FIELD MANUAL · VOLUME I

What the *role* got you.

IDENTITY, DETECTION, AND THE SHAPE OF A BAD THURSDAY

Six lessons pulled out of the novel, with the fiction cut away. The attacks are what actually happens. The detections are what actually fires. If you recognize any of this, good. Go check.

Raajhesh Kannaa · April 2026

The novel takes seventy-two hours. This takes an afternoon. Same material, no plot, no Jess.

I wrote *Assumed Role* because I wanted a story I could hand someone without a security background and have them understand what we do on call. But the readers who actually *do* this kept asking for the parts underneath. This is those parts.

You can read it front to back. You can skip to the one you're arguing about at work. You can copy the detections and ignore the prose. I don't mind. The only rule is that every query in here ran somewhere against real data before I put it on the page, and every fix is one I've either shipped or watched someone ship. If I don't know a clean answer I'll say so.

ACT I

Identity.

Every other cloud problem is downstream of this one. I keep learning it.

01

The credential you didn't revoke.

OFFBOARDING IS A CLOUD PROBLEM NOW

Someone left the company. HR filed the ticket. IT disabled Okta. The laptop came back. Good — that's three out of maybe fifteen things.

What HR doesn't see is the IAM user the engineer created two years ago to wire up a CI tool they were evaluating. Or the access key that ended up in a GitHub Actions secret. Or the role whose trust policy names their personal AWS account so they could test from their side project. None of those get touched by offboarding.

This is where the book starts. It's also where most of the real incidents I've seen start.

WHAT TO ACTUALLY DO

- Maintain a short list of *which identities can survive a person leaving* — roles assumed by services, access keys for break-glass, SSO permission sets. Everything else should break when the human does.
- Run a weekly query over CloudTrail looking for AccessKey usage where the key owner isn't in your active directory. Not because it'll be right every time. Because *having the query* is the point.
- Kill the concept of long-lived access keys for humans. Full stop. Humans assume roles through SSO. If someone needs an access key, they get a STS session credential with a one-hour lifetime. This is a fight worth having once.

DETECTION QUERY (CLOUDTRAIL LAKE)

```
-- Access keys whose owner no longer exists in our IDP roster.
-- The roster is a lookup table you maintain; don't over-engineer it.
SELECT userIdentity.userName, eventName, COUNT(*) AS n
FROM cloudtrail_events
WHERE eventTime > current_date - interval '7' day
  AND userIdentity.type = 'IAMUser'
  AND userIdentity.userName NOT IN (SELECT username FROM active_roster)
GROUP BY 1, 2
ORDER BY n DESC;
```

> **FIELD NOTE**

> *The first time I ran this against a real account, it found three keys belonging to people who'd been gone for 18 months, six months, and a week. The week-old one had made 40,000 API calls the day before. That's the one that keeps me honest.*

REAL INCIDENTS

Uber 2022 — initial access through a contractor's credentials that had been harvested and were still valid. · **Colonial Pipeline 2021** — a legacy VPN account with no MFA whose password had leaked. · **Microsoft 2023 (Storm-0558)** — a signing key that hadn't been rotated and was used to forge tokens for 25+ organizations.

02

Role chaining is a feature. Pretend it isn't.

ASSUMEROLE TRUST POLICIES & THE LONG WALK

You can `sts:AssumeRole` from one role into another, and from that one into another, and so on. AWS calls this a design feature. Attackers call it a tunnel.

The specific problem: if Role A is allowed to assume Role B, and Role B has permissions that Role A doesn't, then compromising Role A gets you Role B's blast radius. In most organizations this chain is three or four hops long because no one mapped it. You wrote a trust policy in 2022 because Jenkins needed it. Jenkins is gone. The trust policy isn't.

WHAT THE POLICY USUALLY LOOKS LIKE WHEN IT'S WRONG

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "AWS": "arn:aws:iam::111122223333:root" },
    "Action": "sts:AssumeRole"
  }]
}
// "root" here means "any identity in account 111122223333 with the right policy."
// Which, if that account is less locked down than this one, is most of them.
```

WHAT IT SHOULD LOOK LIKE

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

WHAT TO ACTUALLY DO

- Render the trust-policy graph. *Literally draw it.* There is no good AWS-native tool; I've used `awspx` and it does the job. You'll be surprised which roles connect to which.
- Put a `Condition` block on every trust policy. Even if it's just an `aws:SourceAccount` check, it's better than nothing.
- Flag cross-account trust to any account you don't govern. Including your own personal account. Especially your own personal account.

> **OPINION**
>
> *The AWS docs for trust policies read like they don't want to scare you. They should scare you a little. A misconfigured trust policy is one of the quietest ways an account gets taken over.*

03

The IAM user who didn't need to exist.

IAM USERS VS. ROLES, AND THE ORG-LEVEL CONTROL THAT ENDS THE ARGUMENT

Every IAM user is a set of keys that lives forever until you delete it. Every role is a set of temporary credentials that expire. This is the entire design of IAM in one sentence. If you are still creating IAM users for services in 2026, you are paying a tax for convenience you don't actually get.

The fix is one SCP at the org level:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "NoNewIAMUsers",
    "Effect": "Deny",
    "Action": ["iam:CreateUser", "iam:CreateAccessKey"],
    "Resource": "*"
  }]
}
```

It breaks some things on the first day. You'll want an exception account for the edge cases (third-party SaaS that can't do OIDC yet). Those exceptions should live in a spreadsheet you review quarterly, not in your production OU.

> **THE SHAPE OF THE ARGUMENT**
>
> *Whoever pushes back will say: "But this vendor needs an access key." Ask whether the vendor supports IAM Roles Anywhere or OIDC federation. Half of them do and nobody noticed. The other half should be asked why in writing, which creates the paper trail you'll need in two years when you're migrating off them.*

ACT II

Detection.

The tools work. The gap is that nobody looks at them until Thursday night.

04

CloudTrail evasion by *not* logging.

STOPLOGGING, STARTLOGGING, AND THE FOUR-MINUTE GAP

CloudTrail has an API called `StopLogging`. If an attacker has the right permission, they can call it, do what they came to do, and call `StartLogging` again. The gap is invisible unless you're looking for it. Most organizations aren't.

Nobody should be calling `StopLogging` in production. It is a high-signal event. Alert on it like you alert on `ConsoleLogin` with MFA=false.

DETECTION QUERY

```
SELECT eventTime, userIdentity.arn, sourceIPAddress
FROM cloudtrail_events
WHERE eventName IN ('StopLogging', 'DeleteTrail', 'UpdateTrail')
  AND eventTime > current_date - interval '30' day
ORDER BY eventTime DESC;
```

THE SUBTLER VERSION

The brute-force attacker calls `StopLogging`. The careful one modifies the trail to exclude a specific event source, waits, then reverts. That shows up as `UpdateTrail`, which is boring-looking until you diff the before and after. Keep the JSON of every `UpdateTrail` invocation and diff it.

> **WHY THIS WORKS FOR ATTACKERS**
>
> *Because most detection systems are downstream of CloudTrail. If the events don't exist, the detections don't fire. It's the cloud version of cutting the phone line before the break-in.*

REAL INCIDENTS

Capital One 2019 — the intrusion used an SSRF to assume a role; the WAF misconfiguration and the lack of alerting on anomalous STS calls is the part that mattered. · **ChaosDB 2021** — not CloudTrail, but the same pattern: logs existed, nobody was watching for the specific shape of the call.

05

S3 replication is an exfil channel.

LEGITIMATE FEATURE, SAME DATA OUT, NO EC2 REQUIRED

S3 replication is the thing you turn on to copy a bucket's contents to another bucket, maybe in another region, for DR. An attacker with `s3:PutBucketReplication` on a sensitive bucket can point the replication at a bucket they own in an account they own. AWS will dutifully copy the data for them. Every object. Forever. Until someone notices the replication rule.

The event in CloudTrail is `PutBucketReplication`. It is almost never legitimate in a production account. Treat it as a fire drill.

PREVENTION (SCP)

```
{
  "Sid": "OnlyReplicateToOurAccounts",
  "Effect": "Deny",
  "Action": "s3:PutBucketReplication",
  "Resource": "*",
  "Condition": {
    "StringNotEqualsIfExists": {
      "aws:ResourceAccount": ["<list of our accounts>"]
    }
  }
}
```

DETECTION (RUNS EVERY 15 MIN)

```
SELECT eventTime, userIdentity.arn, requestParameters
FROM cloudtrail_events
WHERE eventName = 'PutBucketReplication'
  AND eventTime > current_timestamp - interval '15' minute;
```

> **RARER THAN PEOPLE THINK**
>
> *In four years of looking I have seen exactly one legitimate `PutBucketReplication` outside of Terraform. The cost of alerting on every one is low. The cost of missing one is catastrophic.*

06

Write the detection *before* you need it.

DETECTION-AS-CODE, AND THE 3 AM TEST

The detections in this manual are cheap. Writing them after the incident is not. The single largest multiplier on incident response is whether the query already exists. If it exists, you paste and run. If it doesn't, you write it while your adrenaline is up and it's slightly wrong.

Three rules.

— **Every detection is a file in a repo.** Not a saved search in a vendor UI. A file. You can diff it, review it, roll it back.

— **Every detection has a test.** A synthetic event, a known payload, something that makes the query fire. If it doesn't fire on the test, it won't fire in the incident.

— **The 3 AM test.** Imagine reading your own detection at 3 AM on a phone. If it doesn't tell you what to do next in the first two sentences, rewrite it.

> **WHERE THIS CAME FROM**
>
> *Jess's notebook in the novel is real. The detections she copy-pastes from her own past incidents are the pattern I wish I'd started with. It took me too long to realize that the best thing I could do between incidents was make the next incident cheaper.*

TOOLS WORTH THE TIME

— Panther or StreamAlert if you want detections-as-code as a first-class practice.

— varc when you need a volatile-artifact snapshot from a running EC2 instance during IR.

— Your own CloudTrail Lake queries, versioned in a repo, with a GitHub Action that runs them against a test dataset on every PR.

END OF VOLUME I

Six lessons is not a full program. It's the floor. If your identity hygiene, detection coverage, and response muscle are better than what's in here, good — the next volume covers supply chain, which is a different problem. If they aren't, start with the week-old key in Lesson 1 and work down.

The long-form is in *Assumed Role*, the novel this manual comes out of. Jess's seventy-two hours are mine, and they're someone else's you know.

Set in IBM Plex Mono and Source Serif Pro. Written on paper first, typed second. Print-ready; Cmd-P strips the screen aesthetic and gives you ink on white.

Raajhesh Kannaa · Toronto · defensive.works
Cloud Security Cinematic Universe · Book 01 · Field Manual I · April 2026

— END OF FIELD MANUAL I —