From Breach to Fix

The Confused Deputy in Cross-Account IAM

Craig Reeder

Staff Cloud Engineer @ *UTurn Data Solutions*

Who am I?

- Staff Cloud Engineer @ *UTurn Data Solutions*
- Consulted for over a hundred companies, from startups to large enterprises
- Experience across AWS, Automation, DevOps, and Kubernetes
- Hobby magician and mentalist

What is the Confused Deputy Problem?

- Coined by Norm Hardy in 1988
- A **deputy** is a program that acts with authority on behalf of others
- The problem: the deputy is *tricked* into misusing its authority for an attacker
- The deputy isn't malicious, it's **confused** about *who* it's acting for
- The attacker never needs *their own* access, they borrow the deputy's

Common Examples

- **Cross-site request forgery (CSRF)** — the browser is tricked into performing sensitive actions against a web application the user is logged into
- **Server-side request forgery (SSRF)** — a server is tricked into making requests on the attacker's behalf, reaching internal systems it can access but the attacker can't
- **Indirect prompt injection** — an AI agent is tricked by instructions hidden in untrusted content into misusing its tools with the user's authority

A Non-Technical Example

- You hand your keys to a **valet** and get a numbered ticket
- The valet's job: return a car to *whoever presents the matching number*
- The attacker **guesses your number**
- The valet can't tell the difference, so he hands over **your** car
- **The valet is the confused deputy**

The Formula

authority + ambiguity about the requester = vulnerability

A confused deputy needs *both*:

- **Authority** — the deputy holds real privilege (the valet's access to your car, the vendor's admin role, the browser's session cookie)
- **Ambiguity about the requester** — the deputy can't reliably tell *on whose behalf* it's acting, so an attacker's request looks identical to a legitimate one

The SaaS Scenario

SaaS platforms sometimes need to act in your AWS account.

The standard answer: a cross-account **role** they assume.

A Primer on AWS Roles

A **role** isn't a user with a password. It's an identity others *assume*: they temporarily act with the role's permissions instead of their own.

Every role has an **ARN** (Amazon Resource Name), a unique identifier like `arn:aws:iam::123456789012:role/VendorAccess`.

A **cross-account** role lets an identity in *another* AWS account assume it.

Its **trust policy** declares who is allowed to assume it, and AWS enforces that on every request.

How Cross-Account Access Works

1. Vendor gives you **their** AWS account ID
2. You create an IAM role with a **trust policy** naming that account as `Principal`
3. Vendor calls `sts:AssumeRole` on your role → gets credentials

STS issues the credentials; the role's trust policy handles authorization.

The (Vulnerable) Trust Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:root" },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- `111122223333` = the **vendor's** account
- This says: *"this vendor's AWS account is able to assume this role"*
- Looks fine. It's the industry-standard copy-paste.

Why it's Broken

The vendor is multi-tenant. It assumes roles on behalf of *all* its customers.

- At onboarding, each customer tells the vendor which role ARN to assume for them
- Your role ARN is **not a secret**: it's just your (non-secret) AWS account ID plus a standard role name the vendor tells everyone to use
- Nothing ties that ARN to the customer who submitted it

Nothing stops an attacker from onboarding and submitting *your* ARN as their own.

The Attack

1. Attacker signs up for the **same vendor** as you
2. During onboarding, attacker supplies *your* role ARN
3. Vendor (the deputy) calls `sts:AssumeRole` on your role
4. Your trust policy says "vendor account allowed" → **it succeeds**
5. Attacker now drives your environment *through the vendor's UI*

The attacker is now using the deputy to access your environment.

Solutions

The general fix:

Bind the request to a **specific, verifiable intent or principal**, so a request only works for the customer it actually belongs to.

AWS's answer for cross-account roles: the **External ID**.

External ID

A unique identifier the **vendor** hands you during onboarding, that you pin into your trust policy. It operates as a **token of intent**.

The only requirements: it's **unique per customer**, and it's **provided by the vendor**, not the customer.

- The vendor sends it on *every* `sts:AssumeRole` call
- AWS enforces the match — no match, no access
- It ties the assume request to *this specific customer relationship*

The (Fixed) Trust Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:root" },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "customer-a1b2c3d4"
        }
      }
    }
  ]
}
```

Now the vendor account **plus** the matching External ID are both required.

Why This Stops the Attack

- The vendor uses **your** External ID for **your** role
- If an attacker supplies *your* ARN, the vendor sends the **attacker's** External ID
- `customer-a1b2c3d4` ≠ attacker's ID → **AssumeRole is denied**

The deputy can no longer be confused about who it's acting for.

Who Supplies the External ID Matters

The fix only works if the **vendor** supplies the External ID.

If the customer supplies it, an attacker can supply the victim's, and we're back to a confused deputy.

The ARN is easily guessable, so the External ID becomes the *only* thing gating access. It's now doing the job of a secret, but nobody treats it like one.

The Attack, Again

1. Attacker onboards to the **same vendor**
2. Supplies the victim's role ARN *and* the victim's External ID as their own
3. Vendor sends that External ID against the victim's role → **it matches**
4. Access granted

The attacker could claim the victim's value only because customers were allowed to supply it at all.

Building SaaS?

- **As the vendor, generate an External ID**, never let the customer choose
- Make it **unique per customer**
- Send it on every assume-role call automatically
- Bake the External ID into the role policy you give the user at onboarding

Buying SaaS?

If the vendor's role setup has no External ID, or allows you to specify it, the vendor is a risk for you.

Red Teaming?

- Pay attention to which vendors let a user supply their own External ID
- Use read permissions in AWS to pull External IDs for those vendors' roles
- Use that vendor as a confused deputy to escalate privileges in the customer account
- Some companies treat External IDs as if they're secret information, don't chase it unless the vendor is an avenue

Demo

github.com/abyss/aws-confused-deputy-lab

Recap

- **Confused deputy**: a trusted party tricked into misusing its authority
- In AWS it shows up in **cross-account IAM roles** for SaaS
- Trusting only the vendor's account = any attacker who knows your ARN gets in
- **External ID** binds each assume to a specific customer. It must be supplied by the vendor.
- It's an **intent token, not a secret**

Questions?

Craig Reeder, Staff Cloud Engineer

creeder@uturndata.com // me@cwr.io
https://cwr.io

craig-reeder

craigmindreeder

Companion Repository & Slides

github.com/abyss/aws-confused-deputy-lab