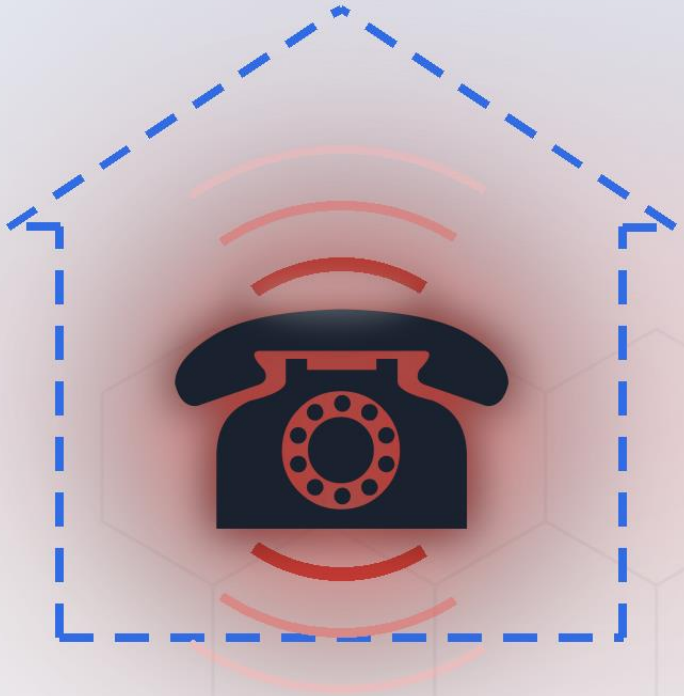
KubeCon | CloudNativeCon
North America 2026

SALT LAKE CITY | NOVEMBER 2026

The Operator Calls Are Coming From Inside the Namespace



Confused deputy flaws in Kubernetes cloud operators

Kubernetes | Google Cloud | AWS | Microsoft Azure

Justin O'Leary | olearysec.com | @olearysec | @olearysec@infosec.exchange

FIRST · WHO IS THE ATTACKER?

The “namespace user”, and you already have hundreds.

Any identity that can `kubectl apply` in one namespace. No console, no service-account key, **zero cloud IAM.** Pure Kubernetes, every platform. The operator holds the cloud keys.

USER · GROUP

Developers & tenant teams

A namespace with edit/admin RBAC per team. The users come from your cloud's own directory.

✗ zero cloud IAM

CONTROLLER SA

CI/CD & GitOps bots

Argo CD, Flux, GitHub Actions apply whatever's in Git. A malicious PR is a kubectl apply.

✗ zero cloud IAM

POD SA + TOKEN

Every running pod

Every pod mounts a ServiceAccount token by default. One RCE and it's the attacker's.

✗ zero cloud IAM

USER

Contractors & on-call

Anyone given scoped kubectl to a shared cluster: vendors, contractors, SREs, interns.

✗ zero cloud IAM

SAME USER, EVERY PLATFORM

GKE · Google groups | EKS · IAM principals | AKS · Entra ID | any K8s · OIDC / certs

`namespace user` → `kubectl apply` → **the operator** (holds org-admin cloud creds) → **your cloud**

None of them have cloud access. That's the whole point of the operator. **Any one that can create its CRDs inherits its reach.** Every multi-tenant cluster running one, on any cloud, already has dozens.

YOU DID EVERYTHING RIGHT.

Every box your auditor asked for. Ticked.

- ✓ **RBAC locked down**
  Roles and bindings scoped to exactly what each team needs.
- ✓ **Namespaces scoped**
  Every workload fenced into its own namespace.
- ✓ **Network policy in place**
  Nothing talks to anything it shouldn't.
- ✓ **Least privilege everywhere**
  Service accounts hold only what they need.

Every control working exactly as designed.

Everyone runs one.

Operators that turn Kubernetes YAML into cloud API calls, using their own credentials.

Config Connector
Google Cloud

ACK
AWS

ASO
Microsoft Azure

Crossplane
Multi-cloud

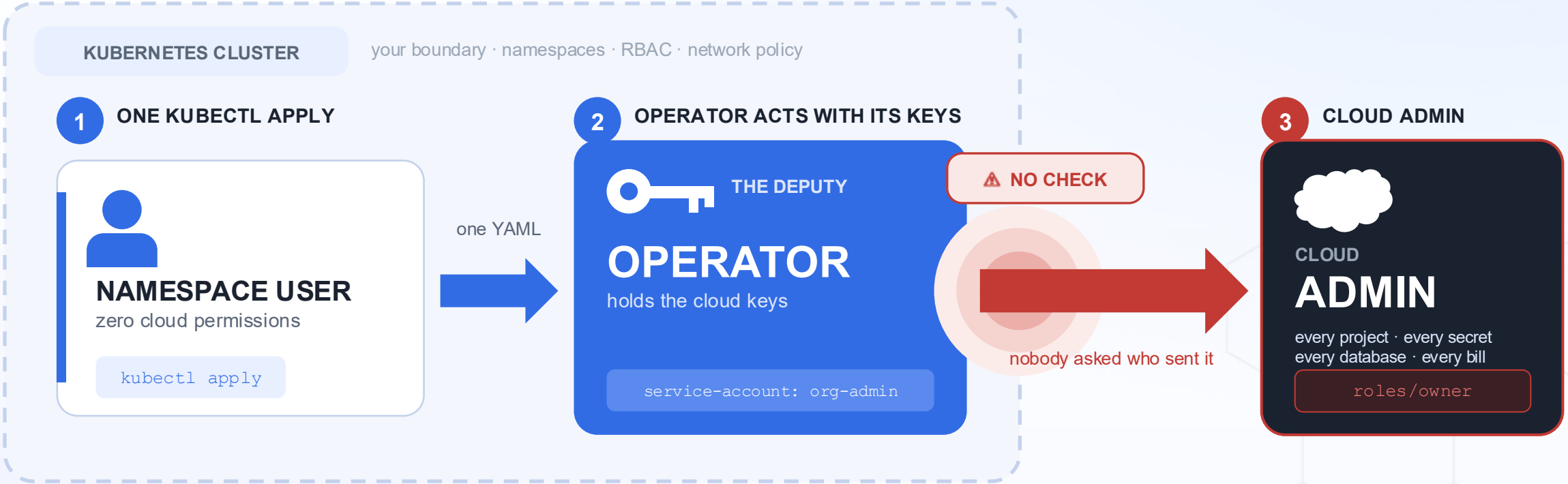
External Secrets
Any secret store

kubectl apply → operator reads it → **cloud API call, with the operator's credentials**

Almost no one checks what it'll do for the wrong person.

THE CALL IS COMING FROM INSIDE THE CLUSTER

The attacker never left their namespace



The boundary held. **The call still got out.** Through your own operator.

DEMO 1 · GOOGLE CLOUD

CONFIGCONFUSION · CVSS 9.9 · UNPATCHED

Config Connector

GCP's own operator. Kubernetes YAML in, GCP resources out.



GKE

Real cluster. Config Connector installed per the GCP docs.

0

Cloud permissions for our namespace user. One namespace.

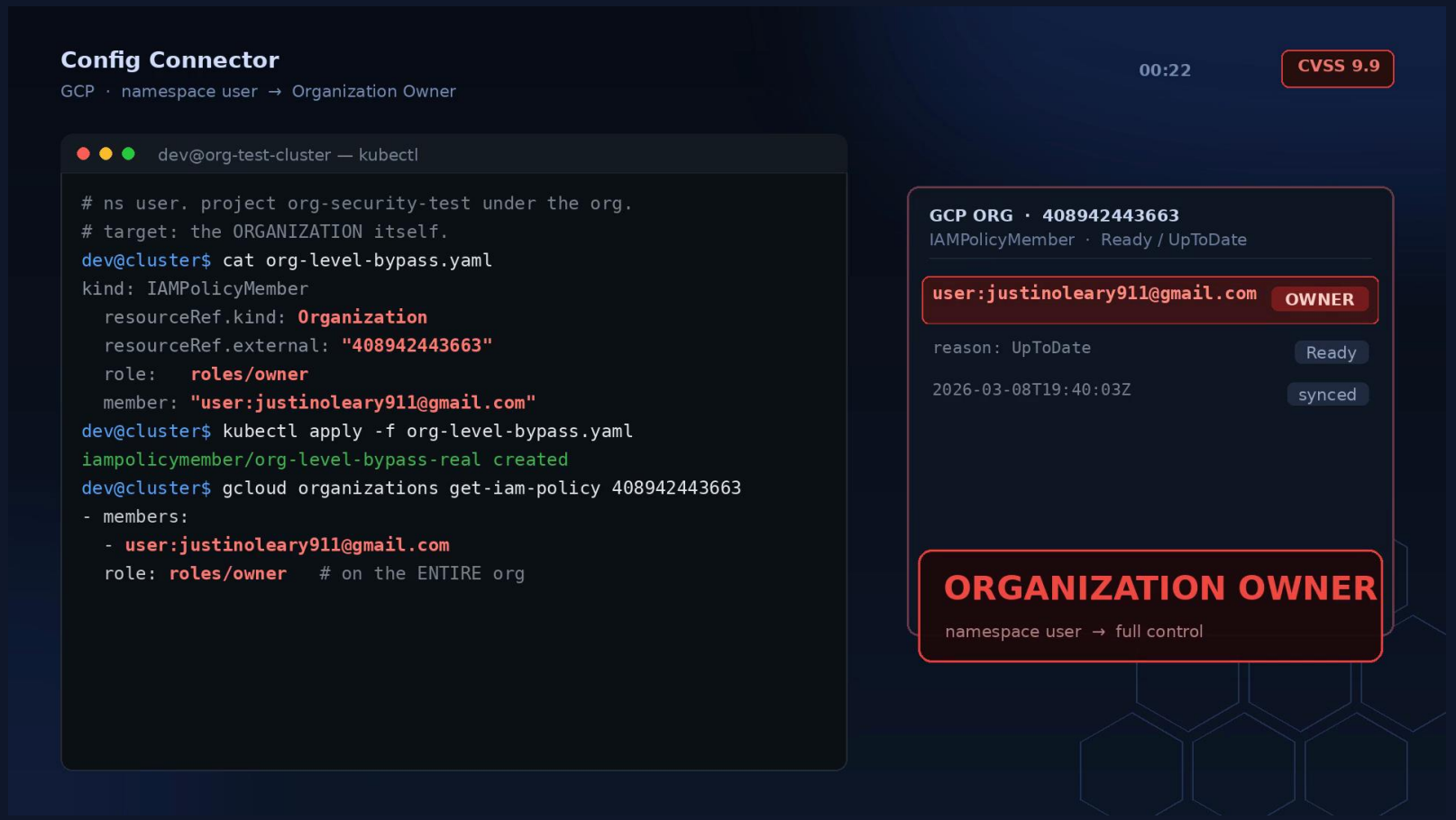
1

YAML file. An IAMPolicyMember, pointed at the organization.

<60s

Start to finish.

Watch for: who sends the request, and whose credentials make the call.



WHAT YOU'RE WATCHING

1 I start as a namespace user with zero GCP permissions. I can't touch org IAM.

2 I apply one IAMPolicyMember YAML that targets the whole organization.

3 Config Connector holds org admin rights. It sends my request to GCP as itself and never checks who asked.

4 I'm now Organization Owner, with access to all the projects and secrets underneath.

One YAML from one namespace makes me owner of the entire GCP organization.

▶ **Recorded demo · namespace user to GCP Org Owner**

THE ACTUAL BUG · CONFIG CONNECTOR

AS SHIPPED · NO CHECK

ATTACKER INPUT
their org ID, used as-is

externalonly.go — simplified

```
UnstructHandler: func(ref) {

    spec["org_id"] = ref.External

    return spec    // → GCP. no check.
}
```

the check goes here
there isn't one, here or anywhere upstream

add one check

PROPOSED FIX · ONE GUARD CLAUSE

proposed fix — one guard clause

```
UnstructHandler: func(ref, caller) {

    // may the caller touch THIS org?
    allowed := caller.Can(
        "resourcemanager.organizations.setIamPolicy",
        ref.External)

    if !allowed {
        return errUnauthorized    // fail closed
    }

    spec["org_id"] = ref.External
    return spec
```

THE CHECK
allowed? no → refuse

The whole fix: **check who's asking, then fail closed.** The same move GCP already shipped for Cloud Run.

GOOGLE WROTE THE CHECK, JUST NOT HERE

lbrouteextension_controller.go

```
// the reference must live in the same project
refProject := common.ExtractProjectID(ref.External)

if refProject == "" || refProject != projectID {
    return nil, fmt.Errorf(
        "cross-project references are not supported")
}
```

READS THE REFERENCE
Pulls the project ID out of the caller's reference, before doing anything with it.

CHECKS, THEN REFUSES
Not its own project? It returns an error and does nothing. A guard that fails closed.

Same operator. Same kind of reference. Only one path is guarded.

networkservices extensions	reference check ✓
IAM · org / billing acct	no check ✗

They know how to write this check. They just never wrote it on the path that grants Organization Owner.

THE TIMELINE · EVERY DATE IS PUBLIC GIT HISTORY

Reference checks are newer than my bug report.

MAR 8 2026
I report it.
Namespace user → GCP Organization Owner.
VRP #490876526

MAR 22 2026
First reference-check in Config Connector's history.
On a load-balancer controller, not the IAM path.
+14 DAYS · commit 4192d13

APR 7 2026
Google: "working as intended."
No bounty.
VRP panel

MAY 1 2026
Google refuses a CVE.
"We do not believe it is a vulnerability."
Alphabet CNA

JUN–SEP 2026
The check ships in three more new controllers.
authz · edge · traffic extensions
bcbe473 · 481a3bd · 92c23f1

TODAY
The IAM path still has no check.
`externalonly.go` · org_id = ref.External · untouched since 2022
4 YEARS · 0 CHECKS

Zero controllers validated a reference before my report. Every one built since does. The IAM path still doesn't.

`$ git log` every hash is public. verify it yourself `github.com/GoogleCloudPlatform/k8s-config-connector`

IT DOESN'T STOP AT ONE BINDING

IAMPolicy is the **authoritative** one. One apply doesn't add a binding. It **replaces the entire org policy.**

iampolicy_controller.go :302

```
// set the etag to an empty string, since
// IAMPolicy is the authoritative intent,
// KCC wants to overwrite the underlying
// policy regardless
policy.Spec.Etag = ""
```

GOOGLE'S OWN COMMENT
“overwrite the underlying policy regardless”. Their words, in the source.

THE SAFETY CATCH, EMPTIED
The etag is GCP's concurrency guard: apply only if nothing changed. Empty = skip it, overwrite everything.

BEFORE 50+ owners
CTO · CISO · security · finance · every dev



AFTER you alone
everyone else deleted. They can't even log in to undo it

Not escalation. A lockout.
Recovery means Google Support, re-proving you own the domain, days to weeks.

One kubectl apply → sole Organization Owner, and everyone who could stop you is gone.

THE ATTACKER IS INVISIBLE

Your logs won't save you

WHAT KUBERNETES SEES



`system:serviceaccount:dev:attacker`

created an **`IAMPolicyMember`** in its namespace

```
$ kubectl apply
```

on the Kubernetes side, a routine resource create

≠

different identity

WHAT GCP SEES

GCP CLOUD AUDIT LOG

```
{
  "principalEmail": "config-connector@org-security-test…"
  "methodName"    : "SetIamPolicy"
  "resourceName"  : "organizations/408942443663"
  "callerIp"      : "35.192.x.x"   // a GKE node
}
```

Two logs. Two systems. Nothing links them.

The cloud log blames the operator's service account. Forensics chases an account that did nothing wrong, and the real actor never appears on the GCP side.

DON'T TAKE MY WORD FOR IT

Google's own docs told you to grant org-owner.

Their docs said it. Their code shipped it in 2020. The Register reported it. **All public.**

GOOGLE'S OWN DOCS · archived Jul 14, 2026

Organization owner permission

To expand Config Connector's permissions so it can manage all projects and folders for a given organization, complete the following steps:

1. Create the following YAML manifest:

```
apiVersion: iam.cnrm.cloud.google.com/v1beta1
kind: IAMPolicyMember
metadata:
 name: iampolicymember-orglevel-permission
 namespace: NAMESPACE
spec:
 member: serviceAccount:SERVICE_ACCOUNT_NAME@HOST_PROJECT_ID.iam.gserviceaccount.com
 role: roles/owner
 resourceRef:
   kind: Organization
   external: ORGANIZATION_ID
```

A whole section, "Organization owner permission", telling you to grant `roles/owner` on the org.

THE RECEIPTS

THE DOCS · Jul 14, 2026
scroll to the use case

THE FEATURE · 2020
github.com · issue #276

The Register

June 18, 2026

"Google Told Researcher 'Nice Catch!' Then Denied Bug Bounty for Flaw It Still Hasn't Fixed"

Google, to The Register:
"…goes against Google Cloud's **publicly shared best practices**."

Read the full story
theregister.com

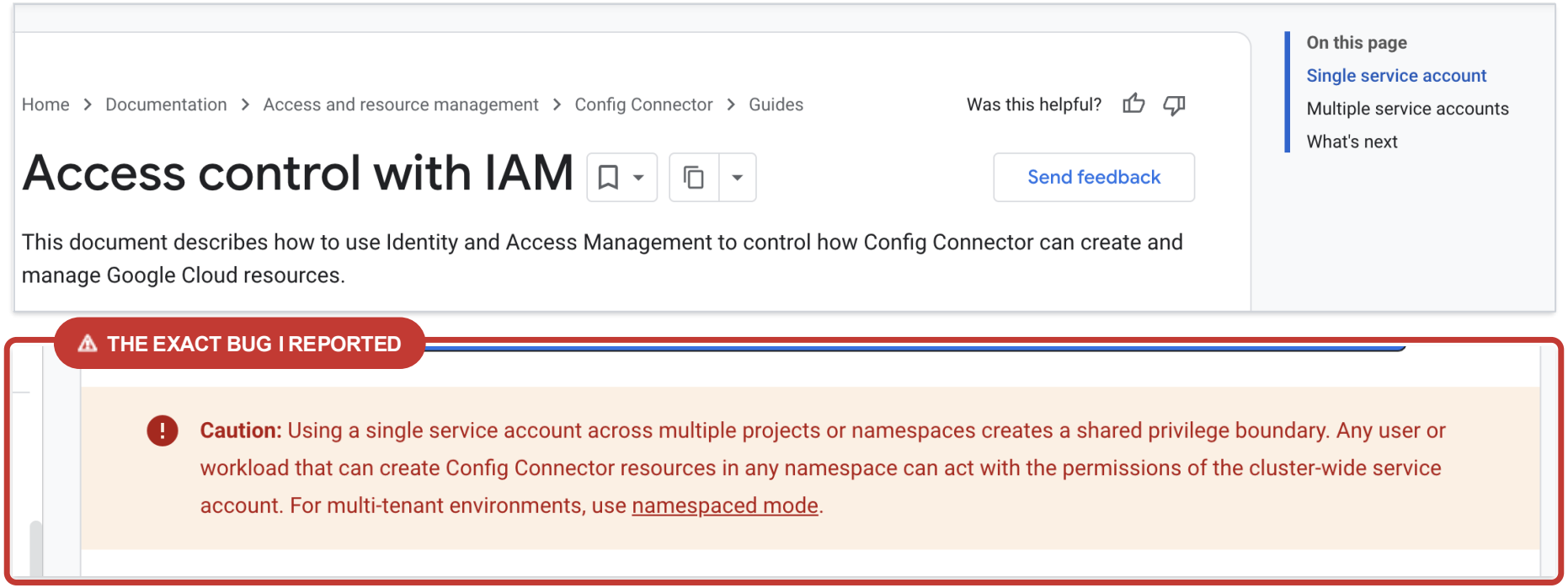
Google told The Register this "goes against best practices". Their own docs and code said otherwise, for six years.

SAME FLAW, NEW DOCS

Then, 15 days before this talk, they deleted it.

Same page, same URL, **rewritten Oct 26, 2026. Everything that told you to grant org-owner is gone.**

AFTER · live page · docs.cloud.google.com/config-connector … configure-iam-permissions



Six sections deleted. One warning added. Zero lines of code fixed. They patched the story, not the software.

ALREADY SOLVED · TWICE

Google has fixed this exact class, two different ways.

The two ways to kill a confused deputy: both shipped in production, both skipped for Config Connector.

FIX A · ADD THE CHECK

ImageRunner

Cloud Run · Jan 2025

An identity with Cloud Run revision edit rights, and no registry access, made Cloud Run pull a private image.

Fix: Cloud Run now validates the deployer has read access to the image before acting. Shipped Jan 28, 2025 as a breaking change.

→ **Verify the caller.**

SCAN · TENABLE REPORT

FIX B · REMOVE THE DEPUTY

ConfusedComposer

Cloud Composer · Apr 2025

With only composer.environments.update, an identity installed PyPI packages through the over-privileged Cloud Build SA.

Fix: PyPI installs now run as the environment's own service account, not the default Cloud Build SA. Composer 3 was never exposed.

→ **Shrink the deputy.**

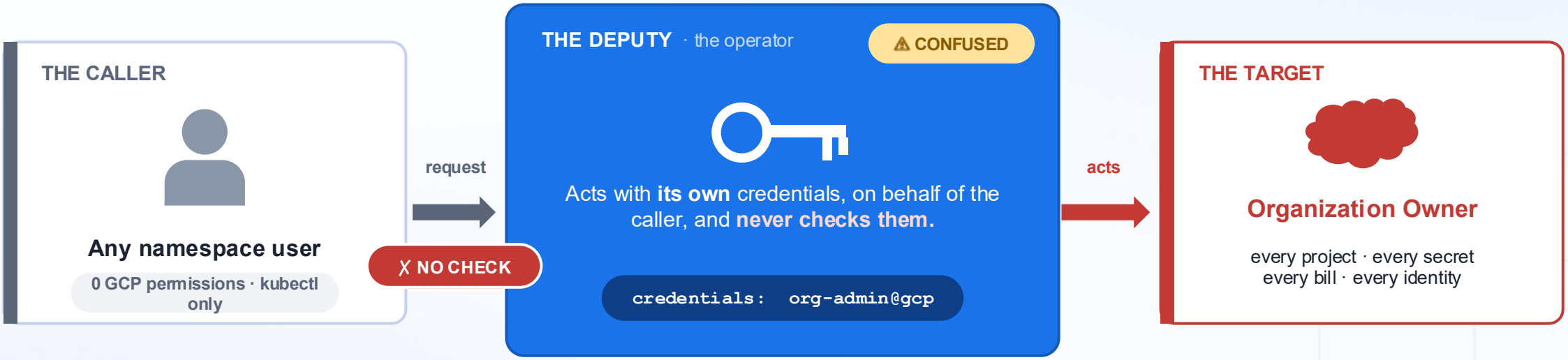
SCAN · TENABLE REPORT

Google owns both fixes. It shipped both for Tenable's reports.

For Config Connector, it has shipped neither. The one-line check still isn't there. The fix isn't hypothetical; it has already shipped, twice.

This is a confused deputy (CWE-441)

Norm Hardy named it in 1988. Thirty-eight years later, it runs your cloud.



The vulnerability is the missing check: **the deputy never verifies the caller is allowed before it acts with its keys.**

The deputy has the keys. You just tell it which door.

WHY THIS IS EVERYWHERE

Every cloud operator does the same thing.

A **user-controlled reference** → the operator's **cluster-wide credentials** → no check on the caller. = **confused deputy.**

Elastic Cloud on Kubernetes

CVE-2026-72640 · CWE-441

Trusts a **namespace annotation** and copies Secrets across namespaces.

✓ Fixed · ECK 3.5.0

Aiven Operator

CVE-2026-39961

A ClickhouseUser **names a namespace**, and the operator hands over its Secret.

✓ Fixed · v0.37.0

Red Hat OpenShift AI

CVE-2026-16456

loadSecret reads a **user-supplied namespace** and leaks API keys & cloud creds.

✓ Fixed · RHSA erratum

Red Hat ACM

CVE-2026-64927

A Channel's `secretRef.namespace` points anywhere, including other namespaces' Secrets.

✓ Fixed · RHSA erratum

Config Connector · GCP, run by the cloud vendor

A namespace user points an IAMPolicyMember at the **organization** and becomes **Organization Owner.**

✗ NO CVE

✗ UNPATCHED

Four independent teams found this shape, filed a CVE, and shipped a fix. The cloud vendor's own operator did neither.

Every patch did the same thing. Stop trusting the caller's namespace.

Four vendors, four codebases, one idiom. Every diff is public git history.

THE BUG · the same idiom in all four

```
1  ns := ref.Namespace                  // the caller typed this
   if ns == "" {
2      ns = resource.Namespace          // fallback ONLY if blank
   }
3  client.Get(ctx, NamespacedName{ns, ref.Name}, secret)
```

Verbatim in ACM pre-fix. Still on the public main branch of odh-model-controller (loadSecret).

THE FIX · Red Hat ACM, PR #535, merged Aug 12, 2026

```
4  // Referenced Secrets must live in the Channel's own
   // namespace. Ignore any caller-supplied namespace so a
   // tenant cannot aim our privileged client elsewhere.
5  srtRef.Namespace = instance.GetNamespace()
6  // one assignment. no SubjectAccessReview needed.
```

stolostron/multicloud-operators-channel · channel_controller.go · handleReferencedObjects()

WHAT WAS WRONG

1. **The reference carries a namespace, and the caller wrote it.** Anyone who can create the resource can fill it in.
2. **The operator only falls back to its own namespace if that field is blank.** Fill it in, and it is honored as-is.
3. **The operator fetches the Secret with its own cluster-wide credentials.** The caller's limits never enter the picture.

WHAT THEY CHANGED

4. **Decide the rule up front:** a referenced Secret must live where the resource lives. Nowhere else.
5. **Overwrite the caller's namespace with the resource's own.** One assignment. The caller can now only reach what they already could.
6. **No permission check needed.** The operator simply refuses to act outside the caller's scope.

ACM clamped it. Aiven deleted the field. Elastic allow-listed it. Not one needed a SubjectAccessReview.

They stopped letting the caller choose the scope. **Config Connector still lets a namespace user choose the organization.**

DEMO 2 · MICROSOFT AZURE

KINGMAKER · CVSS 9.9 · UNPATCHED

Azure Service Operator

Microsoft's operator. Kubernetes YAML in, Azure resources out.



Full write-up
Scan for the complete research
olearysec.com

AKS

Real cluster. ASO installed per Microsoft's docs.

0

Azure credentials for our namespace user. One namespace.

4

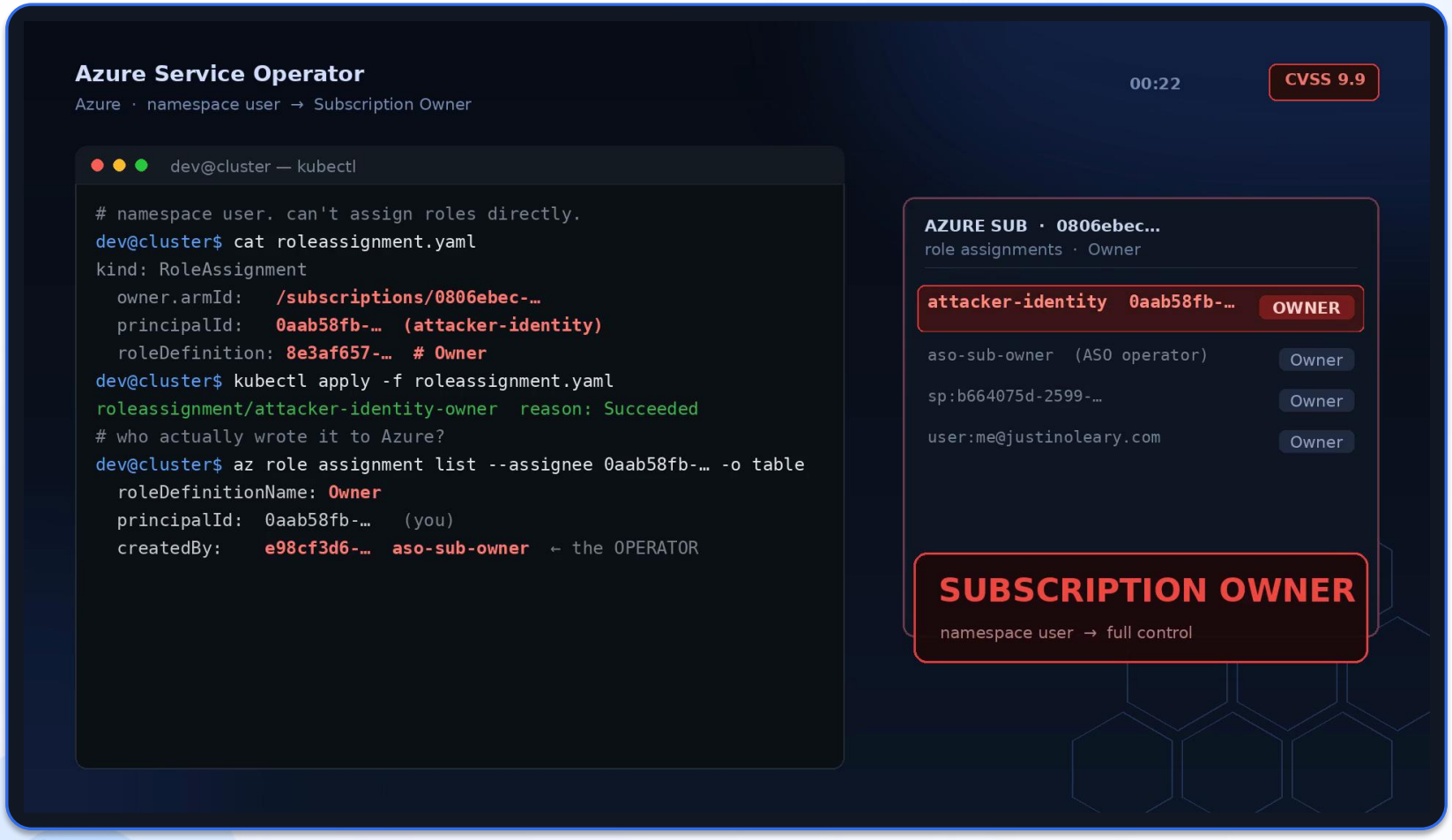
YAMLs: identity, role, federation, service account.

<60s

Start to finish.

Watch for: who sends the request, and whose credentials make the call.

AZURE · ASO · LIVE DEMO



▶ Recorded PoC · real IDs, real API receipts

WHAT YOU'RE WATCHING

1 I create my own managed identity. It has no Azure permissions yet.

2 I apply one RoleAssignment that makes it Owner of the whole subscription. ASO grants it without checking if I'm allowed.

3 I link that identity to a pod I control.

4 I sign in with az login as Subscription Owner. Azure's logs show ASO made the grant, not me.

Four YAMLs in one namespace give me full control of the subscription.

THE ACTUAL BUG · AZURE SERVICE OPERATOR

resolver.go · ResolveOwner() · Azure Service Operator (trimmed)

```go
func (r *Resolver) ResolveOwner(ctx, obj) (OwnerDetails, error) {
    owner := obj.Owner()

    if owner.IsDirectARMReference() {                          // 1
        return OwnerDetailsFromARM(owner.ARMID), nil           // 2
    }
    // otherwise, resolve the owner through Kubernetes
}
```

1 Your input

A raw ARM ID you supplied: the scope you named in `owner.armId`.

2 No check

ASO returns it as-is and assigns the role with its **own Owner identity**, without checking you're allowed at that scope.

```
$ git blame v2/internal/resolver/resolver.go · line 261

d1c7f88  2023-09-04  return OwnerDetailsFromARM(owner.ARMID), nil

↳ introduced by PR #3245: "Support arbitrary ARM ID owners."
```

3 YEARS
unchanged · live at HEAD

THE MISSING CHECK

"Can this user assign roles on subscription X?"

ASO resolves your refs, secrets, configmaps, but never your Azure RBAC. It just acts.

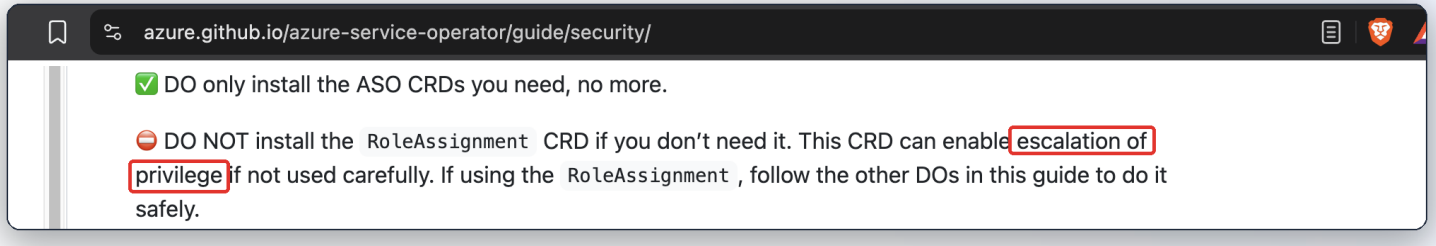
THE FIX · one guard clause, before it acts

Check the caller holds `Microsoft.Authorization/roleAssignments/write` at that scope.
No? Fail closed. Capture the real caller at admission, never from an annotation.

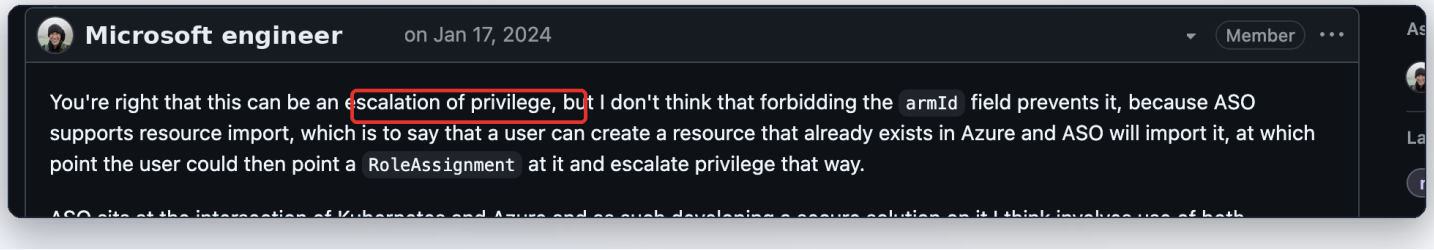
THE CONTRADICTION

Microsoft's docs and its own engineer call this escalation of privilege. MSRC told me it isn't.

① IN MICROSOFT'S LIVE DOCS



② THEIR OWN ENGINEER · PUBLIC ISSUE #3645, JAN 2024



JAN 2024 · THEY KNEW

Their engineer, on public issue #3645: "…escalation of privilege."

APR 2026 · THEY DENIED ME

MSRC Case 111735: "…does not demonstrate privilege escalation."

AUG 2026 · DOCUMENTED IT

PR #5639 adds that warning + a new Confused Deputy page. 4 months later.

They denied it to me, then documented the exact risk for everyone else. Still no CVE, still no fix.

Verify: MSRC Case 111735 · github.com/Azure/azure-service-operator/issues/3645 (open since 2023) · warning + Confused Deputy page added in PR #5639, commit 01dbe75a1, Aug 25 2026. Next slide shows how to check it yourself.

THE RECEIPTS

Don’t take my word for any of it. Run one command, scan one code.

Everything on the last two slides lives on Microsoft’s own `main` branch. Here’s how to confirm it in 30 seconds.

zsh · verifying Microsoft’s own repo

```
$ git clone https://github.com/Azure/azure-service-operator
$ git log -S'escalation of privilege' \
      -- docs/hugo/content/guide/security/_index.md

01dbe75a1  Improve security documentation (#5639)
          Date: Aug 25 2026   ← 4 months after my report
→ the ONLY commit that ever added that phrase.
```

full commit SHA `01dbe75a15ccf3243334aff8a86f212033811c50`

What PR #5639 added to the docs: the diff:

```
+  DO NOT install the RoleAssignment CRD … can enable escalation of privilege.
+  guide/security/confused-deputy/_index.md     ← brand-new page (0 → new file)
```

Before this commit: zero mentions of “escalation of privilege” or “confused deputy” anywhere in the docs.

SCAN IT YOURSELF

straight to GitHub, no middleman

The commit, live
PR #5639 · “Files changed”
`…/pull/5639/files`

They knew in 2023
Public issue #3645
`…/issues/3645`

You don’t even have to trust the date: the phrase isn’t in the commit before this one, and public issue #3645 predates my report by two years. It’s all on Microsoft’s `main` branch. Clone it and check.

Reproduce: git log -S'escalation of privilege' -- docs/hugo/content/guide/security/_index.md → 01dbe75a1 (PR #5639, Aug 25 2026) · github.com/Azure/azure-service-operator/issues/3645 (opened Dec 2023) · MSRC 111735

DEMO 3 · AWS

TRUSTFALL · CVSS 9.9 · UNPATCHED

AWS Controllers for Kubernetes

AWS's official operator framework. Kubernetes YAML in, AWS resources out.



Full write-up
Scan for the complete research
olearysec.com

EKS

Real cluster. ACK installed per AWS's docs.

0

AWS credentials for our namespace user. One namespace.

1

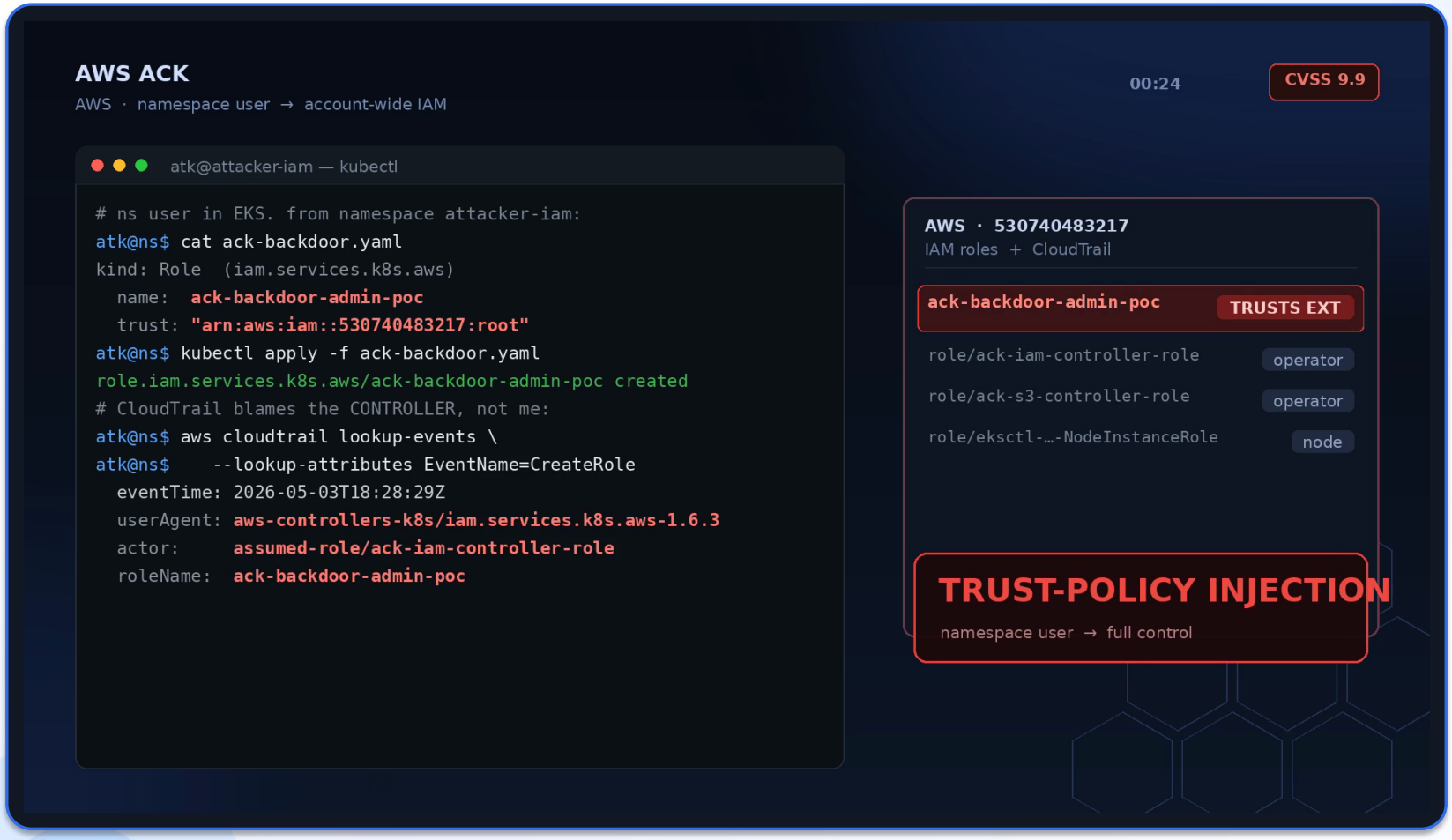
One YAML: an IAM Role with a trust policy.

<60s

Start to finish.

Watch for: who makes the call to AWS, and whose identity CloudTrail records.

AWS · ACK · LIVE DEMO



▶ Recorded PoC · real IDs, real API receipts

WHAT YOU'RE WATCHING

1 I'm a namespace user in the EKS cluster with no AWS admin rights.

2 I apply an admin Role whose trust policy lets my own outside AWS account assume it.

3 ACK's controller creates that role in AWS as itself. It never checks the trust policy.

4 CloudTrail shows ACK as the one who did it, not me. The access survives key rotation.

One YAML from a namespace leaves a cross-account admin backdoor that the logs blame on ACK.

THE ACTUAL BUG · AWS ACK

role/sdk.go · newCreateRequestPayload() · ACK IAM Controller (trimmed)

```
func newCreateRequestPayload(r) (*CreateRoleInput, error) {
    res := &svcsdk.CreateRoleInput{}

    if r.ko.Spec.AssumeRolePolicyDocument != nil {
        res.AssumeRolePolicyDocument = r.ko.Spec.AssumeRolePolicyDocument
    }
    // ... name, path, tags: copied straight through
}  →  rm.sdkapi.CreateRole(ctx, input)
```

1 Your input

The trust policy you wrote in `spec.assumeRolePolicyDocument`. You can name any AWS account, even your own.

2 No check

ACK copies it verbatim into `CreateRole` and signs the call with its **own privileged IAM identity**. It never parses the principal.

```
$ git blame pkg/resource/role/sdk.go · line 354
00de17f  2025-02-05  res.AssumeRolePolicyDocument = r.ko.Spec.…
↳ no validating webhook anywhere in the repo. The trust policy is never parsed.
```

LIVE
v1.11.3 · no fix

THE MISSING CHECK

“Is this caller allowed to trust that principal?”

ACK never parses the policy. Recommended IAM: 60 actions, Resource *.

THE FIX · before it calls CreateRole

Parse `assumeRolePolicyDocument`; reject principals outside your org.

Fail closed. AWS ships neither the check nor a webhook.

AWS documented the confused deputy as a feature.

I read every page AWS points you to. Not one protects against this. Most of them walk you into it.

ACK PERMISSIONS DOCS

“The Kubernetes user that makes a Kubernetes API call… has no association with an IAM role.”

That’s the confused deputy, written as the design, with no warning. And “least privilege” for the IAM controller still means: create any role.

✗ IT’S THE DESIGN, NOT A WARNING

EKS MULTI-TENANCY GUIDE

Pages of OPA/Gatekeeper & Kyverno policy, and every example is for pods, nodes, network.

Zero examples for ACK, or for validating a trust policy. Deploy exactly what this guide recommends, and TrustFall still works.

✗ SOLVES THE WRONG THREAT

AWS’S CONFUSED-DEPUTY PAGE

“…an entity that doesn’t have permission… can coerce a more-privileged entity…”

AWS wrote the definition of my bug. Their fixes (External ID, aws:SourceArn) are cross-account & cross-service. None apply to ACK.

✗ A DIFFERENT CONFUSED DEPUTY

Google rewrote the pages. Microsoft added a warning. AWS documented the confused deputy as a feature, then told me to file a GitHub issue.

Sources: aws-controllers-k8s.github.io/…/authorization · docs.aws.amazon.com/eks/latest/best-practices/tenant-isolation.html · docs.aws.amazon.com/IAM/latest/UserGuide/confused-deputy.html

I reported a 9.9. AWS told me to “file a GitHub issue.”

It’s AWS’s own repo. The code is still live in the current release. And no IAM policy they recommend can stop it.

```
zsh · verifying AWS’s own repo

$ git blame pkg/resource/role/sdk.go  # line 354
00de17f  res.AssumeRolePolicyDocument = r.ko.Spec.…  live in v1.11.3
$ grep -rl ValidatingWebhook .   → (none)
          ACK never parses the trust policy.
```

You can’t scope your way out. No IAM policy stops this:

✗	**Recommended IAM:**	60 actions · Resource “*” · CreateRole on any role.
✗	**Scope to role/ack-*:**	Attacker names it ack-backdoor. The pattern matches.
✗	**Check the trust policy?**	No IAM condition key can inspect the principals.

SCAN IT YOURSELF

straight to AWS’s repo, no middleman

The live code
sdk.go · git blame

Recommended IAM
Resource: “*”

Reported to AWS security on May 3, 2026. Response, May 5: “it’s open source, file a GitHub issue.”

No CVE. AWS’s vulnerability program excludes ACK. Unpatched as of v1.11.3. Six months later, the line is unchanged.

Verify: git blame …/role/sdk.go → 00de17f (line 354, live in v1.11.3) · config/iam/recommended-inline-policy → Resource "*" · IAM has no condition key for trust-policy principals (AWS Service Authorization Reference; AWS re:Post) · CWE-441

DEMO 4 · CNCF CROSSPLANE

UNBOUNDED · CVSS 9.9 · UNPATCHED

Crossplane provider-gcp

The vendor-neutral operator everyone trusts, CNCF graduated. We demo GCP; the identical flaw ships in every cloud provider.

Full write-up

Scan for the complete research

olearysec.com

GKE

Real cluster. provider-gcp installed per the docs.

0

GCP credentials for our namespace user. One namespace.

1

One YAML: an OrganizationIAMMember with a raw org ID.

<30s

Start to finish.

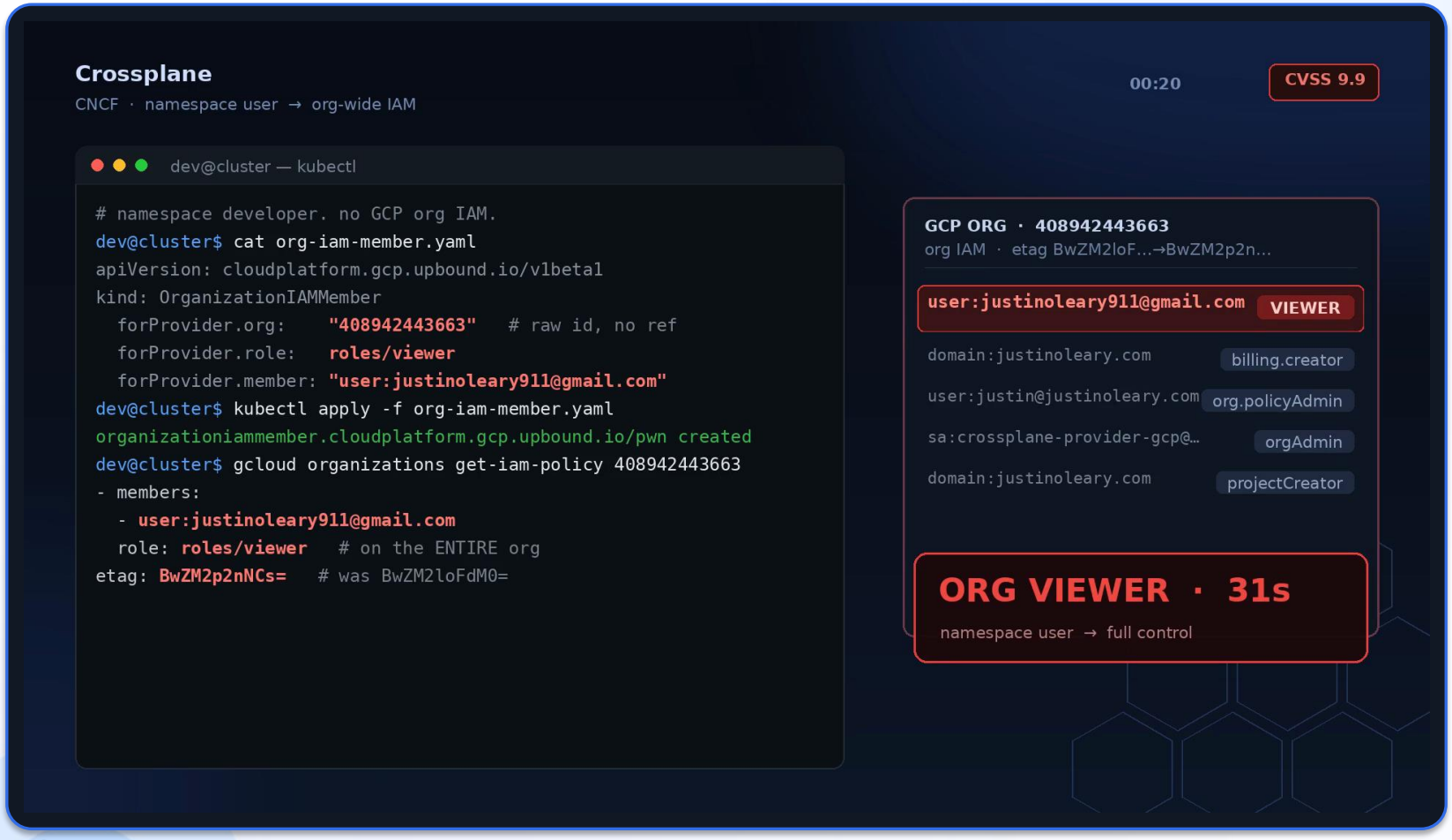
SAME CODE, EVERY CLOUD

provider-gcp | provider-aws | provider-azure | provider-azuread

OrganizationIAMMember · RolePolicyAttachment · RoleAssignment: the identical raw-string bypass, verified in each.

Watch for: who makes the call to GCP, and whose identity the org IAM policy records.

GCP · CROSSPLANE · LIVE DEMO



▶ Recorded PoC · real IDs, real API receipts

WHAT YOU'RE WATCHING

1 I'm a namespace developer with zero GCP permissions.

2 I apply a Crossplane OrganizationIAMMember using a raw org ID instead of a checked reference.

3 Crossplane writes my binding to the org's IAM policy with its own credentials. There's nothing to check me against.

4 Watch the etag change. The org's IAM policy really changed, 31 seconds after I applied it.

One YAML into a CNCF-graduated operator gives me access across the whole org.

Not theoretical. I ran it.

All three IAM CRDs, confirmed live on a real GKE cluster, March 12 2026 (org **408942443663**).

gke · crossplane-std-cluster · us-central1-a

```
# attacker = namespace user, zero GCP IAM
$ kubectl apply -f org-iam-member.yaml
organizationiammember.../org-bypass-poc created

# BEFORE   org 408942443663   etag BwZM2loFdM0=
#   user:justinoleary911@gmail.com  NOT FOUND
# AFTER    (+31s)              etag BwZM2p2nNCs=
  - members:
    - user:justinoleary911@gmail.com
    role: roles/viewer   ← on the ENTIRE org
```

The etag moved. A namespace user rewrote a real org's IAM policy in 31 seconds.

Tested on provider-gcp-cloudplatform v1.10.0 (v1beta1). Current release uses v1beta2. **Both versions vulnerable.**

OrganizationIAMMember

org 408942443663

9.9

✓ CONFIRMED

FolderIAMMember

folders/635039408468

8.5

✓ CONFIRMED

ProjectIAMMember

org-security-test

8.0

✓ CONFIRMED

+ member injection confirmed
grants to any identity the attacker controls, e.g. **attacker-test-sa**

Three caller-controlled strings. Nothing to check them against.

apis/namespaced/cloudplatform/v1beta1/zz_organizationiammember_types.go

```
// +kubebuilder:resource:scope=Namespaced    // namespace user owns it
type OrganizationIAMMemberParameters struct {
    OrgID  *string  `json:"orgId,omitempty"`   // raw string
    Role   *string  `json:"role,omitempty"`
    Member *string  `json:"member,omitempty"`
}

// group cloudplatform.gcp.m.upbound.io: no OrgIDRef,
// no ResolveReferences, nothing to check
```

CALLER CONTROLS ALL THREE

The namespace user fills in the org, the role (any role, e.g. roles/owner), and the member (themselves). All plain strings.

NOTHING TO CHECK IT

No OrgIDRef, no ResolveReferences. Nothing in-cluster to authorize the caller against. v2 made it namespaced; the “admin-only” assumption is gone.

MISSING CHECK: “Is this namespace user allowed to touch org 408942443663?”

The operator never asks. It takes the string and calls `SetIamPolicy()` with its own credentials.

They added the references, then made them optional.

All three IAM CRDs accept a **raw string**. Org has no reference field at all; folder and project have one, marked `+optional`.

OrganizationIAMMember — 9.9

```
OrgID  *string
  `json:"orgId"`

// no OrgIDRef field
// raw string only
```

✗ **No reference exists**

can't require what isn't there

FolderIAMMember — 8.5

```
Folder     *string  ← raw, works
+optional
FolderRef  *Reference
  ← skippable
```

⚠ **Ref exists, but optional**

raw folder bypasses it

ProjectIAMMember — 8.0

```
Project    *string  ← raw, works
+optional
ProjectRef *Reference
  ← skippable
```

⚠ **Ref exists, but optional**

raw project bypasses it

A reference you don't require isn't a guardrail. It's a suggestion the attacker declines.

Org has no field to require; folder and project have one and don't. **Raw string wins all three. Confirmed live on GKE.**

To use the resource, you must grant the permission that breaks it.

Their docs walk you through the setup. Their security policy pre-declares there's no guardrail. **Every path, line and date below is public.**

Configuration.md · provider-upjet-gcp @ 6378ca5

```
# the documented provider setup
ROLE=roles/editor        # their e.g., L360
gcloud projects add-iam-policy-binding \
  $PROJECT_ID --role=$ROLE   # L434
$ grep -ci "least privilege" → 0
```

One ProviderConfig “default”, shared by every namespace.

what OrganizationIAMMember actually requires

```
OrganizationIAMMember
   → organizations.setIamPolicy
roles/editor                setIamPolicy  ✗
roles/resourcemanager.organizationAdmin  setIamPolicy  ✓
# GCP: setIamPolicy “allows the user to grant all other permissions”
```

Managing org IAM requires org-admin. By design.

Granting the provider org-admin is the intended use, not a misconfiguration.

A central platform team managing org-wide IAM through Crossplane is exactly what `OrganizationIAMMember` is for.

The resource ships with no way to scope who may wield it. Their self-assessment calls that containment a NON-GOAL:

“…does not intend to provide further granular access control… these resources are cluster scoped.”

commit `9baa223f` · Jared Watts · 2024-11-26, written before v2 made managed resources `namespaced by default`. **Premise false. Disclaimer unchanged.**

No reply to me. A new “non-goal” for everyone else.

Reported April 23. Five months later they edited the security policy, not the code. Every date is public git history.

Mar 12 — Reported · Google VRP
#492267271

Mar 26 — Google: Won't Fix
customer-owned infra

Apr 23 — Reported · Crossplane
docs: 2 non-goals

Sep 9 — Non-goal ADDED
docs: 2 → 3

Nov 10 — Public disclosure
this talk

`git show 1ac57cfc · crossplane/crossplane · security/self-assessment.md + SECURITY.md`

```
commit 1ac57cfc   Jared Watts   2026-09-09
  docs(security): add non-goal that Composition authors are trusted

  “We've received numerous security vulnerability reports that are
   actually inline with the project's security design and policy…
   that position wasn't documented anywhere in the project…
   points reporters at the non-goals … before they submit a report.”
```

Google deleted the page. Microsoft added a warning. Crossplane added a non-goal, plus a gate for the next reporter.

Three vendors, same answer: **document a reason it isn't a bug. Ship no fix, assign no CVE.**

THE “CAN'T I JUST…?” SLIDE

Every quick fix misses the same thing.

Each one is real hardening. None of them makes the operator check who it's acting for.

01 STOPGAP

“Can't I just add Kyverno or OPA?”

It sees your YAML and your Kubernetes identity, not your cloud IAM. A per-namespace allowlist helps, but you write and maintain it for every operator.

02 BLAST RADIUS ONLY

“Can't I just scope the operator down?”

A weaker deputy still gets tricked. It just reaches less. And Google's own docs told you to grant Config Connector org owner.

03 WRONG LAYER

“Can't I just use SCPs or Org Policy?”

They cap what identities may do, and the operator is allowed to do this by design. Domain restriction stops a Gmail, not a service account in your own org.

04 AFTER THE FACT

“Can't I just watch the audit logs?”

The cloud log names the operator. The real caller only appears in the Kubernetes audit log, if it's on, and nothing links the two.

05 WRONG IDENTITY

“Isn't this fixed by OIDC?”

OIDC secures how the operator gets credentials, not who it acts for. SPIFFE and Teleport identify the workload, not the caller. My Config Connector demo ran on Workload Identity.

WHAT ACTUALLY CLOSES IT

1. **Verify the caller before acting.**
2. **Authorize the reference, not a raw string.**
3. **Fail closed when you can't verify.**

FOUR CRITICAL BUGS. FOUR SHRUGS.

Every finding confirmed and unpatched. Every vendor edited the docs, not the code.

FINDING	ESCALATES TO	CVSS	THEIR RESPONSE	STATUS
Google Cloud `Config Connector · resourceRef.external`	**namespace user → Org Owner**	9.9	**“Won't fix. Customer-owned infrastructure.”** Rewrote the page after the report.	0 CVE · UNPATCHED
Crossplane / Upbound `provider-gcp · raw orgId · 3 CVEs`	**namespace user → Org IAM**	9.9	**“Seven months of silence.”** Then quietly added a “non-goal.”	0 CVE · UNPATCHED
Microsoft Azure `Azure Service Operator · owner.armId`	**namespace user → Sub Owner**	9.9	**“Does not demonstrate privilege escalation.”** Then shipped a warning 4 months later.	0 CVE · UNPATCHED
AWS `AWS ACK · AssumeRolePolicyDocument`	**namespace user → Account admin**	9.9	**“It's open source. File a GitHub issue.”** Excluded ACK from its disclosure program.	0 CVE · UNPATCHED

+ Same shape confirmed across Config Sync · secrets-store-csi · AlloyDB · Dataproc · Vertex AI (token theft / RBAC bypass · CVSS 7–9.9)

Four findings. Three clouds. Four shrugs. Zero code fixes.

The fix is proven, Google shipped it elsewhere twice. What's missing isn't the patch, it's the will. **And that's on us.**

INDEPENDENT CONFIRMATION · PEER-REVIEWED

I'm not the only one who found this.

A peer-reviewed NDSS 2026 study measured real operators at scale **and gave the class a name.**

2,268

Go operators analyzed

crawled from GitHub · CodeQL v2.17.4

318 14%

potentially vulnerable

7 CVEs

from 8 confirmed reports

vendors include Google & Red Hat

PEER-REVIEWED · NDSS 2026 Feb 23–27, 2026 · San Diego, CA

Breaking the Bulkhead: Demystifying Cross-Namespace Reference Vulnerabilities in Kubernetes Operators

Andong Chen, Zhenyuan Li (Zhejiang University) · Ziyi Guo, Zhaoxuan Jin, Yan Chen (Northwestern University)

The class they named: **Cross-Namespace Reference Vulnerability**

READ THE PAPER

paper · slides · video

Same root cause: the operator never checks the caller. Only the blast radius differs.

NDSS paper	`namespace` →	`another namespace`	→ bypass stays **inside the cluster**
This talk	`namespace` →	`CLOUD ORG IAM`	→ crosses into the **cloud's control plane**

ISN'T THIS THE CUSTOMER'S PROBLEM?

The customer's IAM was correct. The deputy went around it.

The strongest form of “not our problem” is the shared responsibility model. Here's the exact line where it breaks.

> **“Customers always remain responsible for their access policies and data.”**
>
> **Google Cloud, Shared Responsibility Model.** Google's VRP cited exactly this to close the report as “customer-owned infrastructure.”

WHAT THE MODEL ASSUMES

The caller is an identity the customer granted, and can revoke.

Shared responsibility works because your IAM governs who acts.

WHAT ACTUALLY HAPPENS

The call is made by the operator's service account. The namespace user holds zero cloud IAM.

Your policy never saw this caller. The deputy brought its own.

The customer did their half. The user was never granted org admin. The operator acted with its own credentials.

Shared responsibility has two parties: the cloud and the customer. The operator is neither. **Invoking “your responsibility” for it isn't the model. It's a shield.**

THE 10,000-FOOT VIEW

Find it in three questions. Fix it in three moves.

Two people in this room: the ones who **run** operators, and the ones who **write** them. Here's the whole method for both.

IF YOU RUN OPERATORS

FIND IT · the 3-question audit

1. **Inventory the deputies**
   Which operators hold real cloud credentials?
2. **Find the steerable field**
   Can a user name the target? Org, account, scope, ref.
3. **Ask the one question**
   Does it check the caller before acting? If not, it's a deputy.

IF YOU WRITE OPERATORS

FIX IT · three moves

1. **Verify the caller**
   SubjectAccessReview. Borrow the RBAC you already wrote.
2. **Authorize the reference**
   A target you check against the caller, not a raw string.
3. **Fail closed**
   Can't verify? Refuse. Default deny, a gap that can't breach.

Steerable target + powerful key + no caller check = a confused deputy.

WANT THE DEEP DIVE?

The full playbook: Securing Kubernetes Operators

Every command, every Go fix, every copy-paste admission policy. `olearysec.com/playbooks`

1988

38 YEARS LATER · STILL RINGING

2026

ACM SIGOPS · OPERATING SYSTEMS REVIEW

Vol. 22, No. 4 · October 1988 · pp. 36–38

The Confused Deputy

(or why capabilities might have been invented)

NORMAN HARDY

NAMED 1988



YOUR OPERATOR, TODAY

RATE THIS TALK

sfeedback.com/5H5euY

Scan or type the link during Q&A

Your feedback shapes the next one.

Justin O'Leary · olearysec.com · @olearysec · @olearysec@infosec.exchange