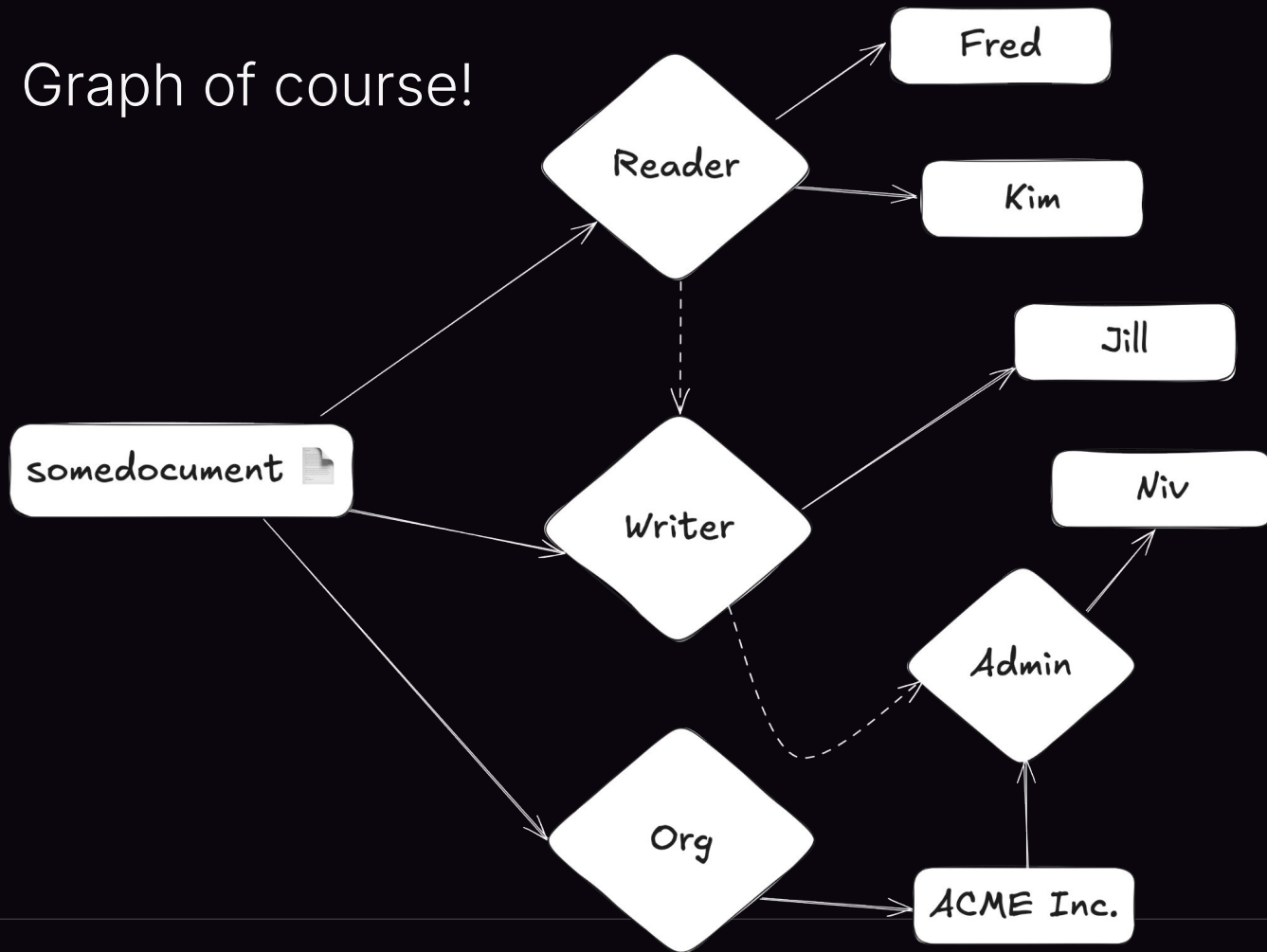
User **3** is an **owner** of a **document** with identifier **123**

What are we really doing here?

Build a Graph of course!

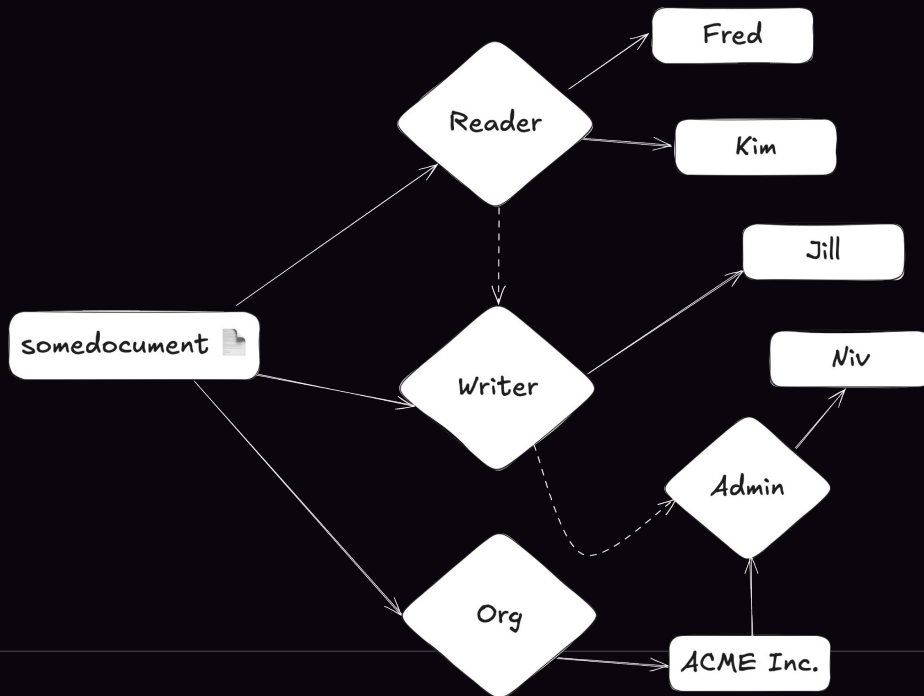


Build a Graph of course!



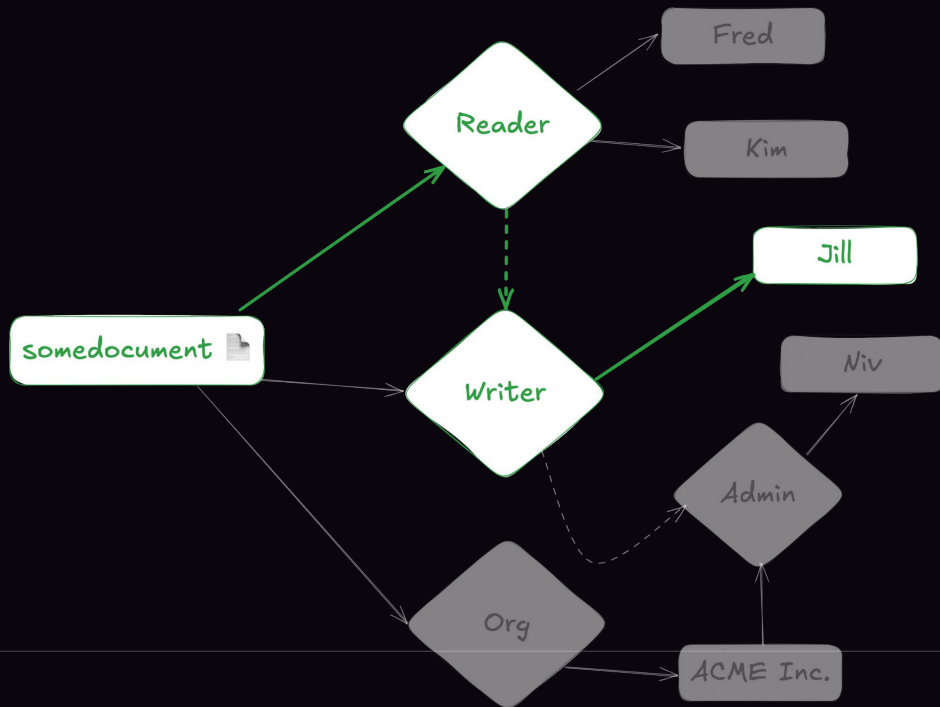
Check graph traversal example

```
check("document:somedocument", "reader", jill)
```



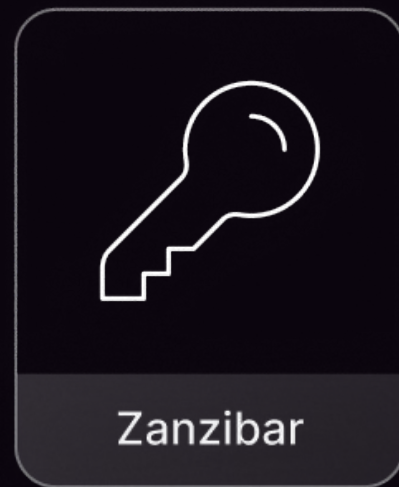
Check graph traversal example

`check("document:somedocument", "reader", jill)`



Where do Zanzibar-like systems shine?

- Low-latency
- High-throughput authorization checks
- Hierarchical permission models
- **Ambient Context** access control



SpiceDB

- Open source implementation of **Zanzibar**
- Widespread project adoption
- Contributors from Fortune 500 companies

github.com/authzed/spicedb

7K 🌟s & counting!



Goose by AAIF

- General-purpose **AI agent** that runs on your machine
- Connect to 70+ extensions
- Any LLM, including your subscriptions



A DevOps Agent

- Delegated Authorization
- Has access to two environments
 - staging
 - production
- Time-bound grants
- Hierarchical contingencies

The screenshot displays the ReBAC (Relationship-Based Access Control) interface for a system named 'goose' connected to 'SpiceDB'. The interface shows a delegation from a 'DELEGATOR user:alice' to an 'AGENT agent:goose_alice'. The delegation is for the 'staging' environment and is time-bound, with a timer showing 59:41. The interface includes buttons for 'Approve prod · 10m', 'Revoke staging', 'Revoke prod', and 'Reset'. A status bar indicates 'Demo reset - staging delegated for 60 min, versions restored'. A button labeled 'Deploy checkout to production' is visible. Below this, a code block shows a command 'deploy(checkout, production)' followed by a yellow warning box that says 'NEEDS APPROVAL'. The text below the warning box reads: 'deploy checkout -> production', 'agent:goose_alice lacks 'deploy'; delegator user:alice holds it - human approval required'.

goose × SpiceDB

ReBAC RBAC

RELATIONSHIP-BASED · CASCADE ON

DELEGATOR user:alice → DELEGATES → AGENT agent:goose_alice → GRANTS

staging ⌚ 59:41

Approve prod · 10m Revoke staging Revoke prod Reset

⌵ Demo reset - staging delegated for 60 min, versions restored

Deploy checkout to production

```
▶ deploy(checkout, production)
```

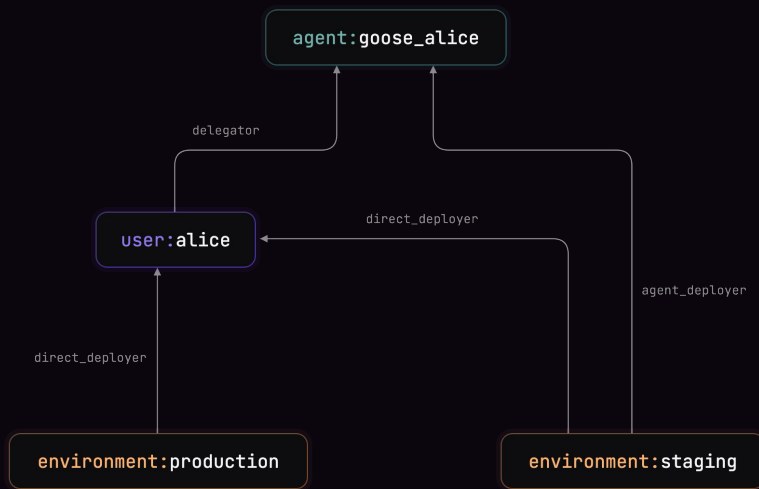
NEEDS APPROVAL

deploy checkout -> production

agent:goose_alice lacks 'deploy'; delegator user:alice holds it - human approval required

1. Delegated Authorization

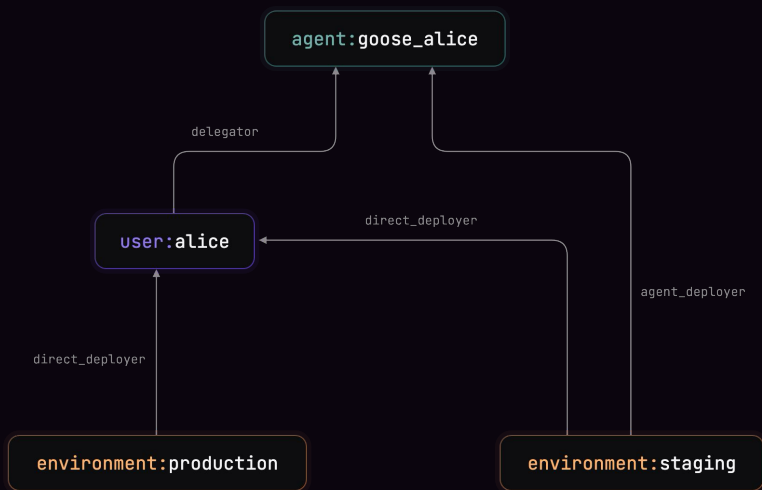
- Agent **goose_alice** acts on behalf of **user:alice**
- Permission is checked for every action



✗ With RBAC / tokens: copying a human's roles onto the agent drifts and over-grants

a. Schema

- Defines
 - Types of objects found
 - How those objects **relate** to one another
 - **Permissions** that can be computed off of those relations.



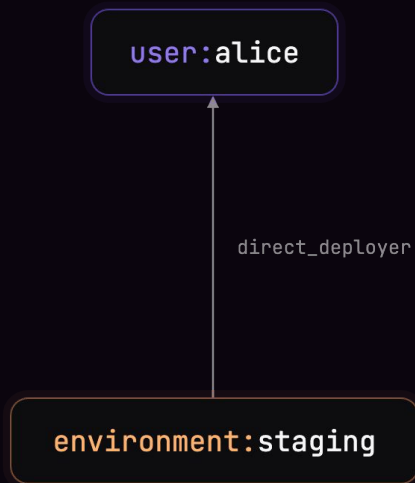
○ ○ ○

```
1 definition user {}
2
3 definition agent {
4     relation delegator: user
5 }
6
7 definition environment {
8     relation direct_deployer: user
9     relation agent_deployer: agent
10    permission deploy = direct_deployer + agent_deployer
11 }
```

`environment:staging#direct_deployer@user:alice`

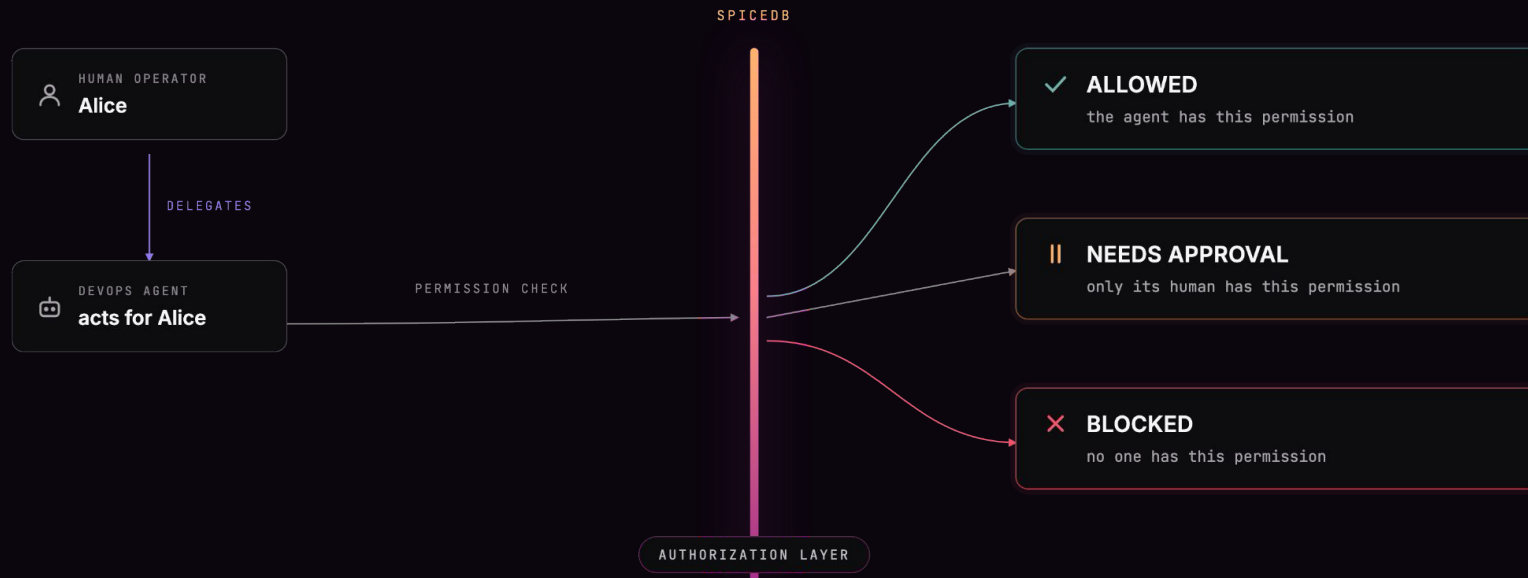
b. Relationships

- Relationships bind together a **Subject** and a **Resource** via a **Relation**.
- A functioning Permissions System is the combination of Schema and Relationships



c. Permission Checks

- SpiceDB is called for all permission decisions
- Three-way decision



2. Time-Bound Access

- "Give **prod access** for the duration of the incident"

The screenshot displays a ReBAC (Relationship-Based Access Control) interface for a system named 'goose x SpiceDB'. The interface is divided into two main sections. The top section shows a 'DELEGATOR' (user:alice) delegating access to an 'AGENT' (agent:goose_alice). The delegation is for 'production' access, with a duration of 09:48. The 'staging' access is also shown, with a duration of 59:47. Below this, there are buttons for 'Approve prod - 10m', 'Revoke staging', 'Revoke prod', and 'Reset'. The bottom section shows a log of actions, including 'Demo reset - staging delegated for 60 min, versions restored' and 'alice approved production for 10 min'. A button 'Deploy checkout to production' is also visible. The interface is dark-themed with green and orange accents.

✗ With RBAC/tokens: tokens lag; a cron leaves a window where a dead grant still works

Expiring Relationships

- Grant a user access to a resource for a **limited time**
- The relationship is not deleted immediately
 - It is subject to garbage collection

○ ○ ○

```
1 use expiration
2
3 definition user {
4 }
5
6 definition environment {
7     relation direct_deployer: user
8     relation agent_deployer: agent with expiration
9     ...
10 }
```

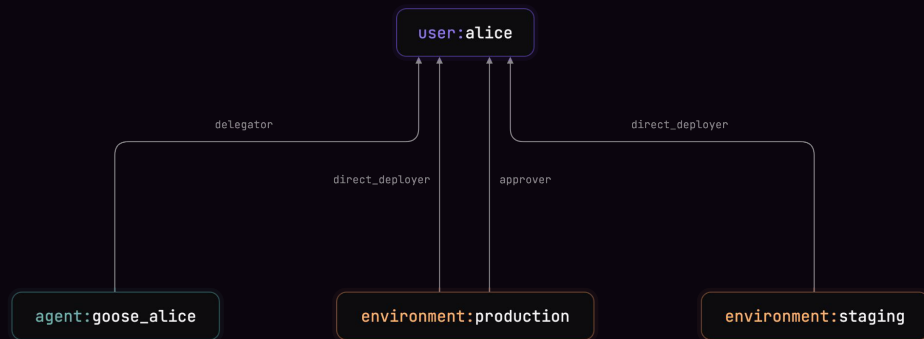
Add it to the relationship

○ ○ ○

```
1 from authz import expiry_from_now
2
3 rel("environment", "staging", "agent_deployer", "agent", AGENT_ID,
4     expires_at=expiry_from_now(window_minutes)),
```

3. Hierarchies

- “The agent can deploy to prod **only while** it can also deploy to staging”
- Model real-world hierarchies, nested groups and dependencies.
- Contingent evaluation $\neq$ cascading delete.

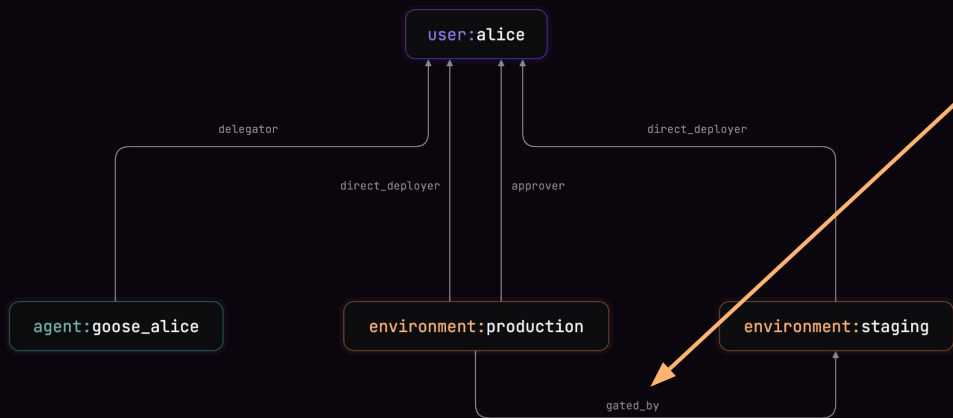


✗ With RBAC/tokens: no role can depend on another - you drop into app code



3. Hierarchies

- “The agent can deploy to prod **only while** it can also deploy to staging”
- Model real-world hierarchies, nested groups and dependencies.
- Contingent evaluation $\neq$ cascading delete.



✗ With RBAC/tokens: no role can depend on another - you drop into app code

Schema Operators

- + Union
- & Intersection
- - Exclusion
- -> Arrow

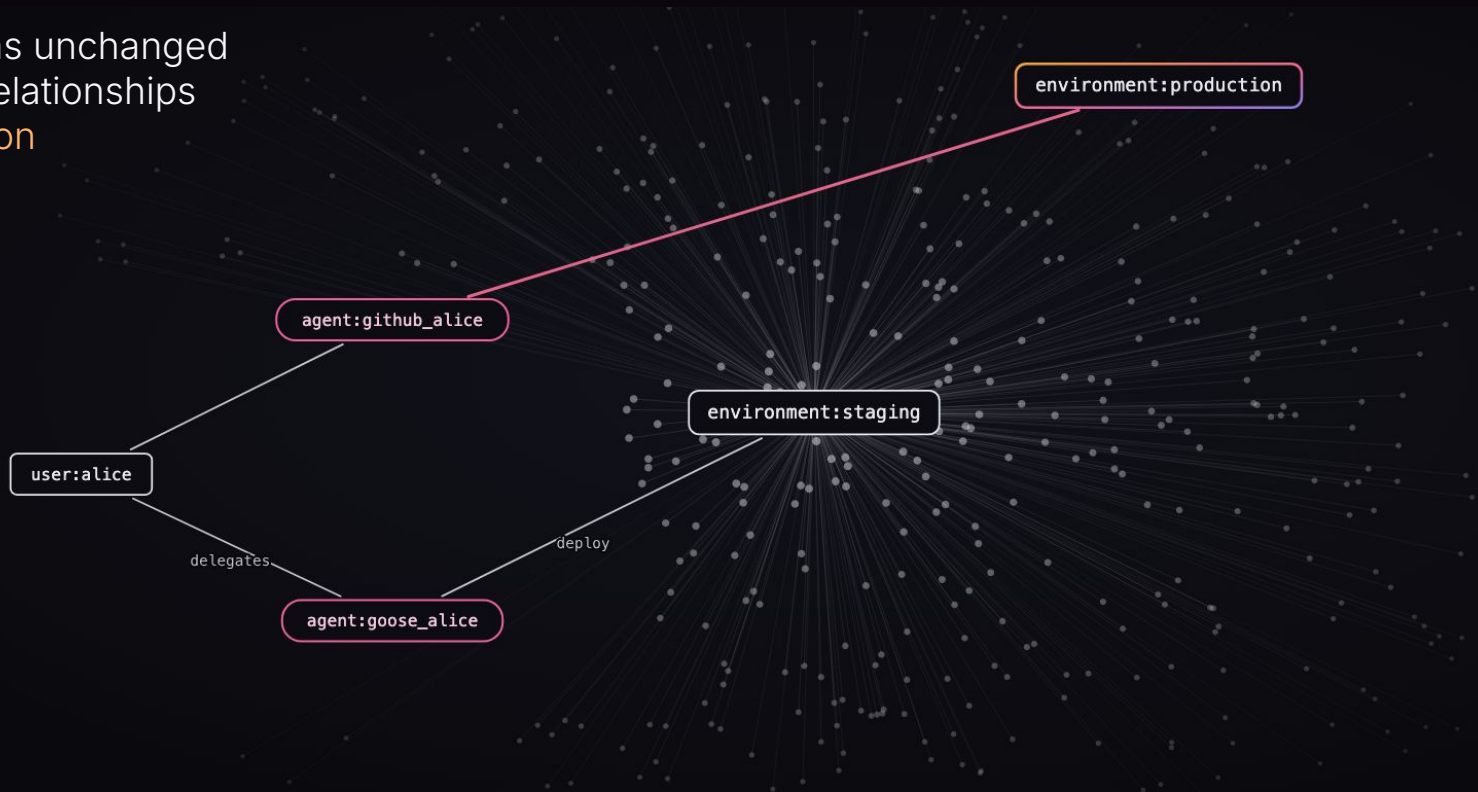
○ ○ ○

```
1 definition environment {  
2   relation direct_deployer: user  
3   relation agent_deployer: agent with expiration  
4   ...  
5   relation gated_by: environment  
6   permission agent_deploy = agent_deployer & gated_by->agent_deployer  
7   permission deploy = direct_deployer + agent_deploy  
8   ...  
9 }
```



Many users, agents, environments

- Schema remains unchanged
- Just add new relationships
- No **role explosion**



Recap

Pattern	How ReBAC does it	Where Roles & Tokens break
Delegated Authorization	Agent holds its own grant, bounded to its human via delegator	Copy the human's roles onto the agent — a snapshot that drifts and over-grants
Expiring Grants	agent_deployer: agent with expiration — server-side, self-cleaning	Tokens live till they expire; a cron leaves a window where a dead grant still works
Instant revocation	Delete one relationship — the next check is a live lookup	Minted tokens stay valid; RBAC means cleanup in N places, hoping you got them all
Hierarchical contingencies	agent_deploy = agent_deployer & gated_by→agent_deployer	No role can depend on holding another role — you drop into app code

More Reading

- goose docs
 - goose-docs.ai
- Read the Zanzibar paper
 - zanzibar.tech
- Working Demo Link
 - github.com/sohanmaheshwar/goose-spicedb-delegation

Thank You!

Sohan Maheshwar

@AuthZed

AAIF Ambassador



Sohan Maheshwar

AAIF Ambassador | Experienced DevRel
professional

