AI AGENT SECURITY fwd:cloudsec Europe

# What we learned giving an AI agent read access to our whole Kubernetes fleet

An experience report.

**Amine Raji**
Cloud & Autonomous-Systems Security

# The choice was never agent or no agent

**BEFORE**


personal credentials · often write-capable
no scope · no audit · no rotation · invisible

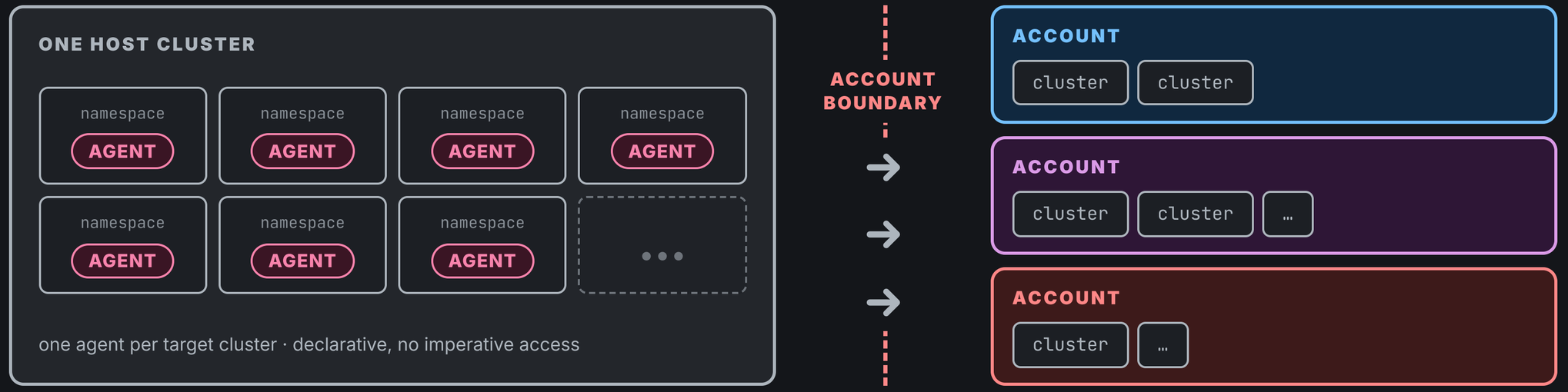
**AFTER**


**PRODUCTION**

one deployment per cluster · declarative, reviewed
every write path gated by a human

*One governed agent, or shadow agents everywhere. We picked the one we could put an identity on.*

# What we built




*Every agent pod runs in one host cluster and reaches across an account boundary. Nothing runs inside the clusters it watches.*

It went through review on one word.

✓ read-only

# Read-only is not a security boundary.

You do not bound an agent by what it is allowed to do.
You bound it by what it can reach, as whom, over what path,
and where a human has to sign.

**The agent has never written anything to a cluster. It broke four times anyway.**

01 Confidentiality
02 Attribution
03 Confused deputy
04 Availability

# Failure 1 · Confidentiality

the agent's tool server — VERBATIM

```
env {
  name  = "ALLOW_SENSITIVE_DATA_ACCESS"
  value = "true"   # it is needed for events reading
}
```

**WHAT WE ASKED FOR**

Pod **events** and **logs**. Which pod restarted, and why.

**WHAT THE SAME FLAG UNLOCKS**

**Secret objects.** One flag, both. The tool has no setting for events yes, secrets no.

*One flag. The tool cannot express events yes, secrets no.*

# Failure 1 · What held

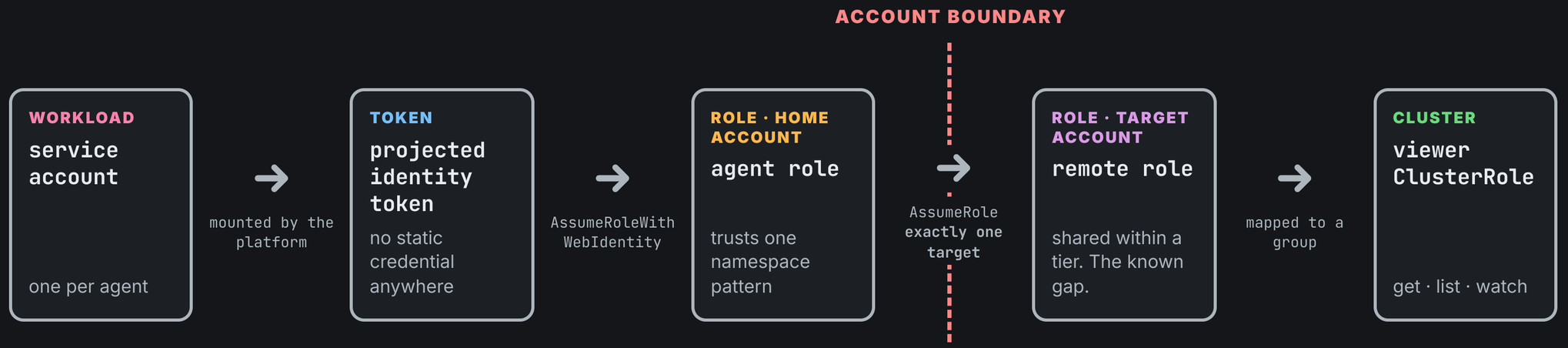

**11** of 16

stripped at the server.
**Five remain.**

W = mutating tool upstream.
Four stripped. The fifth is registered read-only.
A stripped tool never enters `tools/list`.

*You cannot call a tool that was never registered.*

# Failure 2 · Attribution




*Read-only tells you nobody wrote anything. It does not tell you who read.*

# Failure 2 · What held

the agent's cloud config

```
# every agent namespace gets the same file
[profile home]
role_arn                = <home-account>:role/agent-role
web_identity_token_file = /var/run/secrets/.../token

[profile a]      role_arn = <account-a>:role/RemoteRole
[profile b]      role_arn = <account-b>:role/RemoteRole
[profile prod]   role_arn = <account-c>:role/RemoteRole
[profile ...]    role_arn = ...
```

↑ **Every agent's config file lists every cluster, including production.**

## Any of them can ask.

**The one that is not entitled gets refused by IAM.**

Not by the config file.
Not by the model.

*The config file is not the control. It never was.*

# Failure 3 · The confused deputy

```
human types in chat    "check checkout-qa"
model fills the parameter    cluster = "checkout-qa"
the API needs the real name    "checkout-qa—<domain>"
 sometimes the model just guessed
```

**This is the oldest bug in cloud, wearing a new hat.**
**It is trusting a caller-supplied tenant identifier.**

*Read-only, on the wrong cluster, is still the wrong data.*

# Failure 3 · What held

the agent's tool server

```
def patched_get_client(self, cluster_name):
    return _orig_get_client(self, os.getenv('CLUSTER_NAME') or cluster_name)
```

↑ **The model's parameter is discarded, silently, below the tool schema.**

~~identity from the **caller**~~ → identity from the **workload**

*The model can ask about a cluster. It cannot choose which one it is talking to.*

# Failure 4 · Availability

*“please be concise”*

**A REQUEST**

the agent's runtime config

```
tool_output = { max_lines = 800, max_bytes = 40000 }
compaction  = { auto = true, prune = true }
resources   = { limits = { memory = "2Gi" } }
```

**A CONTROL**

*Read-only says nothing about volume.*

# So what is the boundary?

## 01

FAILURES 1 + 4

### Scoped capability

Tools removed at the server, not requested of the model. And a cap on what comes back.

- `server-side deny list`
- `no write flag`
- `output budget`

## 02

FAILURES 2 + 3

### Per-agent identity

Its own service account, its own scoped role, its own name on every trace.

- `workload identity`
- `one assume target`
- `per-agent attribution`

## 03

FAILURE 1

### Egress as capability

Default-deny both directions. Enforced in two layers, because one of them will have a hole.

- `default-deny policy`
- `fetch tools unregistered`

## 04

FROM NOTHING

### Human gates on writes

Anything that mutates state leaves agent autonomy and enters an approval workflow.

- `no write path exists`

**ON MONDAY**

01 Count the tools that can write

02 Find what identity it runs as

03 Check if it can reach the internet

04 Find the write path, and who signs

*Four rungs. Each one is a control you already know how to build.*

# Nothing here is AI security

**RUNG 01 · SCOPED CAPABILITY**

The boring control: **deny-by-default access control and tool allow-lists**, enforced at the server. You have written a hundred of these.

**RUNG 02 · PER-AGENT IDENTITY**

The boring control: **workload identity**. One service account per workload, scoped role, rotated token, audit log. Same as any service.

**RUNG 03 · EGRESS AS CAPABILITY**

The boring control: **egress policy**. Default-deny, then explicit allows. Two layers, because one of them will have a hole.

**RUNG 04 · HUMAN GATES ON WRITES**

The boring control: **change management**. Mutating state exits agent autonomy and enters an approval workflow.

# What we got for it

**TIME**

The agent does the **first ten minutes** of every investigation. Gathering, correlating, summarising, before a human looks at it.

**SHADOW AGENTS**

Engineers stopped asking for exceptions to wire their own tools to production. **An observation, not a measurement.**

**AUDIT**

Every tool call, every investigation, attributable to a named agent. Something the shadow-tool world could **never** offer.

# Read-only is not a security boundary.

**THE ONE CONTROL WE HAVE NOT BUILT**

Redaction between tool output and the model, so secret values never enter its context.
We know the design. We have not shipped it. If you have, come and find me.


linkedin.com/in/araji

**Amine Raji** @aminerj