Vantalect

AWS SECURITY, COMPLIANCE & GOVERNANCE · SAMPLE DELIVERABLE

Audit Report Appendix: Remediation Code

Week 2 of 6: Verify and plan (appendix) · prepared for StrataForge Robotics, Inc.

Prepared by Charles Sieg · charles.sieg@vantalect.com · vantalect.com

July 24, 2026 · AWS Security, Compliance & Governance Audit: The Hardening (six weeks)

SAMPLE DELIVERABLE

This is a sample deliverable. StrataForge Robotics, Inc. is a fictional company: every account, resource, finding, figure, and person in this report is simulated to show the form and depth of a Vantalect engagement. AWS prices are list prices from the AWS Price List API on October 13, 2026.

Contents

How to use this appendix 4

Critical findings 5

SF-SEC-001: A 1,127-day-old IAM user key can sign and publish firmware to every robot 5

SF-SEC-002: One IoT policy lets any of 61,400 robots' certificates publish to any robot's command topic 11

SF-SEC-003: 4 GitHub Actions deploy roles can be assumed from any GitHub repository 16

SF-SEC-004: Jupyter is open to the internet on 51 research and sandbox instances, some with no token 20

SF-SEC-005: 5 EBS snapshots in kestrel-prod are public, and snapshot block public access is off in all 816 account-Regions 22

SF-SEC-006: 4 Kestrel dataset buckets are public: 468.8 million objects of labeled images from customer sites 25

SF-SEC-007: A Kestrel MySQL snapshot with annotation records is restorable by any AWS account 28

SF-SEC-008: Kestrel's Jenkins controller is open to the internet on 8080 and 22 30

SF-SEC-009: RDP is open to the internet on 32 PLM CAD workstations that hold product designs 33

SF-SEC-010: Root users of 3 Kestrel accounts have access keys and no MFA 35

High findings 38

SF-SEC-011: docs-legacy.strataforge.example points at a deleted S3 website bucket that anyone can claim 38

SF-SEC-012: 1,009 active access keys are older than 90 days; 400 are over three years old 40

SF-SEC-013: 108 IAM users sign in to the console without MFA, 2 of them in the management account 44

SF-SEC-014: 1,631 instances still allow IMDSv1 46

SF-SEC-015: 144 Lambda functions have secret-named environment variables 49

SF-SEC-016: All 16 EKS cluster API endpoints accept connections from 0.0.0.0/0 51

SF-SEC-017: 14 customer managed policies grant Action * on Resource * to 17 workload roles and 46 IAM users 53

SF-SEC-018: 12 privilege-escalation paths lead to administrator in 11 accounts 58

SF-SEC-019: Kestrel's former management account still administers four Kestrel member accounts 65

SF-SEC-020: A former contractor's AWS account holds AdministratorAccess in mfg-mes-prod with no ExternalId 68

SF-SEC-021: The staging OTA test key lets the production pipeline sign, and pilot robots still trust it . 71
SF-SEC-022: AWS Config records nothing in 570 account-Regions 76
SF-SEC-023: GuardDuty is off in 558 account-Regions, including every Kestrel account; most protection plans are off 78
SF-SEC-024: S3 Block Public Access is off at the account level in 31 accounts and on 47 buckets 83
SF-SEC-025: The 2019 organization trail can be stopped from the management account, and its logs can be deleted 86
SF-SEC-026: Service control policies stop nothing: only FullAWSAccess and DenyLeaveOrganization are attached 94
SF-SEC-027: SSH bastions accept connections from the whole internet 100
SF-SEC-028: SSH is open to the internet on 76 development instances 101
SF-SEC-029: 4 more member-account root users have no MFA; 40 use virtual MFA 101
SF-SEC-030: 23 robot certificates are shared by 1,840 robots at 3 customer sites 102

Medium findings 107

SF-SEC-031: 1,247 Lambda functions run on deprecated runtimes 107
SF-SEC-032: EBS encryption by default is off in 762 account-Regions; 1,245 volumes are unencrypted 109
SF-SEC-033: 556 running instances are not managed by Systems Manager; 2,380 run AMIs over a year old 114
SF-SEC-034: 39 internet-facing ALBs have no AWS WAF; 84 ALBs do not redirect HTTP to HTTPS.... 116
SF-SEC-035: 22 database instances store data unencrypted 119
SF-SEC-036: 9 ElastiCache replication groups lack encryption at rest or in transit 121
SF-SEC-037: 5 EFS file systems are not encrypted at rest 124
SF-SEC-038: Security Hub CSPM runs in only 3 Regions, with FSBP only and no CIS standard 126
SF-SEC-039: 7 accounts are outside Control Tower, and the landing zone has drifted 130
SF-SEC-040: Break-glass access is not alarmed and was last tested in April 2025 133
SF-SEC-041: SSH and RDP are open to the internet on 27 sandbox instances 136
SF-SEC-042: Amazon Inspector is off almost everywhere; 132 ECR repositories have no image scanning 137
SF-SEC-043: 750 default VPCs, most in Regions nobody uses, have open default security groups and network ACLs 142
SF-SEC-044: IAM Access Analyzer covers only us-east-1; 768 account-Regions have no external access analyzer 145
SF-SEC-045: 31 databases keep one day of backups; 112 resilience checks fail 147

Low findings 150

SF-SEC-046: 130 customer managed policies use service-wide wildcards such as s3:* 150
SF-SEC-047: 2,850 of 14,464 resources have no Team tag, so no owner can be found for them 151
SF-SEC-048: 1,922 workload roles have not been used in 90 days or more 153
SF-SEC-049: MFA delete is off on every bucket 157
SF-SEC-050: Account baseline gaps: no security contact in 42 accounts, no support role in 20, CloudShell full access on 195 identities 159
SF-SEC-051: Server access logging is off on 233 buckets 162
SF-SEC-052: 220 buckets accept requests without TLS 165
SF-SEC-053: 78 customer managed keys do not rotate 167
SF-SEC-054: EKS control-plane logging is off on 15 of 16 clusters, and every cluster still uses the aws-auth ConfigMap 170

How to use this appendix

This appendix is the companion to the Audit Report (deliverables 2 to 6 of the Audit). The audit report says what is wrong, how it was rated, and what to do about it, with the steps and the key code for every Critical and High finding. This document holds the rest: for each of the 54 findings, under the same finding ID, the evidence, how exploitable it is and how far it reaches, the controls and SOC 2 criteria it touches, the full remediation with **every** code block, how to verify it, and **how to roll it back**.

- **Find a finding by its ID.** Findings are in the register's order within each severity: 10 Critical, 20 High, 15 Medium, and 9 Low. Several findings share one remediation (the same change in different accounts); the later ones point to the first.
- **Where the code goes.** Every change is made in code, in the repository the finding names: `platform-landing-zone` (CloudFormation StackSets, service control policies, and account baselines for every account), `fleet-infra`, `crucible-infra`, or `enterprise-infra` (AWS CDK in TypeScript for those groups of accounts). Where a resource is not in code yet, the steps are a runbook of AWS CLI commands. Nothing is changed by hand in the console.
- **Placeholders.** Values in angle brackets, such as `<account-id>`, are the only things you fill in. Everything else is complete and meant to run or merge as written.
- **Effort.** The hours on each finding are the engineering time a team would spend doing the change by hand, from first commit to a verified close, including review and the soak in non-production. In the Hardening (weeks 3 to 5), Vantalect writes the pull requests with agents, under review, inside its 110 hours; the owners review, merge, and roll out. The hours size the work. They are not the time StrataForge's engineers must find.

- **Rolling back.** Every change goes in as a pull request, so a rollback starts as a revert of the commit and a run of your pipeline. The rollback under each remediation covers what a revert does not: the order to undo things in, the command for a change made outside code, and what cannot be undone by design, such as a deleted credential, a revoked certificate, or a compliance-mode retention lock. For a change that opens a security hole when undone (public data, an open port, a root session), the rollback says so and gives the safer way to unblock the team.

Critical findings

10 Critical findings, 137 hours between them.

SF-SEC-001: A 1,127-day-old IAM user key can sign and publish firmware to every robot

Critical. Benchmark rating Medium (Security Hub CSPM IAM.3 on the key; raised for blast radius and safety). How it was rated: benchmark Medium (2) +0 exploitability (moderate) +2 blast radius (fleet) = 4

Where: `fleet-ota-prod` (…5117) in us-east-1, 1 IAM user. **Controls:** `IAM.2` (IAM users should not have IAM policies attached); `IAM.3` (IAM users' access keys should be rotated every 90 days or less); also Vantalect OTA chain-of-trust review. **SOC 2:** CC6.1, CC6.3, CC8.1. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 24 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- Credential report: ota-signing-ci, access key created 06/14/2023 (1,127 days), last used the day before the scan (kms:Sign, us-east-1).
- Attached policy allows kms:Sign on alias/ota-firmware-signing-prod (ECC_NIST_P384) and s3:PutObject on sf-ota-ota-images-5117-us-east-1-000/releases/*.
- CloudTrail (90 days): Sign calls from GitHub-hosted runners and from two engineer laptops in June 2026.
- Robots verify the signature only; nothing checks who produced the build or whether a second person approved it.

Examples: `arn:aws:iam::…5117:user/ota-signing-ci`, `alias/ota-firmware-signing-prod`, `sf-ota-ota-images-5117-us-east-1-000`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). One leaked key is enough: no MFA, source-IP, or VPC-endpoint condition, and the key has been copied to two laptops. **Blast radius.** Robots in the field at customer sites. A signed image in the distribution bucket reaches all 212,000 connected robots. This finding has a safety impact on robots in the field, which is why it is Critical whatever the benchmark says.

How we verified it. Walked the release pipeline with Sam Okafor on 07/21/2026; the key is the only production signer.

Remediation: Replace the OTA signing IAM user with two GitHub OIDC roles and a locked key policy. An access key more than three years old for `ota-signing-ci` can both sign firmware and publish it, so one stolen key reaches every robot in the fleet. Split signing and publishing into two short-lived roles behind a two-person approval, and let only the signer role use the key.

Steps

1. Create the GitHub environment `production` on `strataforge/ota-release` with two required reviewers, `prevent_self_review`, and a deployment branch policy that allows only `main`. GitHub needs one approval, and self-review prevention makes the approver a second person, which is the two-person rule.
2. Deploy the CDK stack that creates `ota-release-signer` (kms:Sign only, ECDSA_SHA_384) and `ota-release-publisher` (s3:PutObject on `releases/*` only), both trusting only `repo:strataforge/ota-release:environment:production`.
3. Change the release workflow to assume the signer role for the signing job and the publisher role for the upload job, with `id-token: write` and no stored AWS secrets.
4. Look up the key ID with `describe-key` (`put-key-policy` takes a key ID or key ARN, never an alias) and save the current key policy to the change record. Then apply the new key policy so kms:Sign is denied to everyone except the signer role.
5. Create the EventBridge rule in the key's Region with state `ENABLED_WITH_ALL_CLOUDTRAIL_MANAGEMENT_EVENTS`, because Sign is a read-only management event and an ordinary `ENABLED` rule never sees it, and add the target that pages the security team. Confirm one full release signs and publishes.
6. Deactivate the user's access key, wait one release cycle, then delete the key and the user.

Signer and publisher roles trusted only by the production environment (AWS CDK, TypeScript)

```typescript
import { Duration, Stack, StackProps, aws_iam as iam } from 'aws-cdk-lib';
import { Construct } from 'constructs';

const PROVIDER = 'arn:aws:iam::<fleet-ota-prod-account-id>:oidc-provider/
token.actions.githubusercontent.com';

export class OtaReleaseRolesStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const trust = new iam.WebIdentityPrincipal(PROVIDER, {
      StringEquals: {
        'token.actions.githubusercontent.com:aud': 'sts.amazonaws.com',
        'token.actions.githubusercontent.com:sub': 'repo:strataforge/ota-
release:environment:production',
      },
    });
    const signer = new iam.Role(this, 'Signer', { roleName: 'ota-release-signer', assumedBy:
trust, maxSessionDuration: Duration.hours(1) });
    signer.addToPolicy(new iam.PolicyStatement({
      actions: ['kms:Sign'],
      resources: ['arn:aws:kms:us-east-1:<fleet-ota-prod-account-id>:key/<ota-firmware-signing-
key-id>'],
      conditions: { StringEquals: { 'kms:SigningAlgorithm': 'ECDSA_SHA_384' } },
    }));
    const publisher = new iam.Role(this, 'Publisher', { roleName: 'ota-release-publisher',
assumedBy: trust, maxSessionDuration: Duration.hours(1) });
    publisher.addToPolicy(new iam.PolicyStatement({
      actions: ['s3:PutObject'],
      resources: ['arn:aws:s3:::<ota-distribution-bucket>/releases/*'],
    }));
  }
}
```

Key policy for alias/ota-firmware-signing-prod: only the signer role may call Sign (KMS key policy)

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnableIamAdministration",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<fleet-ota-prod-account-id>:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "SignerRoleMaySign",
      "Effect": "Allow",
      "Principal": {
```

```
        "AWS": "arn:aws:iam::<fleet-ota-prod-account-id>:role/ota-release-signer"
      },
      "Action": "kms:Sign",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:SigningAlgorithm": "ECDSA_SHA_384"
        }
      }
    },
    {
      "Sid": "SignerRoleMayDescribeTheKey",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<fleet-ota-prod-account-id>:role/ota-release-signer"
      },
      "Action": [
        "kms:DescribeKey",
        "kms:GetPublicKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "NobodyElseMaySign",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "kms:Sign",
      "Resource": "*",
      "Condition": {
        "ArnNotEquals": {
          "aws:PrincipalArn": "arn:aws:iam::<fleet-ota-prod-account-id>:role/ota-release-
signer"
        }
      }
    }
  ]
}
```

ota-unexpected-sign-pattern.json: page on any Sign call that did not come from the signer role (EventBridge event pattern)

```JSON
{
  "source": [
    "aws.kms"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "kms.amazonaws.com"
    ],
    "eventName": [
      "Sign"
    ],
    "userIdentity": {
      "arn": [
        {
          "anything-but": {
            "prefix": "arn:aws:sts::<fleet-ota-prod-account-id>:assumed-role/ota-release-
signer/"
          }
        }
      ]
    }
  }
}
```

Require two reviewers, save and apply the key policy, add the Sign alarm, then retire the old key (AWS CLI)

```bash
set -euo pipefail
gh api --method PUT repos/strataforge/ota-release/environments/production --input - <<'JSON'
{"reviewers": [{"type": "User", "id": <release-manager-github-user-id>}, {"type": "User", "id":
<security-reviewer-github-user-id>}], "prevent_self_review": true,
 "deployment_branch_policy": {"protected_branches": false, "custom_branch_policies": true}}
JSON
gh api --method POST repos/strataforge/ota-release/environments/production/deployment-branch-
policies -f name=main -f type=branch

KEY_ID=$(aws kms describe-key --profile <fleet-ota-prod-profile> --region us-east-1 --key-id
alias/ota-firmware-signing-prod \
  --query KeyMetadata.KeyId --output text)
# Save the current policy first; the rollback restores exactly this file.
aws kms get-key-policy --profile <fleet-ota-prod-profile> --region us-east-1 --key-id "$KEY_ID"
--policy-name default \
  --query Policy --output text > ota-key-policy.before.json
test -s ota-key-policy.before.json
aws kms put-key-policy --profile <fleet-ota-prod-profile> --region us-east-1 \
  --key-id "$KEY_ID" --policy-name default --policy file://ota-key-policy.json

# Sign is a read-only management event: only this rule state delivers it. The SNS topic must
allow events.amazonaws.com to publish.
aws events put-rule --profile <fleet-ota-prod-profile> --region us-east-1 --name sf-ota-
unexpected-sign \
  --state ENABLED_WITH_ALL_CLOUDTRAIL_MANAGEMENT_EVENTS --event-pattern file://ota-unexpected-
sign-pattern.json
aws events put-targets --profile <fleet-ota-prod-profile> --region us-east-1 --rule sf-ota-
unexpected-sign \
  --targets Id=page-security,Arn=<security-alarm-topic-arn>

aws iam update-access-key --profile <fleet-ota-prod-profile> --user-name ota-signing-ci \
  --access-key-id <ota-signing-ci-access-key-id> --status Inactive
# After one clean release cycle. delete-user fails with DeleteConflict while the user still has
policies or groups: save and remove those first.
aws iam delete-access-key --profile <fleet-ota-prod-profile> --user-name ota-signing-ci --
access-key-id <ota-signing-ci-access-key-id>
aws iam delete-user --profile <fleet-ota-prod-profile> --user-name ota-signing-ci
```

How it is verified

- `aws iam list-access-keys --user-name ota-signing-ci` shows the key Inactive in fleet-ota-prod; after the next clean release the key and the user are deleted (the last step), and `aws iam get-user` then returns NoSuchEntity.
- A release run assumes both roles with no stored AWS secret, and CloudTrail shows every Sign call under `assumed-role/ota-release-signer`.
- A test kms:Sign call from any other role, an administrator included, is denied and raises the EventBridge page (check the rule's `State` is `ENABLED_WITH_ALL_CLOUDTRAIL_MANAGEMENT_EVENTS`).

How to roll it back. Before step 4, the current key policy is saved as `ota-key-policy.before.json`. To roll back, restore that policy first, before anything else, with `aws kms put-key-policy --key-id <key-id> --policy-name default --policy file://ota-key-policy.before.json` (use the key ID from `describe-key`, never the alias). Reverting the CDK stack first would leave a key policy that denies Sign to everyone, because the signer role it names would be gone. Then revert the CDK stack and the release workflow change in `fleet-infra`, and reactivate the old access key with `aws iam update-access-key --user-name ota-signing-ci --access-key-id <key-id> --status Active`, which works only until the key is deleted in step 6. Keep the key deactivated, not deleted, for one full release cycle so this path stays open. Once the key is deleted it cannot come back; a replacement key has to be created by the PlatformIdentityAdmin role, because `sf-protect-account-defaults` denies iam:CreateAccessKey to everyone else, so scope it to the signer actions only and treat the rollback as a new incident. Remove the `sf-ota-unexpected-sign` rule last, with `aws events remove-targets` and `aws events delete-rule`. Verify by running one release and checking CloudTrail for the Sign call and the principal that made it.

SF-SEC-002: One IoT policy lets any of 61,400 robots' certificates publish to any robot's command topic

Critical. Benchmark rating Critical (AWS IoT Device Defender IOT_POLICY_OVERLY_PERMISSIVE_CHECK). How it was rated: benchmark Critical (4) -1 exploitability (limited) +2 blast radius (fleet) = 5; rule 2: Critical needs a remote path and a production-or-wider blast radius; capped at High; rule 4: safety impact on robots in the field; raised to Critical

Where: `fleet-ingest-prod` (…5012) in 4 Regions, 61,400 robot certificates. **Controls:** No Security Hub CSPM control covers this. Found by: IOT_POLICY_OVERLY_PERMISSIVE_CHECK. **SOC 2:** CC6.1, CC6.6. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 40 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- Policy sf-robot-legacy-v1 allows iot:* on * and is attached to 61,400 certificates.
- Device Defender's daily scheduled audit (us-east-1 only) has reported the check as non-compliant since 03/2024.
- No scheduled audit runs in eu-central-1, ap-northeast-1, or ap-southeast-1.

Examples: `sf-robot-legacy-v1`, `thing group: forgearm-gen2`, `thing group: stratarunner-gen1`.

Exploitability. Needs a foothold inside, an insider, or physical access. Needs one certificate and key, which gen-2 ForgeArm units keep on the filesystem rather than in a secure element. **Blast radius.** Robots in the field at customer sites. Command topics sf/v1/<thing>/cmd reach motion planning on every robot on the policy. This finding has a safety impact on robots in the field, which is why it is Critical whatever the benchmark says.

How we verified it. Reproduced with the fleet team on a bench robot on 07/22/2026: its certificate published to another robot's command topic.

Remediation: Replace `sf-robot-legacy-v1` with a thing-scoped IoT policy and roll it out by thing group. The IoT policy `sf-robot-legacy-v1` grants `iot:*` on `*` to every robot certificate, so one stolen certificate can read, publish to, or control the whole fleet (Device Defender IOT_POLICY_OVERLY_PERMISSIVE_CHECK). `sf-robot-scoped-v2` limits each robot to its own client ID, topics, shadow, and jobs, and `sf-robot-legacy-v1-restricted` does the same on the old topic tree for robots whose firmware has not moved yet.

Steps

1. Create `sf-robot-scoped-v2` and `sf-robot-legacy-v1-restricted` in each of the four IoT Regions: us-east-1, eu-central-1, ap-northeast-1, and ap-southeast-1.
2. The `${iot:Connection.Thing.ThingName}` variable resolves only when a robot connects with its thing name as the client ID, so check the connect logs of a bench robot and of each wave first. Put the robots that pass in `wave-1-things.txt` (one thing name per line), and leave out any that connect with another client ID until their firmware is fixed.
3. Create the wave's thing group, add every robot in `wave-1-things.txt` to it, and attach v2 to the group; IoT policies add up, so v2 changes nothing until v1 is detached.
4. Detach `sf-robot-legacy-v1` from the wave's certificates, skipping any certificate that more than one robot uses (SF-SEC-030 retires those), and watch connection failures for 24 hours before the next wave.
5. Robots whose firmware still publishes under `sf/v1/<thing>/` go into the thing group `sf-legacy-firmware` under `sf-robot-legacy-v1-restricted`: each robot keeps its own v1 topics, its shadow, and its jobs (the jobs topics deliver the firmware fix), every cloud command topic is closed to a robot certificate (these robots take commands only from the site's local controller), and no robot can reach another robot's topics. Record the residual risk as a signed acceptance with an end date.
6. Enable Device Defender audit in all four Regions on a daily schedule, so a permissive policy is found again if it returns.

sf-robot-scoped-v2 (AWS IoT policy)

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:*:<fleet-ingest-prod-account-id>:client/$
{iot:Connection.Thing.ThingName}"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topic/sf/v2/$
{iot:Connection.Thing.ThingName}/*",
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topic/$aws/things/$
{iot:Connection.Thing.ThingName}/shadow/*",
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topic/$aws/things/$
{iot:Connection.Thing.ThingName}/jobs/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topicfilter/sf/v2/$
{iot:Connection.Thing.ThingName}/*",
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topicfilter/$aws/things/$
{iot:Connection.Thing.ThingName}/shadow/*",
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topicfilter/$aws/things/$
{iot:Connection.Thing.ThingName}/jobs/*"
      ]
    }
  ]
}
```

sf-robot-legacy-v1-restricted: robots on old firmware, until it is updated (AWS IoT policy)

```
JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:*:<fleet-ingest-prod-account-id>:client/$
{iot:Connection.Thing.ThingName}"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
```

```
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topic/sf/v1/$
{iot:Connection.Thing.ThingName}/*",
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topic/$aws/things/$
{iot:Connection.Thing.ThingName}/shadow/*",
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topic/$aws/things/$
{iot:Connection.Thing.ThingName}/jobs/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topicfilter/sf/v1/$
{iot:Connection.Thing.ThingName}/*",
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topicfilter/$aws/things/$
{iot:Connection.Thing.ThingName}/shadow/*",
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topicfilter/$aws/things/$
{iot:Connection.Thing.ThingName}/jobs/*"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topic/sf/v1/*/cmd"
    },
    {
      "Effect": "Deny",
      "Action": "iot:Subscribe",
      "Resource": "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topicfilter/sf/v1/*/cmd"
    }
  ]
}
```

Create the policies in all four Regions and roll out one wave (AWS CLI)

```bash
set -euo pipefail
for region in us-east-1 eu-central-1 ap-northeast-1 ap-southeast-1; do
  aws iot create-policy --region "$region" --policy-name sf-robot-scoped-v2 --policy-document
file://sf-robot-scoped-v2.json
  aws iot create-policy --region "$region" --policy-name sf-robot-legacy-v1-restricted \
    --policy-document file://sf-robot-legacy-v1-restricted.json
done
REGION=us-east-1   # repeat from here for each Region, with that Region's wave file
aws iot create-thing-group --region "$REGION" --thing-group-name sf-rollout-wave-1
while read -r thing; do
  aws iot add-thing-to-thing-group --region "$REGION" --thing-group-name sf-rollout-wave-1 --
thing-name "$thing"
done < wave-1-things.txt
```

```
GROUP_ARN=$(aws iot describe-thing-group --region "$REGION" --thing-group-name sf-rollout-
wave-1 --query thingGroupArn --output text)
aws iot attach-policy --region "$REGION" --policy-name sf-robot-scoped-v2 --target "$GROUP_ARN"
aws iot list-attached-policies --region "$REGION" --target "$GROUP_ARN" --query
'policies[].policyName' --output text | grep -qw sf-robot-scoped-v2

# cutover GROUP POLICY: detach sf-robot-legacy-v1 from the certificate of every robot in GROUP,
# but only after that group carries POLICY, and never from a certificate shared by several
robots.
cutover() {
  local group_arn
  group_arn=$(aws iot describe-thing-group --region "$REGION" --thing-group-name "$1" --query
thingGroupArn --output text)
  aws iot list-attached-policies --region "$REGION" --target "$group_arn" --query
'policies[].policyName' --output text | grep -qw "$2"
  for thing in $(aws iot list-things-in-thing-group --region "$REGION" --thing-group-name "$1"
--query 'things[]' --output text); do
    for cert in $(aws iot list-thing-principals --region "$REGION" --thing-name "$thing" --
query 'principals[]' --output text); do
      users=$(aws iot list-principal-things --region "$REGION" --principal "$cert" --query
'things | length(@)')
      if [ "$users" != "1" ]; then echo "skip $cert: used by $users robots"; continue; fi
      aws iot detach-policy --region "$REGION" --policy-name sf-robot-legacy-v1 --target
"$cert"
      echo "$cert" >> "detached-certificates-$REGION-$1.txt"   # the rollback re-attaches v1 to
exactly these
    done
  done
}
# After a clean 24 hours:
cutover sf-rollout-wave-1 sf-robot-scoped-v2

# Robots that cannot move yet: their own group, the restricted policy, then the same cutover.
aws iot create-thing-group --region "$REGION" --thing-group-name sf-legacy-firmware
while read -r thing; do
  aws iot add-thing-to-thing-group --region "$REGION" --thing-group-name sf-legacy-firmware --
thing-name "$thing"
done < legacy-firmware-things.txt
LEGACY_ARN=$(aws iot describe-thing-group --region "$REGION" --thing-group-name sf-legacy-
firmware --query thingGroupArn --output text)
aws iot attach-policy --region "$REGION" --policy-name sf-robot-legacy-v1-restricted --target
"$LEGACY_ARN"
cutover sf-legacy-firmware sf-robot-legacy-v1-restricted
aws iot list-targets-for-policy --region "$REGION" --policy-name sf-robot-legacy-v1 --query
'targets | length(@)'
```

Turn on Device Defender audit checks in all four Regions (AWS CLI)

```bash
CHECKS="IOT_POLICY_OVERLY_PERMISSIVE_CHECK DEVICE_CERTIFICATE_SHARED_CHECK
CONFLICTING_CLIENT_IDS_CHECK IOT_POLICY_POTENTIAL_MISCONFIGURATION_CHECK"
for region in us-east-1 eu-central-1 ap-northeast-1 ap-southeast-1; do
  aws iot update-account-audit-configuration --region "$region" \
    --role-arn arn:aws:iam::<fleet-ingest-prod-account-id>:role/DeviceDefenderAuditRole \
    --audit-check-configurations '{"IOT_POLICY_OVERLY_PERMISSIVE_CHECK": {"enabled": true},
"DEVICE_CERTIFICATE_SHARED_CHECK": {"enabled": true}, "CONFLICTING_CLIENT_IDS_CHECK":
{"enabled": true}, "IOT_POLICY_POTENTIAL_MISCONFIGURATION_CHECK": {"enabled": true}}'
  aws iot create-scheduled-audit --region "$region" --scheduled-audit-name sf-daily-iot-audit
--frequency DAILY --target-check-names $CHECKS
done
```

How it is verified

- `aws iot list-targets-for-policy --policy-name sf-robot-legacy-v1` returns only the certificates the cutover skipped as shared in each of the four Regions (SF-SEC-030 retires those), and `sf-robot-legacy-v1-restricted` is attached only to the `sf-legacy-firmware` thing group.
- A Device Defender audit shows no IOT_POLICY_OVERLY_PERMISSIVE_CHECK finding.
- A test certificate attached to `robot-a` cannot connect as client ID `robot-b` or publish to `sf/v2/robot-b/telemetry`.
- A certificate in `sf-legacy-firmware` cannot publish to or subscribe to any `sf/v1/<thing>/cmd` topic or reach another robot's topics, and its robot still receives its own jobs.

How to roll it back. Re-attach `sf-robot-legacy-v1` to the certificates the cutover detached, which are listed in `detached-certificates-<region>-<group>.txt`, with `aws iot attach-policy --target <certificate-arn>`; that restores the old access within seconds. Detach `sf-robot-scoped-v2` from the wave's thing group only if it blocks connections. Roll back one wave at a time and keep the Device Defender audit on. The `sf-legacy-firmware` group keeps `sf-robot-legacy-v1-restricted`; never put the unrestricted policy back on it. Verify with the wave's connection count and a Device Defender run.

SF-SEC-003: 4 GitHub Actions deploy roles can be assumed from any GitHub repository

Critical. Benchmark rating High (Vantalect baseline; IAM Access Analyzer reports the federated principal as external access). How it was rated: benchmark High (3) +1 exploitability (trivial) +1 blast radius (organization) = 5

Where: `sf-cicd` (…1356), `fleet-api-prod` (…5044), `customer-portal-prod` (…5215), `fleet-staging` (…6014) in us-east-1, 4 IAM roles. **Controls:** No Security Hub CSPM control covers this. Found by: IAM Access Analyzer external access. **SOC 2:** CC6.1, CC6.6, CC8.1. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `fleet-infra`. **Effort:** 6 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- Trust policies condition only on token.actions.githubusercontent.com:aud = sts.amazonaws.com.
- Created in 2022 before StrataForge's CDK construct for GitHub roles existed; 33 later roles pin sub correctly.

Examples: `gha-deploy-fleet-api (fleet-api-prod)`, `gha-deploy-portal (customer-portal-prod)`, `gha-deploy-staging (fleet-staging)`, `gha-artifact-publisher (sf-cicd)`.

Exploitability. Remote and unauthenticated, with a known technique. Any GitHub user can mint a token with audience sts.amazonaws.com from their own workflow. **Blast radius.** Every account in the organization. gha-artifact-publisher pushes the images every workload account deploys.

How we verified it. Assumed gha-deploy-staging from a Vantalect-owned test repository on 07/21/2026 with written approval.

Remediation: Pin every GitHub Actions deploy role to one repository and environment. Four deploy roles trust `token.actions.githubusercontent.com` with only an `aud` condition, so a workflow in any GitHub repository can assume them. Add a `sub` condition that names the repository and environment.

Steps

1. List the repository and environment each role really serves, from 90 days of AssumeRoleWithWebIdentity events.
2. Write one trust policy per role with `aud` equal to `sts.amazonaws.com` and `sub` pinned with StringEquals (no wildcards).
3. Protect each GitHub environment (required reviewers for production) so the `environment` claim means something.
4. The four roles already exist, and CloudFormation cannot create a role whose name is taken, so first bring them under the stack with `cdk import` (the stack sets `RemovalPolicy.RETAIN` on each role, which an import requires); then `cdk deploy` writes the pinned trust policies. Save each role's current trust policy (`aws iam get-role --query Role.AssumeRolePolicyDocument`) before the import. Use the CLI command only for an emergency change, and reconcile the stack afterward.

5. Run each deploy workflow once and confirm it assumes the role; run a workflow from a scratch repository and confirm AccessDenied.

Trust policy for gha-deploy-fleet-api (IAM trust policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<fleet-api-prod-account-id>:oidc-provider/
token.actions.githubusercontent.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
          "token.actions.githubusercontent.com:sub": "repo:strataforge/fleet-
api:environment:production"
        }
      }
    }
  ]
}
```

One loop for all four roles, each pinned to its repository and environment (AWS CDK, TypeScript)

```TYPESCRIPT
import { Duration, RemovalPolicy, Stack, StackProps, aws_iam as iam } from 'aws-cdk-lib';
import { Construct } from 'constructs';

const PROVIDER = 'arn:aws:iam::<fleet-api-prod-account-id>:oidc-provider/
token.actions.githubusercontent.com';
const PINS: Record<string, string[]> = {
  'gha-deploy-fleet-api': ['repo:strataforge/fleet-api:environment:production'],
  'gha-deploy-portal': ['repo:strataforge/customer-portal:environment:production'],
  'gha-deploy-staging': ['repo:strataforge/fleet-api:environment:staging', 'repo:strataforge/
customer-portal:environment:staging'],
  'gha-artifact-publisher': ['repo:strataforge/build-artifacts:environment:release'],
};

export class GithubDeployRolesStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    for (const [roleName, subjects] of Object.entries(PINS)) {
      const role = new iam.Role(this, roleName, {
        roleName,
        maxSessionDuration: Duration.hours(1),
        assumedBy: new iam.WebIdentityPrincipal(PROVIDER, {
          StringEquals: {
```

```
            'token.actions.githubusercontent.com:aud': 'sts.amazonaws.com',
            'token.actions.githubusercontent.com:sub': subjects.length === 1 ? subjects[0] :
subjects,
          },
        }),
      });
      role.applyRemovalPolicy(RemovalPolicy.RETAIN);
    }
  }
}
```

Find real callers, import the existing roles, apply an emergency trust change, and read it back (AWS CLI)

```bash
# GNU date first, then macOS date.
START=$(date -u -d '90 days ago' +%Y-%m-%dT%H:%M:%SZ 2>/dev/null || date -u -v-90d +%Y-%m-
%dT%H:%M:%SZ)
aws cloudtrail lookup-events --region us-east-1 --max-results 50 \
  --lookup-attributes AttributeKey=EventName,AttributeValue=AssumeRoleWithWebIdentity \
  --start-time "$START" \
  --query 'Events[].CloudTrailEvent' --output text | grep -o '"subject":"[^"]*"' | sort | uniq
-c
# Import first (it asks for each role's name and changes nothing else), then deploy.
npx cdk import GithubDeployRolesStack
npx cdk deploy GithubDeployRolesStack
# Emergency only:
aws iam update-assume-role-policy --role-name gha-deploy-fleet-api --policy-document file://
gha-deploy-fleet-api-trust.json
aws iam get-role --role-name gha-deploy-fleet-api --query Role.AssumeRolePolicyDocument
```

How it is verified

- Every trust policy shows both `:aud` and `:sub` conditions with StringEquals and no `*`.
- A workflow in a repository outside `strataforge` fails with `Not authorized to perform sts:AssumeRoleWithWebIdentity`.
- The production deploy workflow still succeeds, and CloudTrail shows its subject as `repo:strataforge/fleet-api:environment:production`.

How to roll it back. Revert the CDK change in `fleet-infra` to restore the previous trust policy for the role; every role in the stack has `RemovalPolicy.RETAIN`, so taking it out of the stack never deletes the role. For an emergency, apply the trust policy you saved before the import (from `aws iam get-role --query Role.AssumeRolePolicyDocument`) with `aws iam update-assume-role-policy`, then reconcile the stack in the next pull request so the console change does not drift. The earlier policy trusts any repository on GitHub, so keep the rollback inside a single deploy window. Verify by running the deploy workflow once and checking that it assumes the role.

SF-SEC-004: Jupyter is open to the internet on 51 research and sandbox instances, some with no token

Critical. Benchmark rating Critical (Security Hub CSPM EC2.19). How it was rated: benchmark Critical (4) +1 exploitability (trivial) +0 blast radius (production) = 5

Where: `crucible-research` (…7088), `kestrel-dev` (…9076), `sandbox-robotics` (…4033), `sandbox-ml` (…4056) in us-east-1, 51 instances. **Controls:** `EC2.18` (Security groups should only allow unrestricted incoming traffic for authorized ports); `EC2.19` (Security groups should not allow unrestricted access to ports with high risk); `EC2.53` (EC2 security groups should not allow ingress from 0.0.0.0/0 to remote server administration ports); `EC2.54` (EC2 security groups should not allow ingress from ::/0 to remote server administration ports). **SOC 2:** CC6.6, C1.1. **Owner:** Hiro Tanaka, Director, Crucible Platform. **Repository:** `crucible-infra`. **Effort:** 10 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- 6 security groups admit 0.0.0.0/0 on 8888 (and 22).
- User data on 9 instances sets NotebookApp.token to an empty string.

Examples: `sg-17b0f4dd28cebff0c`, `sg-5be4bb134e47e145f`, `sg-8a067690b1d6009cd`.

Exploitability. Remote and unauthenticated, with a known technique. Nine instances were launched with an empty notebook token: anyone reaching 8888 runs code. **Blast radius.** Production systems or customer data. Instance roles read the Crucible sensor-log lake and Kestrel model artifacts over IMDSv1.

How we verified it. Read user data with describe-instance-attribute on 07/21/2026; no port scan was run.

Remediation: Take Jupyter off the internet and give researchers Session Manager or SageMaker AI Studio. Jupyter on port 8888 is open to 0.0.0.0/0 in crucible-research, kestrel-dev, and the sandbox accounts, and some servers were launched with an empty token, so anyone can run code on them. Close the port, rotate whatever the notebooks could read, and move researchers to port forwarding or SageMaker AI Studio.

Steps

1. Revoke the 8888 rules open to the internet in every account and Region.
2. Stop notebook servers that run with an empty token, and check their instance profiles and home directories for credentials; rotate anything found.
3. Review CloudTrail for the instance roles of the open notebooks for activity the researchers did not run.

4. Give researchers a Session Manager port-forwarding command for the servers they must keep.
5. Create a SageMaker AI Studio domain in private subnets for the rest, with an execution role scoped to the research data.

Revoke port 8888 from the internet in a Region (AWS CLI)

```bash
REGION=us-east-1
for sg in $(aws ec2 describe-security-groups --region $REGION \
    --filters Name=ip-permission.from-port,Values=8888 Name=ip-permission.cidr,Values=0.0.0.0/0
\
    --query 'SecurityGroups[].GroupId' --output text); do
  aws ec2 revoke-security-group-ingress --region $REGION --group-id "$sg" --protocol tcp --port
8888 --cidr 0.0.0.0/0
done
for sg in $(aws ec2 describe-security-groups --region $REGION \
    --filters Name=ip-permission.from-port,Values=8888 Name=ip-permission.ipv6-cidr,Values=::/0
\
    --query 'SecurityGroups[].GroupId' --output text); do
  aws ec2 revoke-security-group-ingress --region $REGION --group-id "$sg" \
    --ip-permissions 'IpProtocol=tcp,FromPort=8888,ToPort=8888,Ipv6Ranges=[{CidrIpv6=::/0}]'
done
```

Reach a notebook server through Session Manager (AWS CLI)

```bash
aws ssm start-session --target <notebook-instance-id> --document-name AWS-
StartPortForwardingSession \
  --parameters '{"portNumber": ["8888"], "localPortNumber": ["8888"]}'
# Then open http://localhost:8888 and sign in with the token from `jupyter server list`.
```

SageMaker AI Studio domain with no internet-facing access (AWS CDK, TypeScript)

```typescript
import { Stack, StackProps, aws_iam as iam, aws_sagemaker as sagemaker } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class ResearchStudioStack extends Stack {
  constructor(scope: Construct, id: string, props: StackProps & { vpcId: string; subnetIds:
string[] }) {
    super(scope, id, props);
    const role = new iam.Role(this, 'StudioExecutionRole', {
      assumedBy: new iam.ServicePrincipal('sagemaker.amazonaws.com'),
      managedPolicies:
[iam.ManagedPolicy.fromAwsManagedPolicyName('AmazonSageMakerFullAccess')],
    });
    new sagemaker.CfnDomain(this, 'Domain', {
      domainName: 'crucible-research',
      authMode: 'IAM',
      appNetworkAccessType: 'VpcOnly',
      vpcId: props.vpcId,
      subnetIds: props.subnetIds,
      defaultUserSettings: { executionRole: role.roleArn },
    });
  }
}
```

How it is verified

- `aws ec2 describe-security-groups --filters Name=ip-permission.from-port,Values=8888 Name=ip-permission.cidr,Values=0.0.0.0/0` returns nothing in every account and Region.
- Security Hub CSPM EC2.19 passes after the next evaluation.
- A scan from outside the VPC finds nothing listening on port 8888 at any of the old addresses.

How to roll it back. Do not reopen port 8888. If a researcher is blocked, give them a Session Manager port-forwarding command. Remove the SageMaker AI Studio domain by reverting the CDK stack in `crucible-infra`, which does not delete its EFS home directories unless you choose to. Verify by scanning the instance's public address.

SF-SEC-005: 5 EBS snapshots in kestrel-prod are public, and snapshot block public access is off in all 816 account-Regions

Critical. Benchmark rating Critical (Security Hub CSPM EC2.1 and EC2.182). How it was rated: benchmark Critical (4) +1 exploitability (trivial) +0 blast radius (production) = 5

Where: All 48 accounts in 17 Regions, 5 EBS snapshots. **Controls:** `EC2.1` (Amazon EBS snapshots should not be publicly restorable); `EC2.182` (Block public access settings should be enabled for Amazon EBS snapshots). **SOC 2:** CC6.6, C1.1. **Owner:** Jonah Reyes, Director, Perception Platform (formerly Kestrel Vision). **Repository:** `platform-landing-zone`. **Effort:** 6 hours. **Planned:** Week 3, starting Jul 27. **Depends on:** SF-SEC-026.

Evidence

- describe-snapshot-attribute createVolumePermission: Group all.
- All of them are unencrypted: AWS does not allow an encrypted snapshot to be shared publicly, so encryption is not what stands between these and the internet.
- get-snapshot-block-public-access-state: unblocked in every Region of every account.

Examples: `snap-018ef87a05488c7c8`, `snap-09a3ac06503c75deb`, `snap-022c53f8d17b6691b`, `snap-05a2128dd9e475c86`, `snap-067662fec2fb99973`.

Exploitability. Remote and unauthenticated, with a known technique. Any AWS account can create a volume from a public snapshot. **Blast radius.** Production systems or customer data. Volumes of the Kestrel perception API hosts, including application configuration.

How we verified it. Listed the snapshots' source volumes with Jonah Reyes on 07/20/2026; none needs to be shared.

Remediation: Unshare the public EBS snapshots and block public snapshot sharing everywhere. EBS snapshots in kestrel-prod are shared with everyone. Remove the public permission, then turn on snapshot block public access for every account with an EC2 declarative policy so it cannot happen again.

Steps

1. Find public snapshots in each Region of kestrel-prod and record what each contains.
2. Remove the `all` group from each snapshot's createVolumePermission.
3. Enable snapshot block public access with `block-all-sharing` in each Region; note that switching it back to `block-new-sharing` would make existing public snapshots public again.
4. Enable the declarative policy type, create the EC2 policy, and attach it to the root.
5. Treat the contents as disclosed until the owning team confirms otherwise.

Find and unshare public snapshots, then block sharing per Region (AWS CLI)

```bash
for region in us-east-1 eu-west-1; do
  for snap in $(aws ec2 describe-snapshots --region $region --owner-ids self --restorable-by-user-ids all \
                  --query 'Snapshots[].SnapshotId' --output text); do
    aws ec2 modify-snapshot-attribute --region $region --snapshot-id "$snap" \
      --attribute createVolumePermission --operation-type remove --group-names all
  done
  aws ec2 enable-snapshot-block-public-access --region $region --state block-all-sharing
  aws ec2 get-snapshot-block-public-access-state --region $region
done
```

`sf-ec2-block-public-sharing`: no public snapshots, no new public AMIs (EC2 declarative policy)

```JSON
{
  "ec2_attributes": {
    "snapshot_block_public_access": {
      "state": {
        "@@assign": "block_all_sharing"
      }
    },
    "image_block_public_access": {
      "state": {
        "@@assign": "block_new_sharing"
      }
    }
  }
}
```

Enable the policy type, create the policy, and attach it to the root (AWS CLI)

```BASH
ROOT=<org-root-id>
aws organizations enable-policy-type --root-id "$ROOT" --policy-type DECLARATIVE_POLICY_EC2
POLICY=$(aws organizations create-policy --name sf-ec2-block-public-sharing --type
DECLARATIVE_POLICY_EC2 \
  --description "Block public snapshot sharing and new public AMIs" --content file://policies/
declarative/ec2-block-public-sharing.json \
  --query Policy.PolicySummary.Id --output text)
aws organizations attach-policy --policy-id "$POLICY" --target-id "$ROOT"
```

How it is verified

- `aws ec2 describe-snapshots --owner-ids self --restorable-by-user-ids all` returns nothing in every Region of kestrel-prod.
- `aws ec2 get-snapshot-block-public-access-state` returns `block-all-sharing`.
- Security Hub CSPM EC2.182 passes in every account after the next evaluation.

How to roll it back. Making a snapshot public again is a data disclosure, so do not roll this back. If a snapshot must be shared, share it with named accounts through `modify-snapshot-attribute`. To back out the policy, detach the declarative policy from the root and set block public access to `unblocked` in a Region only with a written exception. Verify by listing snapshots with the `all` group.

SF-SEC-006: 4 Kestrel dataset buckets are public: 468.8 million objects of labeled images from customer sites

Critical. Benchmark rating Critical (Security Hub CSPM S3.2). How it was rated: benchmark Critical (4) +1 exploitability (trivial) +0 blast radius (production) = 5

Where: `kestrel-data` (…9098) in us-east-1, 4 S3 buckets. **Controls:** `S3.1` (S3 general purpose buckets should have block public access settings enabled); `S3.2` (S3 general purpose buckets should block public read access); `S3.8` (S3 general purpose buckets should block public access). **SOC 2:** CC6.6, C1.1. **Owner:** Jonah Reyes, Director, Perception Platform (formerly Kestrel Vision). **Repository:** `crucible-infra`. **Effort:** 14 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- Bucket policy grants s3:GetObject and s3:ListBucket to "Principal": "*"; bucket and account Block Public Access off.
- Server access logging is off, so the only record of downloads is the organization trail's S3 data events, which cover these buckets only since the Kestrel accounts joined in November 2025.
- The buckets fed a 2024 public benchmark that Kestrel withdrew; public access was never turned off.

Examples: `sf-labeled-datasets-image-datasets-9098-us-east-1-000`, `sf-labeled-datasets-image-datasets-9098-us-east-1-001`, `sf-labeled-datasets-image-datasets-9098-us-east-1-002`, `sf-labeled-datasets-image-datasets-9098-us-east-1-004`.

Exploitability. Remote and unauthenticated, with a known technique. Anyone on the internet can list and download them. **Blast radius.** Production systems or customer data. 361 TB of images from customer sites; about 4% of a 200-image sample show people or badge IDs.

How we verified it. Confirmed with Jonah Reyes on 07/20/2026; no product or partner reads the buckets anonymously.

Remediation: Close the public Kestrel dataset buckets and keep them closed. The four dataset buckets the finding names in kestrel-data are open to the internet. Block public access first, remove the public policy and ACLs, and treat the contents as disclosed until the access history says otherwise.

Steps

1. Save each bucket's policy, ACL, and public access block settings to the incident folder as evidence before changing anything.
2. Turn on all four S3 Block Public Access settings on each bucket; that stops public reads at once without touching the objects.

3. Delete the public bucket policy, set the bucket ACL to private, and switch object ownership to BucketOwnerEnforced so ACLs stop working.
4. Put the TLS-only policy on each bucket; access for the Crucible training roles comes from their IAM policies, so the bucket policy needs no Allow.
5. Work out the exposure window: the organization trail has recorded S3 data events for these buckets since the Kestrel accounts joined in November 2025; before that, only the bucket creation dates are known. Give Legal the list of datasets and the outside IP addresses that read them.
6. Turn on server access logging for these buckets (see s3-logging) and move them into the crucible-infra CDK app with `cdk import`.

Evidence, then close the public buckets (AWS CLI)

```bash
for b in <dataset-bucket-1> <dataset-bucket-2> <dataset-bucket-3> <dataset-bucket-4>; do
  mkdir -p evidence/$b
  aws s3api get-bucket-policy --bucket $b > evidence/$b/policy.json 2>&1
  aws s3api get-bucket-acl --bucket $b > evidence/$b/acl.json
  aws s3api put-public-access-block --bucket $b \
    --public-access-block-configuration
BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets=true
  aws s3api delete-bucket-policy --bucket $b
  aws s3api put-bucket-ownership-controls --bucket $b --ownership-controls
'Rules=[{ObjectOwnership=BucketOwnerEnforced}]'
  sed "s/<dataset-bucket-name>/$b/g" tls-only.json > tls-only-$b.json
  aws s3api put-bucket-policy --bucket $b --policy file://tls-only-$b.json
done
```

tls-only.json: the TLS-only policy (the only Deny a bucket policy may carry) (S3 bucket policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyNonTlsRequests",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::<dataset-bucket-name>",
        "arn:aws:s3:::<dataset-bucket-name>/*"
      ],
      "Condition": {
        "Bool": {
          "aws:SecureTransport": "false"
        }
      }
    }
  ]
}
```

The bucket as code in crucible-infra (AWS CDK, TypeScript)

```TYPESCRIPT
import { RemovalPolicy, Stack, StackProps, aws_s3 as s3 } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class LabeledDatasetsStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new s3.Bucket(this, 'ImageDatasets1', {
      bucketName: '<dataset-bucket-1>',
      blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
      enforceSSL: true,
      objectOwnership: s3.ObjectOwnership.BUCKET_OWNER_ENFORCED,
      encryption: s3.BucketEncryption.KMS,
      versioned: true,
      removalPolicy: RemovalPolicy.RETAIN,
    });
  }
}
```

How it is verified

- `aws s3api get-public-access-block` shows all four settings true on each of these buckets (Security Hub CSPM S3.1, S3.8).
- An anonymous `curl` of a known object URL returns 403.
- `aws s3api get-bucket-policy-status` reports `IsPublic` false and the ACL grants nothing to AllUsers or AuthenticatedUsers.

How to roll it back. Re-opening a bucket that held the exposed datasets is not a rollback anyone should perform. If a partner needs access, grant it by IAM role, a presigned URL, or a CloudFront distribution with origin access control. To undo a technical step, restore the saved policy, ACL, and public access block settings from the incident folder; keep Block Public Access on. Verify with an unauthenticated request and the account's public access report.

SF-SEC-007: A Kestrel MySQL snapshot with annotation records is restorable by any AWS account

Critical. Benchmark rating Critical (Security Hub CSPM RDS.1). How it was rated: benchmark Critical (4) +1 exploitability (trivial) +0 blast radius (production) = 5

Where: `kestrel-prod` (…9031) in us-east-1, 1 DB snapshot. **Controls:** `RDS.1` (RDS snapshot should be private). **SOC 2:** CC6.6, C1.1. **Owner:** Jonah Reyes, Director, Perception Platform (formerly Kestrel Vision). **Repository:** `crucible-infra`. **Effort:** 3 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- describe-db-snapshot-attributes: restore = all.
- Unencrypted MySQL 5.7 snapshot taken 10/28/2025 before a migration.

Examples: `kestrel-annotations-2025-10-28-premigration`, `source: kestrel-annotations-db (MySQL 5.7.44)`, `attribute restore: all`.

Exploitability. Remote and unauthenticated, with a known technique. Any AWS account can restore it into its own database. **Blast radius.** Production systems or customer data. Annotator identities and customer-site labels for 1.9 million images.

How we verified it. Jonah Reyes confirmed on 07/20/2026 that it was shared with a labeling vendor using "all" by mistake.

Remediation: Make the Kestrel annotations snapshot private and restrict who can share snapshots. The manual MySQL 5.7 snapshot `kestrel-annotations-2025-10-28-premigration` in kestrel-prod is shared with all AWS accounts (RDS.1). Remove the public attribute, then limit who can change snapshot sharing.

Steps

1. Remove the `all` value from the snapshot's `restore` attribute in every Region where it exists.
2. Ask the Perception Platform team whether the snapshot is still needed; if so, copy it with a customer managed key and delete the original after a hold.
3. Treat the data as disclosed for the time it was shared, and give Legal the date range from the snapshot creation date.

4. Attach `sf-protect-account-defaults` (see scp-guardrails); its ProtectRdsSnapshotSharing statement lets only the platform roles change snapshot sharing.
5. Deploy the Config rule that reports any public snapshot.

Remove public sharing and read the attribute back (AWS CLI)

```bash
REGION=us-east-1
aws rds modify-db-snapshot-attribute --region $REGION \
  --db-snapshot-identifier kestrel-annotations-2025-10-28-premigration \
  --attribute-name restore --values-to-remove all
aws rds describe-db-snapshot-attributes --region $REGION \
  --db-snapshot-identifier kestrel-annotations-2025-10-28-premigration \
  --query 'DBSnapshotAttributesResult.DBSnapshotAttributes'
aws rds copy-db-snapshot --region $REGION --source-db-snapshot-identifier kestrel-
annotations-2025-10-28-premigration \
  --target-db-snapshot-identifier kestrel-annotations-2025-10-28-premigration-private --kms-
key-id alias/sf-rds
aws rds wait db-snapshot-available --region $REGION --db-snapshot-identifier kestrel-
annotations-2025-10-28-premigration-private
```

sf-protect-account-defaults, statement ProtectRdsSnapshotSharing: only the platform roles may change snapshot sharing (the whole policy is in the guardrail SCPs) (Service control policy)

```json
{
  "Sid": "ProtectRdsSnapshotSharing",
  "Effect": "Deny",
  "Action": [
    "rds:ModifyDBSnapshotAttribute",
    "rds:ModifyDBClusterSnapshotAttribute"
  ],
  "Resource": "*",
  "Condition": {
    "ArnNotLike": {
      "aws:PrincipalArn": [
        "arn:aws:iam::*:role/PlatformAdmin",
        "arn:aws:iam::*:role/AWSControlTowerExecution",
        "arn:aws:iam::*:role/stacksets-exec-*"
      ]
    }
  }
}
```

Config rule that reports public RDS snapshots (RDS.1) (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Report RDS snapshots that are public.
Resources:
  RdsSnapshotsPublicProhibited:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sf-rds-snapshots-public-prohibited
      Source:
        Owner: AWS
        SourceIdentifier: RDS_SNAPSHOTS_PUBLIC_PROHIBITED
```

How it is verified

- `describe-db-snapshot-attributes` shows an empty `restore` attribute value list.
- `rds:ModifyDBSnapshotAttribute` from an ordinary administrator role is denied by `sf-protect-account-defaults`.
- Security Hub CSPM RDS.1 passes in kestrel-prod after the next evaluation.

How to roll it back. Do not make the snapshot public again. Share it with specific account IDs through `aws rds modify-db-snapshot-attribute --values-to-add`, and copy it with a customer managed key first if it carries personal data. Revert the SCP and the Config rule stacks last. Verify by reading the snapshot's `restore` attribute.

SF-SEC-008: Kestrel's Jenkins controller is open to the internet on 8080 and 22

Critical. Benchmark rating Critical (Security Hub CSPM EC2.19). How it was rated: benchmark Critical (4) +1 exploitability (trivial) +0 blast radius (production) = 5

Where: `kestrel-legacy-mgmt` (…9120) in us-east-1, 1 instance. **Controls:** `EC2.18` (Security groups should only allow unrestricted incoming traffic for authorized ports); `EC2.19` (Security groups should not allow unrestricted access to ports with high risk); `EC2.53` (EC2 security groups should not allow ingress from 0.0.0.0/0 to remote server administration ports). **SOC 2:** CC6.6, CC8.1. **Owner:** Jonah Reyes, Director, Perception Platform (formerly Kestrel Vision). **Repository:** `crucible-infra`. **Effort:** 16 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- Security group admits 0.0.0.0/0 on 8080 and 22; public IPv4 address attached.
- Credentials store holds 31 AWS access keys, including the kestrel-legacy-mgmt root key.

Examples: `i-0b89690e9c5782dbe`, `sg-0fe1cb106cfe0abd4`.

Exploitability. Remote and unauthenticated, with a known technique. Jenkins 2.3× with anonymous read of job configuration; the instance allows IMDSv1. **Blast radius.** Production systems or customer data. Jenkins holds the Kestrel CI keys and a root access key for kestrel-legacy-mgmt.

How we verified it. Read the controller's configuration from its launch template and AMI notes with Jonah Reyes on 07/20/2026.

Remediation: Lock down the Kestrel Jenkins controller, then retire it or put it behind OIDC sign-in. The Kestrel Jenkins controller `i-0b89690e9c5782dbe` in kestrel-legacy-mgmt accepts 8080 and 22 from 0.0.0.0/0 and still allows IMDSv1. Close the ports now, require IMDSv2, rotate every credential Jenkins holds, and move the jobs to GitHub Actions or put the controller behind an internal load balancer with Okta sign-in.

Steps

1. Revoke the 0.0.0.0/0 rules on ports 8080 and 22 today, and require IMDSv2 on the instance.
2. Treat every credential stored in Jenkins as exposed: list the credentials store, the AWS keys, GitHub tokens, and SSH keys, and rotate each one.
3. Check CloudTrail for the Jenkins instance profile and any IAM user keys it holds for unfamiliar activity since the ports were opened.
4. Move the jobs to GitHub Actions with OIDC roles (see oidc-trust-sub), and shut down the controller when the last job has moved.
5. If the controller must stay for a while, put it behind an internal ALB that authenticates with Okta through OIDC, and allow ports only from the ALB.

Close the ports, require IMDSv2, and list the credentials to rotate (AWS CLI)

```bash
REGION=us-east-1
SG=$(aws ec2 describe-instances --region $REGION --instance-ids i-0b89690e9c5782dbe \
  --query 'Reservations[0].Instances[0].SecurityGroups[0].GroupId' --output text)
for port in 8080 22; do
  aws ec2 revoke-security-group-ingress --region $REGION --group-id "$SG" --protocol tcp --port
$port --cidr 0.0.0.0/0
done
aws ec2 modify-instance-metadata-options --region $REGION --instance-id i-0b89690e9c5782dbe \
  --http-tokens required --http-endpoint enabled --http-put-response-hop-limit 1
aws iam list-access-keys --user-name <jenkins-iam-user-name>
aws iam list-role-policies --role-name <jenkins-instance-role-name>
```

Internal ALB that signs people in with Okta before reaching Jenkins (AWS CDK, TypeScript)

```typescript
import { Duration, SecretValue, Stack, StackProps, aws_ec2 as ec2, aws_elasticloadbalancingv2
as elbv2, aws_elasticloadbalancingv2_targets as targets } from 'aws-cdk-lib';
import * as actions from 'aws-cdk-lib/aws-elasticloadbalancingv2-actions';
import { Construct } from 'constructs';

export class JenkinsFrontStack extends Stack {
  constructor(scope: Construct, id: string, props: StackProps & { vpc: ec2.IVpc }) {
    super(scope, id, props);
    const alb = new elbv2.ApplicationLoadBalancer(this, 'Alb', { vpc: props.vpc,
internetFacing: false, dropInvalidHeaderFields: true });
    const jenkins = new elbv2.ApplicationTargetGroup(this, 'Jenkins', {
      vpc: props.vpc, port: 8080, protocol: elbv2.ApplicationProtocol.HTTP,
      targets: [new targets.InstanceIdTarget('i-0b89690e9c5782dbe', 8080)],
    });
    alb.addListener('Https', {
      port: 443,
      certificates: [elbv2.ListenerCertificate.fromArn('<acm-certificate-arn>')],
      defaultAction: new actions.AuthenticateOidcAction({
        issuer: 'https://strataforge.okta.example',
        authorizationEndpoint: 'https://strataforge.okta.example/oauth2/v1/authorize',
        tokenEndpoint: 'https://strataforge.okta.example/oauth2/v1/token',
        userInfoEndpoint: 'https://strataforge.okta.example/oauth2/v1/userinfo',
        clientId: '<okta-client-id>',
        clientSecret: SecretValue.secretsManager('jenkins/okta-client-secret'),
        sessionTimeout: Duration.hours(8),
        next: elbv2.ListenerAction.forward([jenkins]),
      }),
    });
  }
}
```

How it is verified

- `aws ec2 describe-security-groups --group-ids <jenkins-security-group-id>` shows no rule open to 0.0.0.0/0 or ::/0.
- `aws ec2 describe-instances` shows `HttpTokens` required for `i-0b89690e9c5782dbe` (Security Hub CSPM EC2.8).
- Every credential in the Jenkins credentials store has a rotation record, and the old values are rejected by their services.

How to roll it back. Do not reopen ports 8080 or 22 to the internet. If the new internal load balancer fails, allow the corporate egress range on the security group while it is repaired. IMDSv2 can be set back to optional with `modify-instance-metadata-options --http-tokens optional`, but leave it required. Rotated credentials stay rotated. Verify with a job run and a read of the security group.

SF-SEC-009: RDP is open to the internet on 32 PLM CAD workstations that hold product designs

Critical. Benchmark rating Critical (Security Hub CSPM EC2.19, EC2.53, EC2.54, and EC2.18). How it was rated: benchmark Critical (4) +0 exploitability (moderate) +0 blast radius (production) = 4

Where: `plm-prod` (…8061) in us-east-1, 3 security groups. **Controls:** `EC2.18` (Security groups should only allow unrestricted incoming traffic for authorized ports); `EC2.19` (Security groups should not allow unrestricted access to ports with high risk); `EC2.53` (EC2 security groups should not allow ingress from 0.0.0.0/0 to remote server administration ports). **SOC 2:** CC6.6. **Owner:** Mei Chen, Director, Enterprise Applications. **Repository:** `enterprise-infra`. **Effort:** 10 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- 3 security groups admit 0.0.0.0/0 on port 3389; 32 instances attached.
- 0 also admit ::/0.

Examples: `sg-2c5e87e22c5c89f25`, `sg-b03911a6cba76892c`, `sg-b82c5ba721c9ae80c`, `i-0959515d2c7aca427`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). RDP brute force and credential stuffing against Windows hosts. **Blast radius.** Production systems or customer data. Product designs for every robot line.

How we verified it. Matched each rule to its owner and use with the account owner.

Remediation: Close SSH and RDP to the internet and use Session Manager instead. Security groups on bastions, PLM CAD workstations, and dev and sandbox instances allow SSH (22) and RDP (3389) from 0.0.0.0/0 and ::/0 (EC2.53, EC2.54, EC2.19, EC2.18). Revoke the rules, give instances the Systems Manager role, and reach them through Session Manager and Fleet Manager remote desktop.

Steps

1. Find every security group rule that opens 22 or 3389 to `0.0.0.0/0` or `::/0` in each account and Region.
2. Attach the Systems Manager managed instance role to the affected instances and confirm each shows as a managed node.
3. Revoke the open rules, IPv4 and IPv6 both, and keep a copy of each rule for rollback.
4. Give engineers `ssm:StartSession` on tagged instances through their permission set, and RDP users a port-forwarding session or Fleet Manager remote desktop.

5. Change the CDK security group constructs so no construct can open an administration port to the internet, then remove the bastion hosts that Session Manager replaces.

Find and revoke the open rules in one Region (AWS CLI)

```bash
REGION=us-east-1
for port in 22 3389; do
  for sg in $(aws ec2 describe-security-groups --region $REGION \
      --filters Name=ip-permission.from-port,Values=$port Name=ip-
permission.cidr,Values=0.0.0.0/0 \
      --query 'SecurityGroups[].GroupId' --output text); do
    aws ec2 revoke-security-group-ingress --region $REGION --group-id "$sg" --protocol tcp --
port $port --cidr 0.0.0.0/0
  done
  for sg in $(aws ec2 describe-security-groups --region $REGION \
      --filters Name=ip-permission.from-port,Values=$port Name=ip-permission.ipv6-
cidr,Values=::/0 \
      --query 'SecurityGroups[].GroupId' --output text); do
    aws ec2 revoke-security-group-ingress --region $REGION --group-id "$sg" \
      --ip-permissions "IpProtocol=tcp,FromPort=$port,ToPort=$port,Ipv6Ranges=[{CidrIpv6=::/
0}]"
  done
done
```

Instance with the Systems Manager role and no inbound rules (AWS CDK, TypeScript)

```typescript
import { Stack, StackProps, aws_ec2 as ec2, aws_iam as iam } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class CadWorkstationStack extends Stack {
  constructor(scope: Construct, id: string, props: StackProps & { vpc: ec2.IVpc }) {
    super(scope, id, props);
    const sg = new ec2.SecurityGroup(this, 'WorkstationSg', { vpc: props.vpc, allowAllOutbound:
true, description: 'No inbound rules; access is through Session Manager' });
    const role = new iam.Role(this, 'WorkstationRole', {
      assumedBy: new iam.ServicePrincipal('ec2.amazonaws.com'),
      managedPolicies:
[iam.ManagedPolicy.fromAwsManagedPolicyName('AmazonSSMManagedInstanceCore')],
    });
    new ec2.Instance(this, 'Workstation', {
      vpc: props.vpc,
      instanceType: new ec2.InstanceType('<instance-type>'),
      machineImage:
ec2.MachineImage.latestWindows(ec2.WindowsVersion.WINDOWS_SERVER_2022_ENGLISH_FULL_BASE),
      securityGroup: sg,
      role,
      requireImdsv2: true,
    });
  }
}
```

Reach SSH and RDP through Session Manager (AWS CLI)

```bash
aws ssm start-session --target <instance-id>
aws ssm start-session --target <instance-id> --document-name AWS-StartPortForwardingSession \
  --parameters '{"portNumber": ["3389"], "localPortNumber": ["53389"]}'
# Then connect the RDP client to localhost:53389.
```

How it is verified

- `aws ec2 describe-security-groups --filters Name=ip-permission.from-port,Values=22,3389 Name=ip-permission.cidr,Values=0.0.0.0/0` returns nothing in every Region, and the same for `ip-permission.ipv6-cidr` with `::/0`.
- Security Hub CSPM EC2.53, EC2.54, EC2.19, and EC2.18 pass after the next evaluation.
- Engineers reach each instance through Session Manager, and the session appears in the audit log.

How to roll it back. Restore a revoked rule from the copy kept in step 3 with `aws ec2 authorize-security-group-ingress`, limited to the corporate egress range rather than the internet, for an incident only, and remove it when the incident closes. Revert the infrastructure change in this finding's repository last. Verify by connecting through Session Manager and by scanning the instance's public address.

SF-SEC-010: Root users of 3 Kestrel accounts have access keys and no MFA

Critical. Benchmark rating Critical (Security Hub CSPM IAM.4). How it was rated: benchmark Critical (4) +0 exploitability (moderate) +0 blast radius (production) = 4

Where: `kestrel-prod` (…9031), `kestrel-data` (…9098), `kestrel-legacy-mgmt` (…9120) in us-east-1, 3 root users. **Controls:** `IAM.4` (IAM root user access key should not exist); `IAM.6` (Hardware MFA should be enabled for the root user); `IAM.9` (MFA should be enabled for the root user). **SOC 2:** CC6.1, CC6.2. **Owner:** Grace Liu, Manager, Identity and Access. **Repository:** `platform-landing-zone`. **Effort:** 8 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- get-account-summary: AccountAccessKeysPresent = 1 and AccountMFAEnabled = 0 in 3 accounts.
- Most recent root key use: 06/30/2026 (kestrel-legacy-mgmt).
- Centralized root access management is not enabled for the organization.

Examples: `kestrel-prod (…9031)`, `kestrel-data (…9098)`, `kestrel-legacy-mgmt (…9120)`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). A root key used by a Jenkins job last month lives in Jenkins' credential store. **Blast radius.** Production systems or customer data. Root of kestrel-prod and kestrel-data, which hold the perception APIs and the labeled datasets.

How we verified it. Confirmed with Jonah Reyes on 07/20/2026; nobody at StrataForge holds these root passwords.

Remediation: Delete member-account root credentials centrally and protect the management root with hardware MFA. Some Kestrel accounts have root access keys, some member accounts have no root MFA, and elsewhere root uses a virtual MFA device. Centralized root access management removes root credentials from every member account, so only the management account root needs a hardware MFA device.

Steps

1. From the management account, enable trusted access for IAM, then root credentials management and root sessions.
2. Audit each member account's root credentials with a short root session using the IAMAuditRootUserCredentials task policy.
3. Delete the root login profile, access keys, signing certificates, and MFA devices in each member account with the IAMDeleteRootUserCredentials task policy.
4. Deploy the EventBridge rule for sts:AssumeRoot so every root session pages the security team.
5. Register a FIDO2 hardware key on the management account root user in the console, remove the virtual device, and keep a second key at the second site.
6. Apply the member-account root deny SCP (see scp-guardrails) only after this cleanup finishes.

Enable centralized root access management (management account) (AWS CLI)

```bash
aws organizations enable-aws-service-access --service-principal iam.amazonaws.com
aws iam enable-organizations-root-credentials-management
aws iam enable-organizations-root-sessions
aws iam list-organizations-features
```

Audit, then delete one member account's root credentials (AWS CLI)

```bash
MEMBER=<member-account-id>
REGION=us-east-1
# The global STS endpoint is not supported for AssumeRoot; use a Regional endpoint.
root_session() {
  aws sts assume-root --region "$REGION" --endpoint-url "https://sts.${REGION}.amazonaws.com" \
    --target-principal "$MEMBER" --task-policy-arn "arn=arn:aws:iam::aws:policy/root-task/$1" \
    --duration-seconds 900 --query Credentials --output json
}
use_session() {
  export AWS_ACCESS_KEY_ID=$(echo "$1" | jq -r .AccessKeyId) AWS_SECRET_ACCESS_KEY=$(echo "$1"
| jq -r .SecretAccessKey)
  export AWS_SESSION_TOKEN=$(echo "$1" | jq -r .SessionToken)
}
(use_session "$(root_session IAMAuditRootUserCredentials)"
 aws iam get-account-summary --query 'SummaryMap.
[AccountAccessKeysPresent,AccountMFAEnabled,AccountSigningCertificatesPresent]')
(use_session "$(root_session IAMDeleteRootUserCredentials)"
 for key in $(aws iam list-access-keys --query 'AccessKeyMetadata[].AccessKeyId' --output
text); do aws iam delete-access-key --access-key-id "$key"; done
 for cert in $(aws iam list-signing-certificates --query 'Certificates[].CertificateId' --
output text); do aws iam delete-signing-certificate --certificate-id "$cert"; done
 for serial in $(aws iam list-mfa-devices --query 'MFADevices[].SerialNumber' --output text);
do aws iam deactivate-mfa-device --serial-number "$serial"; done
 aws iam delete-login-profile)
```

Page on every root session started through sts:AssumeRoot (EventBridge event pattern)

```json
{
  "source": [
    "aws.sts"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "sts.amazonaws.com"
    ],
    "eventName": [
      "AssumeRoot"
    ]
  }
}
```

How it is verified

- `aws iam list-organizations-features` lists RootCredentialsManagement and RootSessions.
- In each member account, `aws iam get-account-summary` shows AccountAccessKeysPresent 0 (Security Hub CSPM IAM.4) and the root login profile is gone.

- Security Hub CSPM controls IAM.4, IAM.9, and IAM.6 pass in the management account once the hardware key is registered.

How to roll it back. Deleted member-account root credentials cannot be restored, and that is the point of the change. To get root access back to one account, start a root session through `sts:AssumeRoot` from the management account, or reset the root password through the account's root email. Detach the member-root deny SCP from the affected account first. The EventBridge rule and the hardware key on the management root can be removed by reverting their stacks, but leave them in place. Verify with a root session that lists the account's root credentials.

High findings

20 High findings, 438 hours between them.

SF-SEC-011: docs-legacy.strataforge.example points at a deleted S3 website bucket that anyone can claim

High. Benchmark rating Medium (Vantalect baseline: subdomain takeover). How it was rated: benchmark Medium (2) +1 exploitability (trivial) +0 blast radius (production) = 3

Where: `sf-shared-services` (…1188) in us-east-1, 1 DNS record. **Controls:** No benchmark control covers this. **SOC 2:** CC6.6. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 3 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- The bucket was deleted in 2024 when the documentation moved; the record stayed.
- Swept all 2,418 records in the sf-shared-services hosted zones; this is the only dangling one.

Examples: `docs-legacy.strataforge.example CNAME docs-legacy.strataforge.example.s3-website-us-east-1.amazonaws.com`, `hosted zone strataforge.example (sf-shared-services)`, `missing bucket docs-legacy.strataforge.example`.

Exploitability. Remote and unauthenticated, with a known technique. Creating a bucket with that name in us-east-1 serves any content on the subdomain. **Blast radius.** Production systems or customer data. A StrataForge hostname customers trust; phishing and session-cookie risk for the parent domain.

How we verified it. Confirmed with dig and head-bucket (NoSuchBucket) on 07/21/2026; the bucket name was not claimed.

Remediation: Delete the dangling `docs-legacy` record and sweep Route 53 for the same mistake. `docs-legacy.strataforge.example` still points at an S3 website endpoint whose bucket was deleted in 2024, so anyone who creates a bucket with that name can serve content on a StrataForge subdomain. Delete the record, then sweep every hosted zone for records that point at AWS resources that no longer exist.

Steps

1. Delete the CNAME from the hosted zone in sf-shared-services; do not recreate the bucket, because that only hides the problem.
2. Sweep every hosted zone for CNAME and alias records that point at S3 website endpoints, CloudFront distributions, Elastic Beanstalk environments, and Elastic IP names.
3. Check each target exists; delete the record when it does not, and ask the owner when it is not clear.
4. Add the sweep to the monthly platform review, and require that a record be deleted in the same change that deletes its target.

Delete the dangling record (AWS CLI)

```bash
ZONE=<hosted-zone-id>
cat > delete-docs-legacy.json <<'JSON'
{"Comment": "Remove dangling S3 website record",
 "Changes": [{"Action": "DELETE", "ResourceRecordSet": {
   "Name": "docs-legacy.strataforge.example.", "Type": "CNAME", "TTL": <record-ttl>,
   "ResourceRecords": [{"Value": "docs-legacy.strataforge.example.s3-website-us-
east-1.amazonaws.com"}]}}]}
JSON
aws route53 change-resource-record-sets --hosted-zone-id "$ZONE" --change-batch file://delete-
docs-legacy.json
```

Sweep for S3 website records whose bucket no longer exists (AWS CLI)

```bash
for zone in $(aws route53 list-hosted-zones --query 'HostedZones[].Id' --output text); do
  aws route53 list-resource-record-sets --hosted-zone-id "$zone" \
    --query "ResourceRecordSets[?Type=='CNAME' && contains(ResourceRecords[0].Value, 's3-
website')].[Name,ResourceRecords[0].Value]" --output text |
  while read -r name target; do
    bucket=${target%%.s3-website*}
    if ! aws s3api head-bucket --bucket "$bucket" 2>/dev/null; then echo "DANGLING $name ->
$target"; fi
  done
done
```

How it is verified

- `dig docs-legacy.strataforge.example CNAME` returns no answer.
- The sweep prints no DANGLING line for any hosted zone.
- The monthly review checklist includes the sweep, with the date of the last run.

How to roll it back. Before step 1, save the record with `aws route53 list-resource-record-sets` to the change record. To roll back, recreate it from that copy, using `aws route53 change-resource-record-sets` with action UPSERT. Recreate it only if a live target exists first; recreating a record that points at a deleted bucket brings the takeover risk back. Verify by resolving the name and requesting the target.

SF-SEC-012: 1,009 active access keys are older than 90 days; 400 are over three years old

High. Benchmark rating Medium (Security Hub CSPM IAM.3, IAM.22, and IAM.2). How it was rated: benchmark Medium (2) +0 exploitability (moderate) +1 blast radius (organization) = 3

Where: All 48 accounts in us-east-1, 984 IAM users. **Controls:** `IAM.2` (IAM users should not have IAM policies attached); `IAM.3` (IAM users' access keys should be rotated every 90 days or less); `IAM.22` (IAM user credentials unused for 45 days should be removed). **SOC 2:** CC6.1, CC6.2, CC9.2. **Owner:** Grace Liu, Manager, Identity and Access. **Repository:** `platform-landing-zone`. **Effort:** 63 hours. **Planned:** Week 4, starting Aug 3; the rest (Replace the factory gateway users' keys with IAM Roles Anywhere during factory maintenance windows) is due November 20, 2026. **Depends on:** SF-SEC-003, SF-SEC-008.

Evidence

- 1,394 IAM users; 1,139 active keys; 984 users fail IAM.3.
- 673 users have credentials unused for 45 days or more (IAM.22).
- 874 users have policies attached directly (IAM.2).

Examples: `prasmussen (sf-management)`, `kquintero (sf-management)`, `cokonkwo (sf-management)`, `iadler (sf-management)`, `svc-batch-loader-00 (sf-management)`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Long-lived keys leak through CI logs, laptops, and former staff. **Blast radius.** Every account in the organization. The 90 CI users in sf-cicd can deploy to 31 accounts through the CDK deploy roles.

How we verified it. Sampled 60 keys with their owning teams on 07/21–22/2026; 41 belong to CI or vendor integrations.

Remediation: Retire stale IAM access keys and stop new IAM users from being created. Many IAM users carry access keys older than 90 days or unused for 45 days or more, and some hold attached policies. Triage from the credential report, deactivate before deleting, replace CI keys with OIDC roles, and block new users and keys.

Steps

1. Generate the credential report in every account and list keys older than 90 days (IAM.3) and credentials unused for 45 days or more (IAM.22).
2. Deactivate keys unused for 45 days or more, identifying each key by its creation time (the credential report names a slot, not a key ID), and record `user,key-id` for every key you deactivate. Wait 14 days for anyone to object, then delete only the keys that are still Inactive.
3. For keys still in use, find the workload and replace it: CI keys with OIDC roles (see oidc-trust-sub), workload keys with roles.
4. Save every attached and inline user policy to disk, then detach or delete exactly what was saved, and grant access through groups or roles instead (IAM.2).
5. Attach `sf-protect-account-defaults` (see scp-guardrails); its NoNewIamUsers statement denies iam:CreateUser and iam:CreateAccessKey outside the PlatformIdentityAdmin role. Deploy the AWS Config rule that reports keys over 90 days.

Triage keys by age and last use from the credential report (AWS CLI)

```bash
aws iam generate-credential-report >/dev/null; sleep 5
aws iam get-credential-report --query Content --output text | base64 --decode > credential-report.csv
# GNU date first, then macOS date.
OLD=$(date -u -d '90 days ago' +%Y-%m-%dT%H:%M:%S+00:00 2>/dev/null || date -u -v-90d +%Y-%m-%dT%H:%M:%S+00:00)
UNUSED=$(date -u -d '45 days ago' +%Y-%m-%dT%H:%M:%S+00:00 2>/dev/null || date -u -v-45d +%Y-%m-%dT%H:%M:%S+00:00)
# Key 1: 9 active, 10 last_rotated, 11 last_used_date. Key 2: 14, 15, 16. A key never used counts as unused only once it is 45 days old.
awk -F, -v old="$OLD" 'NR > 1 && $9 == "true" && $10 < old {print $1 ",key1," $10} NR > 1 && $14 == "true" && $15 < old {print $1 ",key2," $15}' \
  credential-report.csv > keys-older-than-90-days.csv
awk -F, -v cutoff="$UNUSED" 'NR > 1 && $9 == "true" && (($11 == "N/A" && $10 < cutoff) || ($11 != "N/A" && $11 < cutoff)) {print $1 ",key1," $10} NR > 1 && $14 == "true" && (($16 == "N/A" && $15 < cutoff) || ($16 != "N/A" && $16 < cutoff)) {print $1 ",key2," $15}' \
  credential-report.csv > keys-unused-45-days.csv
wc -l keys-older-than-90-days.csv keys-unused-45-days.csv
```

Deactivate the exact key, delete it after the hold, and save then detach user policies (AWS CLI)

```bash
```

```
set -uo pipefail
# keys-unused-45-days.csv lines are "user,slot,last_rotated"; last_rotated is the key's
creation time, which identifies the key.
# The credential report names a slot, not a key ID, so match the creation time instead of
taking the user's first active key.
: > keys-deactivated.csv
while IFS=, read -r user slot rotated; do
  key=$(aws iam list-access-keys --user-name "$user" --output json |
    jq -r --arg t "${rotated:0:19}" '[.AccessKeyMetadata[] | select(.Status == "Active" and
(.CreateDate | startswith($t)))] | if length == 1 then .[0].AccessKeyId else empty end')
  if [ -z "$key" ]; then echo "SKIP $user $slot: no single active key created at $rotated" >&2;
continue; fi
  aws iam update-access-key --user-name "$user" --access-key-id "$key" --status Inactive ||
{ echo "FAILED $user $key" >&2; continue; }
  echo "$user,$key,$(date -u +%F)" >> keys-deactivated.csv
done < keys-unused-45-days.csv

# After the 14-day hold: write keys-to-delete.csv ("user,key-id") from keys-deactivated.csv for
keys nobody has reactivated.
: > keys-to-delete.csv
while IFS=, read -r user key day; do
  status=$(aws iam list-access-keys --user-name "$user" --query "AccessKeyMetadata[?
AccessKeyId=='$key'].Status | [0]" --output text)
  [ "$status" = "Inactive" ] && echo "$user,$key" >> keys-to-delete.csv
done < keys-deactivated.csv
while IFS=, read -r user key; do aws iam delete-access-key --user-name "$user" --access-key-id
"$key"; done < keys-to-delete.csv

# Policies: save every one to disk, and detach or delete only what was saved.
mkdir -p saved-user-policies
aws iam list-users --query 'Users[].UserName' --output text | tr '\t' '\n' > all-users.txt
: > users-with-policies.txt
while read -r user; do
  attached=$(aws iam list-attached-user-policies --user-name "$user" --query
'AttachedPolicies[].PolicyArn' --output text)
  inline=$(aws iam list-user-policies --user-name "$user" --query 'PolicyNames[]' --output
text)
  [ -z "$attached$inline" ] && continue
  echo "$user" >> users-with-policies.txt
  printf '%s\n' $attached > "saved-user-policies/$user.attached.txt"
  for name in $inline; do
    aws iam get-user-policy --user-name "$user" --policy-name "$name" --output json > "saved-
user-policies/$user.$name.json" \
      && test -s "saved-user-policies/$user.$name.json" || { echo "SAVE FAILED $user $name;
nothing deleted for this user" >&2; continue 2; }
  done
  for arn in $attached; do aws iam detach-user-policy --user-name "$user" --policy-arn "$arn";
done
  for name in $inline; do aws iam delete-user-policy --user-name "$user" --policy-name "$name";
done
done < all-users.txt
```

sf-protect-account-defaults, statement NoNewIamUsers: no new IAM users or access keys outside the identity administrator role (the whole policy is in the guardrail SCPs) (Service control policy)

```JSON
{
  "Sid": "NoNewIamUsers",
  "Effect": "Deny",
  "Action": [
    "iam:CreateUser",
    "iam:CreateAccessKey",
    "iam:CreateLoginProfile",
    "iam:UpdateLoginProfile"
  ],
  "Resource": "*",
  "Condition": {
    "ArnNotLike": {
      "aws:PrincipalArn": [
        "arn:aws:iam::*:role/PlatformIdentityAdmin"
      ]
    }
  }
}
```

Config rule that reports keys older than 90 days (CloudFormation)

```YAML
AWSTemplateFormatVersion: '2010-09-09'
Description: Report IAM access keys older than 90 days (Security Hub CSPM IAM.3).
Resources:
  AccessKeysRotated:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sf-access-keys-rotated
      InputParameters:
        maxAccessKeyAge: '90'
      Source:
        Owner: AWS
        SourceIdentifier: ACCESS_KEYS_ROTATED
      MaximumExecutionFrequency: TwentyFour_Hours
```

How it is verified

- A fresh credential report shows no active key older than 90 days or unused for 45 days or more (IAM.3, IAM.22).
- `aws iam list-attached-user-policies` returns nothing for every user (IAM.2).
- `aws iam create-user` from an ordinary administrator role is denied by `sf-protect-account-defaults`.

How to roll it back. Reactivate a deactivated key with `aws iam update-access-key --status Active`, using the `user,key-id` line in `keys-deactivated.csv`; this works until the key is deleted after the 14-day hold. A deleted key cannot return: the PlatformIdentityAdmin role (the only one `sf-protect-account-defaults` lets create access keys) issues a new key for the workload, and the

migration to an OIDC role is filed. Re-attach detached policies from `saved-user-policies/<user>.attached.txt` with `aws iam attach-user-policy`, and recreate each deleted inline policy with `aws iam put-user-policy` from the saved `saved-user-policies/<user>.<name>.json` file. Revert the SCP and the Config rule stacks in `platform-landing-zone` to undo the block on new users. Verify with the credential report.

SF-SEC-013: 108 IAM users sign in to the console without MFA, 2 of them in the management account

High. Benchmark rating Medium (Security Hub CSPM IAM.5). How it was rated: benchmark Medium (2) +0 exploitability (moderate) +1 blast radius (organization) = 3

Where: 23 accounts in us-east-1, 108 IAM users. **Controls:** `IAM.5` (MFA should be enabled for all IAM users that have a console password). **SOC 2:** CC6.1, CC6.2. **Owner:** Grace Liu, Manager, Identity and Access. **Repository:** `platform-landing-zone`. **Effort:** 22 hours. **Planned:** Week 4, starting Aug 3. **Depends on:** SF-SEC-010, SF-SEC-029.

Evidence

- Credential report: 108 users with password_enabled true and mfa_active false.
- Legacy Okta SAML federation still grants admin in 6 accounts outside Identity Center.

Examples: `vcho (sf-management)`, `prasmussen (sf-management)`, `vcho (sf-management)`, `prasmussen (sf-management)`, `ftran (sf-shared-services)`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Password spraying or phishing; no sign-in IP restrictions. **Blast radius.** Every account in the organization. The two management-account users are administrators of the organization.

How we verified it. Matched every management-account user to a person with Grace Liu on 07/21/2026.

Remediation: Move every console user without MFA to IAM Identity Center and delete their IAM passwords. IAM users, including some in the management account, have console passwords and no MFA. People should sign in through IAM Identity Center with Okta, so remove their login profiles and block new ones.

Steps

1. Pull the credential report and list every user with a password and no MFA.
2. Assign each person to the right IAM Identity Center permission sets through Okta groups, and have them sign in through the access portal.
3. Delete the IAM login profile of each person who has signed in through Identity Center at least once.

4. Attach `sf-protect-account-defaults` (see scp-guardrails); its NoNewIamUsers statement lets only the PlatformIdentityAdmin role create or change login profiles.
5. Keep passwords only for `breakglass-01` and `breakglass-02`, each with a hardware MFA device (see break-glass).

Find users with a password and no MFA from the credential report (AWS CLI)

```bash
aws iam generate-credential-report >/dev/null; sleep 5
aws iam get-credential-report --query Content --output text | base64 --decode > credential-report.csv
# Columns: 1 user, 4 password_enabled, 8 mfa_active.
awk -F, 'NR > 1 && $4 == "true" && $8 == "false" {print $1}' credential-report.csv > console-users-without-mfa.txt
wc -l console-users-without-mfa.txt
```

Assign an Okta group to a permission set, then delete the login profile (AWS CLI)

```bash
aws sso-admin create-account-assignment --instance-arn <identity-center-instance-arn> \
  --target-id <member-account-id> --target-type AWS_ACCOUNT --permission-set-arn <permission-set-arn> \
  --principal-type GROUP --principal-id <identity-center-group-id>
while read -r user; do aws iam delete-login-profile --user-name "$user"; done < users-confirmed-on-identity-center.txt
```

sf-protect-account-defaults, statement NoNewIamUsers: console passwords only through the identity administrator role (the whole policy is in the guardrail SCPs) (Service control policy)

```json
{
  "Sid": "NoNewIamUsers",
  "Effect": "Deny",
  "Action": [
    "iam:CreateUser",
    "iam:CreateAccessKey",
    "iam:CreateLoginProfile",
    "iam:UpdateLoginProfile"
  ],
  "Resource": "*",
  "Condition": {
    "ArnNotLike": {
      "aws:PrincipalArn": [
        "arn:aws:iam::*:role/PlatformIdentityAdmin"
      ]
    }
  }
}
```

How it is verified

- A new credential report shows no user with `password_enabled` true and `mfa_active` false (Security Hub CSPM IAM.5).
- `iam:CreateLoginProfile` from an administrator who is not the identity administrator role is denied by `sf-protect-account-defaults`.
- The Identity Center sign-in log shows each moved person signing in through Okta.

How to roll it back. Recreate a user's console password as the PlatformIdentityAdmin role, the only role `sf-protect-account-defaults` lets create a login profile, with `aws iam create-login-profile --user-name <user-name> --password file://temporary-password.txt --password-reset-required`, and register a virtual MFA device at the first sign-in. Revert the SCP in `platform-landing-zone` to let the platform role manage login profiles again. Deleting a login profile loses the old password for good, which is intended. Verify by signing in as the user and checking the credential report.

SF-SEC-014: 1,631 instances still allow IMDSv1

High. Benchmark rating High (Security Hub CSPM EC2.8). How it was rated: benchmark High (3) +0 exploitability (moderate) +0 blast radius (production) = 3

Where: 40 accounts in 5 Regions, 1,631 instances. **Controls:** `EC2.8` (EC2 instances should use Instance Metadata Service Version 2 (IMDSv2)). **SOC 2:** CC6.6. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 56 hours. **Planned:** Week 3, starting Jul 27; the rest (Switch the instances that still make IMDSv1 calls after their agents and SDKs are upgraded) is due October 30, 2026.

Evidence

- HttpTokens optional on 1,631 instances in 40 accounts.
- CloudWatch MetadataNoToken shows IMDSv1 calls in the last 30 days on 359 of them; the rest can switch now.

Examples: `i-0d1943bd61e363589`, `i-0cc4bba908f4d3f8f`, `i-0eca43426406f9caf`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). One SSRF in any application reaches the instance role's credentials. **Blast radius.** Production systems or customer data. Includes EKS nodes whose role is an account administrator.

How we verified it. Read from the scan.

Remediation: Require IMDSv2 on every instance, starting with an organization-wide default. Many instances still allow IMDSv1 (EC2.8), which lets a server-side request forgery bug read instance credentials. An EC2 declarative policy makes IMDSv2 the default for new launches, and existing instances are changed one at a time once the `MetadataNoToken` metric shows nothing uses v1.

Steps

1. Create the declarative policy for instance metadata defaults and attach it to the root; it applies to new launches only, and existing instances keep their settings.
2. For each existing instance, read the `MetadataNoToken` CloudWatch metric for 14 days; a nonzero sum means something still calls IMDSv1.
3. Update SDKs and agents that use IMDSv1, then run `modify-instance-metadata-options --http-tokens required`; use a hop limit of 2 for containers that read metadata.
4. Set the account default per Region with `modify-instance-metadata-defaults` as a second layer.
5. Set `requireImdsv2` in every CDK instance and launch template construct.

sf-ec2-imdsv2-default: IMDSv2 required for new launches, hop limit 2 (EC2 declarative policy)

```JSON
{
  "ec2_attributes": {
    "instance_metadata_defaults": {
      "http_tokens": {
        "@@assign": "required"
      },
      "http_put_response_hop_limit": {
        "@@assign": "2"
      }
    }
  }
}
```

Attach the policy, check usage, then change an instance and the Region default (AWS CLI)

```bash
ROOT=<org-root-id>
aws organizations enable-policy-type --root-id "$ROOT" --policy-type DECLARATIVE_POLICY_EC2
POLICY=$(aws organizations create-policy --name sf-ec2-imdsv2-default --type
DECLARATIVE_POLICY_EC2 \
  --description "Require IMDSv2 on new instances" --content file://policies/declarative/ec2-
imdsv2-default.json \
  --query Policy.PolicySummary.Id --output text)
aws organizations attach-policy --policy-id "$POLICY" --target-id "$ROOT"

INSTANCE=<instance-id>
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name MetadataNoToken \
  --dimensions Name=InstanceId,Value=$INSTANCE --start-time "$(date -u -d '14 days ago' +%Y-%m-
%dT%H:%M:%SZ)" \
  --end-time "$(date -u +%Y-%m-%dT%H:%M:%SZ)" --period 86400 --statistics Sum --query
'Datapoints[].Sum'
aws ec2 modify-instance-metadata-options --instance-id $INSTANCE --http-tokens required --http-
endpoint enabled --http-put-response-hop-limit 2
aws ec2 modify-instance-metadata-defaults --http-tokens required --http-put-response-hop-limit
2
```

IMDSv2 required in code (AWS CDK, TypeScript)

```typescript
import { Stack, StackProps, aws_ec2 as ec2 } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class ApiServerStack extends Stack {
  constructor(scope: Construct, id: string, props: StackProps & { vpc: ec2.IVpc }) {
    super(scope, id, props);
    new ec2.Instance(this, 'ApiServer', {
      vpc: props.vpc,
      instanceType: new ec2.InstanceType('<instance-type>'),
      machineImage: ec2.MachineImage.latestAmazonLinux2023(),
      requireImdsv2: true,
    });
    new ec2.LaunchTemplate(this, 'ApiTemplate', {
      machineImage: ec2.MachineImage.latestAmazonLinux2023(),
      instanceType: new ec2.InstanceType('<instance-type>'),
      requireImdsv2: true,
    });
  }
}
```

How it is verified

- `aws ec2 describe-instances --query 'Reservations[].Instances[?MetadataOptions.HttpTokens!=required].InstanceId'` returns an empty list in every account and Region (Security Hub CSPM EC2.8).
- A new instance launched without any metadata option comes up with `HttpTokens` required.

- The `MetadataNoToken` metric is zero for changed instances, and no application error appears in the week after.

How to roll it back. Set an instance back to optional tokens with `aws ec2 modify-instance-metadata-options --http-tokens optional`, which applies at once and needs no restart. Detach the declarative policy from the root to stop the default for new launches. Do this only for an application that still calls IMDSv1, and file the fix for that application. Verify with the `MetadataNoToken` metric.

SF-SEC-015: 144 Lambda functions have secret-named environment variables

High. Benchmark rating High (Vantalect baseline: secrets in plaintext configuration). How it was rated: benchmark High (3) +0 exploitability (moderate) +0 blast radius (production) = 3

Where: `fleet-api-prod` (…5044), `customer-portal-prod` (…5215), `fleet-dev` (…6037), `kestrel-prod` (…9031) in 2 Regions, 144 Lambda functions. **Controls:** No benchmark control covers this. **SOC 2:** CC6.1, C1.1. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 32 hours. **Planned:** Week 4, starting Aug 3; the rest (Move the fleet-dev functions' secrets to Secrets Manager) is due September 30, 2026.

Evidence

- Variable names only were collected; values never left the accounts.
- Most common: GITHUB_TOKEN (37), DATABASE_URL (32), DB_PASSWORD (28), PARTNER_API_KEY (25), SAP_RFC_PASSWORD (22).

Examples: `fleet-api-1a9533`, `fleet-api-773352`, `fleet-api-7f26a6`, `GITHUB_TOKEN (37)`, `DATABASE_URL (32)`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Anyone with lambda:GetFunctionConfiguration (the ReadOnly permission set has it) reads them in plaintext. **Blast radius.** Production systems or customer data. Database, partner API, GitHub, and SAP RFC credentials.

How we verified it. Owning teams confirmed the variables hold values, except three customer-portal references (false positive).

Remediation: Move Lambda secrets from environment variables into Secrets Manager and rotate them. Lambda functions keep `DATABASE_URL`, `PARTNER_API_KEY`, `GITHUB_TOKEN`, `DB_PASSWORD`, or `SAP_RFC_PASSWORD` in plain environment variables, readable by anyone with lambda:GetFunctionConfiguration. Store them in Secrets Manager, grant read to the function role, and rotate every value that was exposed.

Steps

1. List the functions and the names of their secret-named variables (names only; never print values).
2. Create one Secrets Manager secret per credential in the owning account, encrypted with a customer managed key.
3. Change the function code to read the secret at start-up and cache it, and pass only the secret's name in the environment.
4. Grant the function role read access on that one secret with `grantRead` in CDK.
5. Rotate every credential that lived in an environment variable, because it has been readable since it was set, then remove the old variables.

List the secret-named variables, without their values (AWS CLI)

```bash
for region in us-east-1 us-west-2; do
  aws lambda list-functions --region "$region" \
    --query 'Functions[?Environment.Variables!=`null`].{name: FunctionName, vars:
keys(Environment.Variables)}' --output json |
    jq -r --arg region "$region" '.[] | select(.vars | map(test("PASSWORD|SECRET|TOKEN|API_KEY|
DATABASE_URL|PRIVATE_KEY"; "i")) | any) | [$region, .name, (.vars | join(";"))] | @csv'
done > lambda-secret-variables.csv
wc -l lambda-secret-variables.csv
```

Secret, function, and read grant (AWS CDK, TypeScript)

```typescript
import { Duration, Stack, StackProps, aws_kms as kms, aws_lambda as lambda, aws_secretsmanager
as secretsmanager } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class PartnerSyncStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const key = kms.Key.fromLookup(this, 'SecretsKey', { aliasName: 'alias/sf-secrets' });
    const partnerKey = new secretsmanager.Secret(this, 'PartnerApiKey', { secretName: 'fleet/
partner-sync/partner-api-key', encryptionKey: key });
    const fn = new lambda.Function(this, 'PartnerSync', {
      runtime: lambda.Runtime.NODEJS_24_X,
      handler: 'index.handler',
      code: lambda.Code.fromAsset('dist/partner-sync'),
      timeout: Duration.seconds(30),
      environment: { PARTNER_API_KEY_SECRET_ID: partnerKey.secretName },
    });
    partnerKey.grantRead(fn);
  }
}
```

Create the secret from a file, set rotation, and remove the old variable (AWS CLI)

```bash
aws secretsmanager create-secret --name fleet/partner-sync/partner-api-key --kms-key-id alias/
sf-secrets \
  --secret-string file://new-partner-api-key.txt
# For database credentials, rotate on a schedule with the rotation function for your engine.
aws secretsmanager rotate-secret --secret-id fleet/portal/db-password \
  --rotation-lambda-arn <rotation-lambda-arn> --rotation-rules AutomaticallyAfterDays=30
# This replaces the whole variable map, so list every variable the function keeps.
aws lambda update-function-configuration --function-name <function-name> \
  --environment 'Variables={PARTNER_API_KEY_SECRET_ID=fleet/partner-sync/partner-api-
key,LOG_LEVEL=info}'
```

How it is verified

- Re-running the inventory returns no function whose secret-named variable holds a value instead of a secret name.
- `aws lambda get-function-configuration` for a migrated function shows only a secret name in its environment.
- The old credential values are rejected by the partner, GitHub, and the database, which proves the rotation happened.

How to roll it back. Redeploy the previous function version, which still reads the environment variable, by repointing the alias with `aws lambda update-alias --function-name <function-name> --name <alias-name> --function-version <previous-version>`. Do this before the old variable is removed in step 5. Once removed, put the rotated value back as the variable, never the old one, because the old value was exposed. Revert the CDK change last. Verify with a test invocation and the function's error rate.

SF-SEC-016: All 16 EKS cluster API endpoints accept connections from 0.0.0.0/0

High. Benchmark rating High (Security Hub CSPM EKS.1). How it was rated: benchmark High (3) +0 exploitability (moderate) +0 blast radius (production) = 3

Where: 10 accounts in 5 Regions, 16 EKS clusters. **Controls:** `EKS.1` (EKS cluster endpoints should not be publicly accessible). **SOC 2:** CC6.6. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 38 hours. **Planned:** Week 3, starting Jul 27; the rest (Make the non-production endpoints private once CI runners reach them through the Transit Gateway) is due September 30, 2026.

Evidence

- endpointPublicAccess true, publicAccessCidrs 0.0.0.0/0, endpointPrivateAccess false.

Examples: `telemetry-us-east-1-c37`, `telemetry-eu-central-1-c9d`, `telemetry-ap-northeast-1-732`, `telemetry-ap-southeast-1-6c2`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Needs valid credentials, but the API is reachable from anywhere. **Blast radius.** Production systems or customer data. Fleet API, telemetry, and digital-twin clusters.

How we verified it. Read from the scan.

Remediation: Restrict EKS API endpoints to private access plus the corporate egress range. EKS clusters expose a public API endpoint open to 0.0.0.0/0 (EKS.1). Turn on private endpoint access, limit the public endpoint to the corporate egress range, and move to private-only once CI runs inside the VPC.

Steps

1. List every cluster in every account and Region and its `resourcesVpcConfig` endpoint settings.
2. Turn on private access first, so nodes keep reaching the API server inside the VPC.
3. Set `publicAccessCidrs` to the corporate egress range (get the real CIDR from the network team; the examples use the `<corporate-egress-cidr>` placeholder); the update takes several minutes per cluster.
4. Move deploy jobs onto runners inside the VPC (the sf-cicd account), then set `endpointPublicAccess` to false.
5. Set the endpoint access in the CDK cluster constructs so a rebuild keeps it.

Private access on, public access limited to the corporate range (AWS CLI)

```bash
REGION=us-east-1
for cluster in $(aws eks list-clusters --region $REGION --query 'clusters[]' --output text); do
  aws eks update-cluster-config --region $REGION --name "$cluster" \
    --resources-vpc-config
endpointPrivateAccess=true,endpointPublicAccess=true,publicAccessCidrs=<corporate-egress-cidr>
  aws eks wait cluster-active --region $REGION --name "$cluster"
done
# When CI runs inside the VPC, close the public endpoint:
aws eks update-cluster-config --region $REGION --name <cluster-name> \
  --resources-vpc-config endpointPrivateAccess=true,endpointPublicAccess=false
```

Cluster endpoint access in code (AWS CDK, TypeScript)

```typescript
import { Stack, StackProps, aws_eks as eks, aws_ec2 as ec2 } from 'aws-cdk-lib';
import { Construct } from 'constructs';
// The kubectl layer package must match the cluster version: @aws-cdk/lambda-layer-kubectl-
v<minor>.
import { KubectlLayer } from '@aws-cdk/lambda-layer-kubectl-v<kubernetes-minor-version>';

interface FleetApiClusterProps extends StackProps {
  vpc: ec2.IVpc;
  corporateEgressCidr: string; // from the network team, passed in rather than written into the
stack
}

export class FleetApiClusterStack extends Stack {
  constructor(scope: Construct, id: string, props: FleetApiClusterProps) {
    super(scope, id, props);
    new eks.Cluster(this, 'Cluster', {
      vpc: props.vpc,
      // Use the version the cluster runs today, and never one in extended support ($0.60 per
cluster-hour).
      version: eks.KubernetesVersion.of('<kubernetes-version>'),
      kubectlLayer: new KubectlLayer(this, 'Kubectl'),
      endpointAccess:
eks.EndpointAccess.PUBLIC_AND_PRIVATE.onlyFrom(props.corporateEgressCidr),
      // After CI moves into the VPC: endpointAccess: eks.EndpointAccess.PRIVATE,
    });
  }
}
```

How it is verified

- `aws eks describe-cluster --name <cluster-name> --query cluster.resourcesVpcConfig` shows `endpointPrivateAccess` true and `publicAccessCidrs` of only the corporate egress range, or `endpointPublicAccess` false.
- Security Hub CSPM EKS.1 passes after the next evaluation.
- Nodes stay Ready and the deploy pipeline still reaches the cluster.

How to roll it back. Set `publicAccessCidrs` back to `0.0.0.0/0` with `aws eks update-cluster-config` only for an incident, because it reopens the API server to the internet. Private endpoint access can stay on. Each update takes several minutes per cluster, so roll back one cluster at a time. Verify by running `kubectl get nodes` from the CI runner.

SF-SEC-017: 14 customer managed policies grant Action * on Resource * to 17 workload roles and 46 IAM users

High. Benchmark rating High (Security Hub CSPM IAM.1). How it was rated: benchmark High (3) +0 exploitability (moderate) +0 blast radius (production) = 3

Where: 14 accounts in us-east-1, 14 IAM policies. **Controls:** `IAM.1` (IAM policies should not allow full "*" administrative privileges). **SOC 2:** CC6.3. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `fleet-infra`. **Effort:** 40 hours. **Planned:** Week 4, starting Aug 3.

Evidence

- 14 policies with Action * on Resource * in 14 accounts.
- fleet-api-node-role is attached to every fleet-api EKS node, and those nodes allow IMDSv1 with hop limit 2.

Examples: `fleet-api-node-admin (fleet-api-prod)`, `ingest-ops-admin (fleet-ingest-prod)`, `crucible-batch-admin (crucible-sim-prod)`, `research-batch-admin (crucible-research)`, `sap-backup-admin (erp-sap-prod)`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). A compromised workload (an SSRF over IMDSv1, a malicious pod) becomes account administrator. **Blast radius.** Production systems or customer data. Includes the fleet-api EKS node role and the Kestrel perception API.

How we verified it. Generated least-privilege policies from 90 days of CloudTrail for the six largest roles with IAM Access Analyzer.

Remediation: Replace wildcard workload policies with scoped ones generated from CloudTrail. Workload roles carry `"Action": "*"` (IAM.1) and customer managed policies with service wildcards such as s3:* (IAM.21), including the EKS node role `fleet-api-node-role`. Generate least-privilege policies from CloudTrail with IAM Access Analyzer and cap each role with the one `sf-workload-boundary` permissions boundary (defined in the privilege-escalation entry), which allows every service action and takes away only identity and organization actions.

Steps

1. List roles with an `Action: *` statement and customer managed policies with `service:*` actions, and rank them by production impact.
2. Start an IAM Access Analyzer policy generation for each role from the organization trail's last 90 days.
3. Review the generated policy with the owning team, add the actions CloudTrail does not record, and replace the wildcard policy with it.
4. Move pod permissions off the node role by giving each workload its own role through EKS Pod Identity.
5. `fleet-api-node-role` already exists, and CloudFormation cannot create a role whose name is taken, so bring it under the stack with `cdk import` (the stack sets `RemovalPolicy.RETAIN` on it, which an import requires) before the first `cdk deploy`.

6. Publish `sf-workload-boundary` (see privilege-escalation; it is one document for every role) and attach it. In a non-production account first, run `simulate-principal-policy` for the VPC CNI's network-interface and IP actions and for `eks-auth:AssumeRoleForPodIdentity`: the boundary must leave all of them allowed, or new pods get no IP address and Pod Identity stops working.

Generate a policy from 90 days of CloudTrail for fleet-api-node-role (AWS CLI)

```bash
cat > trail-details.json <<'JSON'
{"trails": [{"cloudTrailArn": "arn:aws:cloudtrail:us-east-1:<management-account-id>:trail/sf-org-trail", "allRegions": true}],
 "accessRole": "arn:aws:iam::<fleet-api-prod-account-id>:role/AccessAnalyzerMonitorServiceRole",
 "startTime": "<start-timestamp-iso-8601>", "endTime": "<end-timestamp-iso-8601>"}
JSON
JOB=$(aws accessanalyzer start-policy-generation \
  --policy-generation-details principalArn=arn:aws:iam::<fleet-api-prod-account-id>:role/fleet-api-node-role \
  --cloud-trail-details file://trail-details.json --query jobId --output text)
aws accessanalyzer get-generated-policy --job-id "$JOB" --include-resource-placeholders --include-service-level-template
```

Scoped node role with a permissions boundary, imported rather than created (AWS CDK, TypeScript)

```typescript
import { RemovalPolicy, Stack, StackProps, aws_iam as iam } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class FleetApiNodeRoleStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const boundary = iam.ManagedPolicy.fromManagedPolicyArn(this, 'Boundary', 'arn:aws:iam::<fleet-api-prod-account-id>:policy/sf-workload-boundary');
    const nodeRole = new iam.Role(this, 'NodeRole', {
      roleName: 'fleet-api-node-role',
      assumedBy: new iam.ServicePrincipal('ec2.amazonaws.com'),
      managedPolicies: ['AmazonEKSWorkerNodePolicy', 'AmazonEKS_CNI_Policy', 'AmazonEC2ContainerRegistryReadOnly']
        .map((name) => iam.ManagedPolicy.fromAwsManagedPolicyName(name)),
    });
    nodeRole.applyRemovalPolicy(RemovalPolicy.RETAIN);
    iam.PermissionsBoundary.of(nodeRole).apply(boundary);
  }
}
```

Import the existing role, deploy, and prove the node role still works in non-production (AWS CLI)

```bash
npx cdk import FleetApiNodeRoleStack
npx cdk deploy FleetApiNodeRoleStack
# Non-production first: every action below must come back "allowed".
aws iam simulate-principal-policy --profile <fleet-api-staging-profile> \
  --policy-source-arn arn:aws:iam::<fleet-api-staging-account-id>:role/fleet-api-node-role \
  --action-names ec2:CreateNetworkInterface ec2:AttachNetworkInterface
ec2:AssignPrivateIpAddresses ec2:DescribeNetworkInterfaces \
                eks-auth:AssumeRoleForPodIdentity ecr:GetAuthorizationToken \
  --query 'EvaluationResults[].[EvalActionName,EvalDecision]' --output table
```

sf-workload-boundary: the one document, shared with the privilege-escalation entry (IAM policy)

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEverythingExceptIdentityAndOrganization",
      "Effect": "Allow",
      "NotAction": [
        "iam:*",
        "organizations:*",
        "account:*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "PassOnlyWorkloadRolesToWorkloadServices",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::*:role/sf-workload-*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "lambda.amazonaws.com",
            "ecs-tasks.amazonaws.com",
            "states.amazonaws.com",
            "sagemaker.amazonaws.com",
            "iot.amazonaws.com",
            "events.amazonaws.com",
            "scheduler.amazonaws.com",
            "firehose.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid": "DenyEscalation",
      "Effect": "Deny",
      "Action": [
        "iam:CreatePolicyVersion",
        "iam:SetDefaultPolicyVersion",
```

```
        "iam:PutUserPolicy",
        "iam:PutRolePolicy",
        "iam:PutGroupPolicy",
        "iam:AttachUserPolicy",
        "iam:AttachRolePolicy",
        "iam:AttachGroupPolicy",
        "iam:UpdateAssumeRolePolicy",
        "iam:PutRolePermissionsBoundary",
        "iam:PutUserPermissionsBoundary",
        "iam:DeleteRolePermissionsBoundary",
        "iam:DeleteUserPermissionsBoundary"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DenyChangesToTheBoundaryPolicy",
      "Effect": "Deny",
      "Action": [
        "iam:CreatePolicyVersion",
        "iam:SetDefaultPolicyVersion",
        "iam:DeletePolicy",
        "iam:DeletePolicyVersion"
      ],
      "Resource": "arn:aws:iam::*:policy/sf-workload-boundary"
    }
  ]
}
```

How it is verified

- Security Hub CSPM IAM.1 and IAM.21 show no failed resource in fleet-api-prod.
- The generated policy replaced the wildcard policy, and the service ran a full release cycle with no new AccessDenied errors in CloudTrail.
- `aws iam get-role --role-name fleet-api-node-role` shows `PermissionsBoundary` set, `simulate-principal-policy` allows the CNI and Pod Identity actions, and a new pod in a non-production cluster gets an IP address.

How to roll it back. Before step 3, save the wildcard policy document to the change record. To roll back, re-attach it and remove the permissions boundary from the role with `aws iam delete-role-permissions-boundary`, run as a platform role because `sf-require-workload-boundary` denies that call to everyone else. The role is held by `RemovalPolicy.RETAIN`, so reverting the CDK change in `fleet-infra` and deploying it never deletes `fleet-api-node-role`. Roll back one role at a time, starting with the one that is failing, and file the missing actions as a gap in the generated policy instead of restoring the wildcard for the long term. Verify with the workload's smoke test and by reading CloudTrail for AccessDenied events on the role.

SF-SEC-018: 12 privilege-escalation paths lead to administrator in 11 accounts

High. Benchmark rating High (Vantalect privilege-escalation path analysis). How it was rated: benchmark High (3) +0 exploitability (moderate) +0 blast radius (production) = 3

Where: 11 accounts in us-east-1, 12 paths. **Controls:** No benchmark control covers this. **SOC 2:** CC6.3. **Owner:** Grace Liu, Manager, Identity and Access. **Repository:** `platform-landing-zone`. **Effort:** 28 hours. **Planned:** Week 4, starting Aug 3.

Evidence

- PE-01: FleetDeveloper permission set (212) -> fleet-dev-admin-exec (Action * on *)
- PE-02: FleetDeveloper permission set (212) -> cdk-hnb659fds-cfn-exec-role (AdministratorAccess)
- PE-03: crucible-research IAM users (58) -> crucible-batch-admin-profile (Action * on *, read on crucible-data)
- PE-04: jenkins-deployer user (1) -> Account administrator

Examples: `PE-01: iam:PassRole + lambda:CreateFunction + lambda:InvokeFunction (fleet-dev)`, `PE-02: iam:PassRole + lambda:CreateFunction + lambda:InvokeFunction (fleet-staging)`, `PE-03: iam:PassRole + ec2:RunInstances (crucible-research)`, `PE-04: iam:CreatePolicyVersion (suspended-legacy-ci)`, `PE-05: iam:PutUserPolicy on self, then sts:AssumeRole into Kestrel accounts (kestrel-legacy-mgmt)`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Each path needs one principal's credentials; hundreds of people hold one. **Blast radius.** Production systems or customer data. Every path ends at an administrator role or user.

How we verified it. Each path was confirmed with the IAM policy simulator, not executed.

Remediation: Close IAM privilege-escalation paths with one permissions boundary and scoped PassRole. Several roles can raise their own privileges: iam:PassRole with lambda:CreateFunction or ec2:RunInstances, iam:CreatePolicyVersion, iam:PutUserPolicy on themselves, and iam:UpdateAssumeRolePolicy. One boundary, `sf-workload-boundary`, allows every service action except IAM, Organizations, and Account, adds back only a scoped iam:PassRole, and denies the escalation actions and any change to itself. An SCP forces the boundary onto new roles and protects it, and Identity Center permission sets take it through a permission-set boundary.

Steps

1. Run `simulate-principal-policy` for the escalation actions against every role the scan flagged, to confirm each path.
2. Publish `sf-workload-boundary` as the same document everywhere: a service-managed stack set for every member account, and a stack of its own in sf-management, because service-managed stack sets never deploy to the management account. Its first statement allows everything except `iam:*`, `organizations:*`, and `account:*`, so the VPC CNI, EKS Pod Identity, SSM, IoT, SageMaker, and Kinesis keep working.
3. Scope every remaining iam:PassRole to named `sf-workload-*` roles with `iam:PassedToService`, and never `Resource: *`.
4. Attach the boundary to the flagged IAM roles and users with `put-role-permissions-boundary`. The roles behind Identity Center (`AWSReservedSSO_`) reject that call, so attach it to the permission set instead with `sso-admin put-permissions-boundary-to-permission-set`, and re-provision the permission set.
5. Set the boundary on every CDK app (`PermissionsBoundary.of(app).apply(...)`) so roles the pipelines create carry it, then attach `sf-require-workload-boundary`, which makes new roles and users carry the boundary and denies anyone but the platform roles from deleting or editing it.
6. Before production, run `simulate-principal-policy` against a representative role of each kind (a pipeline role, an EKS node role, a Lambda execution role) in a non-production account and confirm the service actions stay allowed and the escalation actions are denied.

sf-workload-boundary: allow everything except identity and organization actions, scoped PassRole, no self-escalation, the boundary protected (IAM policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEverythingExceptIdentityAndOrganization",
      "Effect": "Allow",
      "NotAction": [
        "iam:*",
        "organizations:*",
        "account:*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "PassOnlyWorkloadRolesToWorkloadServices",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::*:role/sf-workload-*",
```

```
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "lambda.amazonaws.com",
            "ecs-tasks.amazonaws.com",
            "states.amazonaws.com",
            "sagemaker.amazonaws.com",
            "iot.amazonaws.com",
            "events.amazonaws.com",
            "scheduler.amazonaws.com",
            "firehose.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid": "DenyEscalation",
      "Effect": "Deny",
      "Action": [
        "iam:CreatePolicyVersion",
        "iam:SetDefaultPolicyVersion",
        "iam:PutUserPolicy",
        "iam:PutRolePolicy",
        "iam:PutGroupPolicy",
        "iam:AttachUserPolicy",
        "iam:AttachRolePolicy",
        "iam:AttachGroupPolicy",
        "iam:UpdateAssumeRolePolicy",
        "iam:PutRolePermissionsBoundary",
        "iam:PutUserPermissionsBoundary",
        "iam:DeleteRolePermissionsBoundary",
        "iam:DeleteUserPermissionsBoundary"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DenyChangesToTheBoundaryPolicy",
      "Effect": "Deny",
      "Action": [
        "iam:CreatePolicyVersion",
        "iam:SetDefaultPolicyVersion",
        "iam:DeletePolicy",
        "iam:DeletePolicyVersion"
      ],
      "Resource": "arn:aws:iam::*:policy/sf-workload-boundary"
    }
  ]
}
```

The boundary as a stack set template (the same text goes in a stack in sf-management) (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: sf-workload-boundary, the one permissions boundary for workload roles.
```

```
Resources:
  WorkloadBoundary:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      ManagedPolicyName: sf-workload-boundary
      PolicyDocument:
        {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AllowEverythingExceptIdentityAndOrganization",
              "Effect": "Allow",
              "NotAction": [
                "iam:*",
                "organizations:*",
                "account:*"
              ],
              "Resource": "*"
            },
            {
              "Sid": "PassOnlyWorkloadRolesToWorkloadServices",
              "Effect": "Allow",
              "Action": "iam:PassRole",
              "Resource": "arn:aws:iam::*:role/sf-workload-*",
              "Condition": {
                "StringEquals": {
                  "iam:PassedToService": [
                    "lambda.amazonaws.com",
                    "ecs-tasks.amazonaws.com",
                    "states.amazonaws.com",
                    "sagemaker.amazonaws.com",
                    "iot.amazonaws.com",
                    "events.amazonaws.com",
                    "scheduler.amazonaws.com",
                    "firehose.amazonaws.com"
                  ]
                }
              }
            },
            {
              "Sid": "DenyEscalation",
              "Effect": "Deny",
              "Action": [
                "iam:CreatePolicyVersion",
                "iam:SetDefaultPolicyVersion",
                "iam:PutUserPolicy",
                "iam:PutRolePolicy",
                "iam:PutGroupPolicy",
                "iam:AttachUserPolicy",
                "iam:AttachRolePolicy",
                "iam:AttachGroupPolicy",
                "iam:UpdateAssumeRolePolicy",
                "iam:PutRolePermissionsBoundary",
                "iam:PutUserPermissionsBoundary",
                "iam:DeleteRolePermissionsBoundary",
                "iam:DeleteUserPermissionsBoundary"
              ],
```

```
            "Resource": "*"
          },
          {
            "Sid": "DenyChangesToTheBoundaryPolicy",
            "Effect": "Deny",
            "Action": [
              "iam:CreatePolicyVersion",
              "iam:SetDefaultPolicyVersion",
              "iam:DeletePolicy",
              "iam:DeletePolicyVersion"
            ],
            "Resource": "arn:aws:iam::*:policy/sf-workload-boundary"
          }
        ]
      }
```

sf-require-workload-boundary: new roles and users must carry the boundary, and only the platform roles may edit it (Service control policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RequireWorkloadBoundary",
      "Effect": "Deny",
      "Resource": "*",
      "Action": [
        "iam:CreateRole",
        "iam:CreateUser",
        "iam:PutRolePermissionsBoundary",
        "iam:PutUserPermissionsBoundary"
      ],
      "Condition": {
        "StringNotLike": {
          "iam:PermissionsBoundary": "arn:aws:iam::*:policy/sf-workload-boundary"
        },
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::*:role/PlatformAdmin",
            "arn:aws:iam::*:role/AWSControlTowerExecution",
            "arn:aws:iam::*:role/stacksets-exec-*",
            "arn:aws:iam::*:role/PlatformIdentityAdmin"
          ]
        }
      }
    },
    {
      "Sid": "ProtectWorkloadBoundary",
      "Effect": "Deny",
      "Action": [
        "iam:CreatePolicyVersion",
        "iam:SetDefaultPolicyVersion",
        "iam:DeletePolicy",
        "iam:DeletePolicyVersion"
```

```
      ],
      "Resource": "arn:aws:iam::*:policy/sf-workload-boundary",
      "Condition": {
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::*:role/PlatformAdmin",
            "arn:aws:iam::*:role/AWSControlTowerExecution",
            "arn:aws:iam::*:role/stacksets-exec-*",
            "arn:aws:iam::*:role/PlatformIdentityAdmin"
          ]
        }
      }
    },
    {
      "Sid": "KeepWorkloadBoundaryOnIdentities",
      "Effect": "Deny",
      "Resource": "*",
      "Action": [
        "iam:DeleteRolePermissionsBoundary",
        "iam:DeleteUserPermissionsBoundary"
      ],
      "Condition": {
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::*:role/PlatformAdmin",
            "arn:aws:iam::*:role/AWSControlTowerExecution",
            "arn:aws:iam::*:role/stacksets-exec-*",
            "arn:aws:iam::*:role/PlatformIdentityAdmin"
          ]
        }
      }
    }
  ]
}
```

Publish the boundary, attach it, and prove the result (AWS CLI)

```bash
aws cloudformation create-stack-set --stack-set-name sf-workload-boundary --template-body
file://sf-workload-boundary.yaml \
  --permission-model SERVICE_MANAGED --auto-deployment
Enabled=true,RetainStacksOnAccountRemoval=true \
  --capabilities CAPABILITY_NAMED_IAM
aws cloudformation create-stack-instances --stack-set-name sf-workload-boundary \
  --deployment-targets OrganizationalUnitIds=<root-organizational-unit-id> --regions us-east-1
# Service-managed stack sets skip the management account: deploy the same template there as a
stack.
aws cloudformation create-stack --profile <management-profile> --stack-name sf-workload-
boundary \
  --template-body file://sf-workload-boundary.yaml --capabilities CAPABILITY_NAMED_IAM

# IAM roles and users:
aws iam put-role-permissions-boundary --role-name <flagged-role-name> \
  --permissions-boundary arn:aws:iam::<account-id>:policy/sf-workload-boundary
```

```
# Identity Center permission sets (FleetDeveloper, FleetOnCall, DataAnalyst); their
AWSReservedSSO_ roles reject the call above.
INSTANCE_ARN=$(aws sso-admin list-instances --query 'Instances[0].InstanceArn' --output text)
for ps in <fleet-developer-permission-set-arn> <fleet-on-call-permission-set-arn> <data-
analyst-permission-set-arn>; do
  aws sso-admin put-permissions-boundary-to-permission-set --instance-arn "$INSTANCE_ARN" --
permission-set-arn "$ps" \
    --permissions-boundary CustomerManagedPolicyReference={Name=sf-workload-boundary,Path=/}
  aws sso-admin provision-permission-set --instance-arn "$INSTANCE_ARN" --permission-set-arn
"$ps" --target-type ALL_PROVISIONED_ACCOUNTS
done

# Non-production proof, one role of each kind: escalation denied, service actions allowed.
aws iam simulate-principal-policy --policy-source-arn arn:aws:iam::<non-production-account-
id>:role/<flagged-role-name> \
  --action-names iam:CreatePolicyVersion iam:SetDefaultPolicyVersion iam:PutUserPolicy
iam:PutRolePolicy \
                 iam:AttachRolePolicy iam:UpdateAssumeRolePolicy iam:PassRole
lambda:CreateFunction ec2:RunInstances \
  --query 'EvaluationResults[].[EvalActionName,EvalDecision]' --output table
aws iam simulate-principal-policy --policy-source-arn arn:aws:iam::<non-production-account-
id>:role/<flagged-role-name> \
  --action-names ssm:SendCommand iot:Publish sagemaker:CreateTrainingJob kinesis:PutRecord
ec2:AssignPrivateIpAddresses \
                 eks-auth:AssumeRoleForPodIdentity \
  --query 'EvaluationResults[].[EvalActionName,EvalDecision]' --output table
```

How it is verified

- `simulate-principal-policy` returns `implicitDeny` or `explicitDeny` for every escalation action on each flagged role, and `allowed` for the service actions in the second simulation.
- `aws iam create-role` without the boundary is denied by `sf-require-workload-boundary`, and with the boundary it succeeds for the platform workflow.
- `aws iam delete-role-permissions-boundary` and `aws iam create-policy-version` on `sf-workload-boundary` are denied for an ordinary administrator and succeed for the stack set's execution role.
- The next scan reports no role with PassRole plus lambda:CreateFunction or ec2:RunInstances on a wildcard resource.

How to roll it back. Remove the boundary from the affected role or user with `aws iam delete-role-permissions-boundary` or `delete-user-permissions-boundary`, run as a platform role, and from an Identity Center permission set with `aws sso-admin delete-permissions-boundary-from-permission-set` followed by `aws sso-admin provision-permission-set`. If new roles cannot be created, detach `sf-require-workload-boundary` from the OU that holds the account. Revert the stack set that publishes the boundary last, and the stack in sf-management with it. Restoring the old PassRole statements reopens the escalation paths, so rebuild the exact action the workload needs instead. Verify by running `simulate-principal-policy` for the escalation actions.

SF-SEC-019: Kestrel's former management account still administers four Kestrel member accounts

High. Benchmark rating High (Vantalect baseline: cross-account admin trust). How it was rated: benchmark High (3) +0 exploitability (moderate) +0 blast radius (production) = 3

Where: `kestrel-prod` (…9031), `kestrel-staging` (…9054), `kestrel-dev` (…9076), `kestrel-data` (…9098) in us-east-1, 4 IAM roles. **Controls:** No benchmark control covers this. **SOC 2:** CC6.2, CC3.4. **Owner:** Jonah Reyes, Director, Perception Platform (formerly Kestrel Vision). **Repository:** `platform-landing-zone`. **Effort:** 6 hours. **Planned:** Week 4, starting Aug 3. **Depends on:** SF-SEC-008.

Evidence

- Each role trusts arn:aws:iam::…9120:root, Kestrel's management account before November 2025.
- StrataForge's own management account has no role in the Kestrel accounts.

Examples: `kestrel-prod (…9031): OrganizationAccountAccessRole`, `kestrel-staging (…9054): OrganizationAccountAccessRole`, `kestrel-dev (…9076): OrganizationAccountAccessRole`, `kestrel-data (…9098): OrganizationAccountAccessRole`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Any of kestrel-legacy-mgmt's 40 CI keys can grant itself admin there (path PE-05) and then assume the roles. **Blast radius.** Production systems or customer data. kestrel-prod and kestrel-data.

How we verified it. Read the four trust policies with Jonah Reyes on 07/20/2026.

Remediation: Point the Kestrel OrganizationAccountAccessRole trust at the StrataForge management account. Four Kestrel member accounts still let Kestrel's former management account assume `OrganizationAccountAccessRole`. That account is now a StrataForge member running CI, so replace the trust with the real management account.

Steps

1. List who in `kestrel-legacy-mgmt` still assumes the role, from CloudTrail in each Kestrel account, and move that CI to OIDC roles.
2. Open each Kestrel account through an IAM Identity Center permission set, not through the role you are about to change.
3. Replace the role's trust policy so only the management account root can assume it, and require MFA on the session.

4. Deploy the EventBridge rule so any assumption from another account pages the security team.
5. Enroll the Kestrel accounts in the landing zone (see control-tower-enrollment) so the role is managed from then on.

OrganizationAccountAccessRole trusting the StrataForge management account (IAM trust policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<management-account-id>:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "Bool": {
          "aws:MultiFactorAuthPresent": "true"
        }
      }
    }
  ]
}
```

Update the trust in each Kestrel member account (AWS CLI)

```BASH
set -euo pipefail
for profile in kestrel-prod-admin kestrel-staging-admin kestrel-dev-admin kestrel-data-admin;
do
  # Save the current trust policy first; nothing is replaced if the save fails.
  aws iam get-role --profile "$profile" --role-name OrganizationAccountAccessRole --query
Role.AssumeRolePolicyDocument --output json > "saved-trust-$profile.json"
  test -s "saved-trust-$profile.json"
  aws iam update-assume-role-policy --profile "$profile" --role-name
OrganizationAccountAccessRole \
    --policy-document file://org-access-role-trust.json
  aws iam get-role --profile "$profile" --role-name OrganizationAccountAccessRole \
    --query Role.AssumeRolePolicyDocument.Statement[0].Principal
done
```

Page when anyone outside the management account assumes the role (EventBridge event pattern)

```JSON
{
  "source": [
```

```
    "aws.sts"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "sts.amazonaws.com"
    ],
    "eventName": [
      "AssumeRole"
    ],
    "requestParameters": {
      "roleArn": [
        {
          "suffix": ":role/OrganizationAccountAccessRole"
        }
      ]
    },
    "userIdentity": {
      "accountId": [
        {
          "anything-but": [
            "<management-account-id>"
          ]
        }
      ]
    }
  }
}
```

How it is verified

- The four roles show `arn:aws:iam::<management-account-id>:root` and no `<kestrel-legacy-mgmt-account-id>` in their trust policies.
- An assume-role attempt from `kestrel-legacy-mgmt` returns AccessDenied.
- The Kestrel CI jobs that ran under the legacy trust now assume OIDC roles and still pass.

How to roll it back. The change saves each role's current trust policy as `saved-trust-<profile>.json` before it replaces it. To roll back, put that document back in each Kestrel account with `aws iam update-assume-role-policy --profile <profile> --role-name OrganizationAccountAccessRole --policy-document file://saved-trust-<profile>.json`, then remove the EventBridge rule. Do this only if the Kestrel CI that assumed the role has not yet moved to OIDC roles, and move it within the week. Verify by running that CI job once and reading the trust policy back.

SF-SEC-020: A former contractor's AWS account holds AdministratorAccess in mfg-mes-prod with no ExternalId

High. Benchmark rating High (IAM Access Analyzer external access). How it was rated: benchmark High (3) +0 exploitability (moderate) +0 blast radius (production) = 3

Where: `mfg-mes-prod` (…8084) in us-east-1, 1 IAM role. **Controls:** No Security Hub CSPM control covers this. Found by: IAM Access Analyzer external access. **SOC 2:** CC6.2, CC9.2. **Owner:** Mei Chen, Director, Enterprise Applications. **Repository:** `enterprise-infra`. **Effort:** 4 hours. **Planned:** Week 4, starting Aug 3.

Evidence

- Trust: arn:aws:iam::`<contractor-account-id>`:root with no conditions.
- Contract ended 10/31/2024; role last used 11/19/2024.

Examples: `arn:aws:iam::…8084:role/kettleworth-mes-integration`, `arn:aws:iam::aws:policy/AdministratorAccess`, `trusted principal arn:aws:iam::<contractor-account-id>:root`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Anyone with admin in the contractor's account can assume the role; its security is unknown. **Blast radius.** Production systems or customer data. Manufacturing execution for all 11 factories.

How we verified it. Mei Chen confirmed on 07/21/2026 that no contract with the integrator is active.

Remediation: Delete the Kettleworth contractor role and use a scoped, ExternalId-protected pattern for vendors. `kettleworth-mes-integration` in mfg-mes-prod gives a contractor account AdministratorAccess with no ExternalId, and the contract ended in 2024. Delete the role, and make every future vendor role narrow, short-lived, and bound to an ExternalId.

Steps

1. Check the role's last-used date and its CloudTrail activity for the last 90 days; record any real use and who owns it.
2. Detach AdministratorAccess and any inline policies, remove any instance profiles, and delete the role.
3. Ask Legal to confirm the contract end and tell the contractor that access is closed.
4. For any future vendor, create the role in CDK from the pattern below: a named vendor principal, an ExternalId, a one-hour session, and a policy listing only the actions the vendor needs.

5. Add each vendor role to the quarterly access review, with an owner and an expiry date in its tags.

Prove the role is idle, then remove it (AWS CLI)

```bash
set -euo pipefail
ROLE=kettleworth-mes-integration
aws iam get-role --role-name "$ROLE" --query 'Role.
[RoleLastUsed.LastUsedDate,RoleLastUsed.Region]'
# Save everything first; with set -e nothing is detached or deleted if a save fails.
mkdir -p retired-kettleworth
aws iam get-role --role-name "$ROLE" --output json > retired-kettleworth/role.json
aws iam list-attached-role-policies --role-name "$ROLE" --output json > retired-kettleworth/
attached.json
for name in $(aws iam list-role-policies --role-name "$ROLE" --query 'PolicyNames[]' --output
text); do
  aws iam get-role-policy --role-name "$ROLE" --policy-name "$name" --output json > "retired-
kettleworth/inline-$name.json"
done
test -s retired-kettleworth/role.json
for arn in $(aws iam list-attached-role-policies --role-name "$ROLE" --query
'AttachedPolicies[].PolicyArn' --output text); do
  aws iam detach-role-policy --role-name "$ROLE" --policy-arn "$arn"
done
for name in $(aws iam list-role-policies --role-name "$ROLE" --query 'PolicyNames[]' --output
text); do
  aws iam delete-role-policy --role-name "$ROLE" --policy-name "$name"
done
aws iam delete-role --role-name "$ROLE"
```

Vendor role pattern: ExternalId, scoped policy, one-hour sessions (AWS CDK, TypeScript)

```typescript
import { Duration, Stack, StackProps, Tags, aws_iam as iam } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class VendorAccessStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const role = new iam.Role(this, 'VendorRole', {
      roleName: 'vendor-<vendor-name>-integration',
      assumedBy: new iam.AccountPrincipal('<vendor-account-id>'),
      externalIds: ['<vendor-external-id>'],
      maxSessionDuration: Duration.hours(1),
    });
    role.addToPolicy(new iam.PolicyStatement({
      actions: ['s3:GetObject', 's3:PutObject'],
      resources: ['arn:aws:s3:::<exchange-bucket-name>/<vendor-name>/*'],
    }));
    Tags.of(role).add('owner', '<owning-team>');
    Tags.of(role).add('expires', '<expiry-date>');
  }
}
```

Resulting trust policy shape for a vendor role (IAM trust policy)

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<vendor-account-id>:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "<vendor-external-id>"
        }
      }
    }
  ]
}
```

How it is verified

- `aws iam get-role --role-name kettleworth-mes-integration` returns NoSuchEntity in mfg-mes-prod.
- IAM Access Analyzer shows no external-access finding for a principal outside the organization in mfg-mes-prod.

- Each vendor role has an ExternalId condition, a `MaxSessionDuration` of 3600, and no AdministratorAccess.

How to roll it back. A deleted role cannot be restored. The change saves its trust policy, attached policies, and inline policies to `retired-kettleworth/` before anything is detached, with `aws iam get-role --role-name kettleworth-mes-integration`, `aws iam list-attached-role-policies --role-name kettleworth-mes-integration`, and `aws iam get-role-policy` for each inline policy; that copy is the only way back, and the deletion does not run if any of those saves fails. Recreate it only with Legal's written approval, with an ExternalId, the narrow policy from the pattern, and an expiry tag, never with AdministratorAccess. Verify with a test assumption from the vendor account and a CloudTrail check that the session has the expected scope.

SF-SEC-021: The staging OTA test key lets the production pipeline sign, and pilot robots still trust it

High. Benchmark rating High (Vantalect baseline: a signing key shared across environments). How it was rated: benchmark High (3) +0 exploitability (moderate) +0 blast radius (production) = 3

Where: `fleet-ota-prod` (…5117), `fleet-ota-staging` (…6148) in us-east-1, 1 KMS key. **Controls:** No Security Hub CSPM control covers this. Found by: Vantalect OTA chain-of-trust review. **SOC 2:** CC6.1, CC8.1. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 8 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- Key policy statement "AllowReleasePipeline" grants kms:Sign to the production role ota-release-pipeline.
- Firmware 4.1 on 1,240 Sentinel pilot units keeps the test public key in slot 2.

Examples: `alias/ota-test-signing`, `arn:aws:iam::…5117:role/ota-release-pipeline`, `Sentinel firmware 4.1 key slot 2`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). A compromise of the production pipeline role can produce test-signed images and publish them. **Blast radius.** Production systems or customer data. 1,240 Sentinel pilot robots still accept test-signed images.

How we verified it. Read the key policy and the Sentinel bootloader manifest with the fleet team on 07/22/2026.

Remediation: Limit the staging test signing key to the staging pipeline role and remove its public key from the pilot robots. The key policy on `alias/ota-test-signing` lets the production role `ota-release-pipeline` sign with a key that staging treats as disposable, and the 1,240 Sentinel

pilot robots on firmware 4.1 still keep its public key in bootloader slot 2, so they accept test-signed images. Scope the key to the staging pipeline role, then remove the test key from the pilot robots with a job signed by the production key.

Steps

1. Read the CloudTrail history for `Sign` on the test key and list every principal that used it, so nothing legitimate breaks.
2. Look up the key ID with `describe-key` (the key-policy calls take a key ID or key ARN, never an alias) and save the current key policy to the change record.
3. Write the new key policy: account administration, Sign for the staging pipeline role only, DescribeKey and GetPublicKey in their own statement (those two calls carry no signing-algorithm condition), and a deny of Sign for every principal except the staging pipeline role.
4. Apply it with `put-key-policy` and run one staging pipeline build to prove signing still works.
5. Create the EventBridge rule in the key's Region with state `ENABLED_WITH_ALL_CLOUDTRAIL_MANAGEMENT_EVENTS` (Sign is a read-only management event) so any later Sign by another principal pages the platform team.
6. Close the pilot-robot gap: have the firmware team write a job document that clears the test public key from bootloader slot 2, sign the job with the production key, put the 1,240 Sentinel pilot units in the thing group `sf-sentinel-pilot`, and roll the job out in waves with an abort rule. Production robots must already trust only the production public key.
7. Confirm from the job's execution summary and the pilots' next reported key manifest that no unit still lists the test key, then schedule the test key's deletion only after that.

Key policy for alias/ota-test-signing (KMS key policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnableIamAdministration",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<fleet-ota-staging-account-id>:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "StagingPipelineMaySign",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<fleet-ota-staging-account-id>:role/ota-staging-pipeline"
      },
      "Action": "kms:Sign",
```

```
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:SigningAlgorithm": "ECDSA_SHA_384"
        }
      }
    },
    {
      "Sid": "StagingPipelineMayDescribeTheKey",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<fleet-ota-staging-account-id>:role/ota-staging-pipeline"
      },
      "Action": [
        "kms:DescribeKey",
        "kms:GetPublicKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "NobodyElseMaySign",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "kms:Sign",
      "Resource": "*",
      "Condition": {
        "ArnNotEquals": {
          "aws:PrincipalArn": "arn:aws:iam::<fleet-ota-staging-account-id>:role/ota-staging-
pipeline"
        }
      }
    }
  ]
}
```

Review who has used the key, save and apply the policy, and read it back (AWS CLI)

```bash
set -euo pipefail
aws cloudtrail lookup-events --profile <fleet-ota-staging-profile> --region us-east-1 \
  --lookup-attributes AttributeKey=EventName,AttributeValue=Sign --max-results 50 \
  --query 'Events[].CloudTrailEvent' --output text | grep -o '"arn":"[^"]*"' | sort | uniq -c
KEY_ID=$(aws kms describe-key --profile <fleet-ota-staging-profile> --region us-east-1 --key-id
alias/ota-test-signing \
  --query KeyMetadata.KeyId --output text)
# Save the current policy first; the rollback restores exactly this file.
aws kms get-key-policy --profile <fleet-ota-staging-profile> --region us-east-1 --key-id
"$KEY_ID" --policy-name default \
  --query Policy --output text > ota-test-key-policy.before.json
test -s ota-test-key-policy.before.json
aws kms put-key-policy --profile <fleet-ota-staging-profile> --region us-east-1 \
  --key-id "$KEY_ID" --policy-name default --policy file://ota-test-key-policy.json
aws kms get-key-policy --profile <fleet-ota-staging-profile> --region us-east-1 \
  --key-id "$KEY_ID" --policy-name default --query Policy --output text
# Sign is a read-only management event: only this rule state delivers it. The SNS topic must
allow events.amazonaws.com to publish.
aws events put-rule --profile <fleet-ota-staging-profile> --region us-east-1 --name sf-ota-
test-unexpected-sign \
  --state ENABLED_WITH_ALL_CLOUDTRAIL_MANAGEMENT_EVENTS --event-pattern file://ota-test-
unexpected-sign-pattern.json
aws events put-targets --profile <fleet-ota-staging-profile> --region us-east-1 --rule sf-ota-
test-unexpected-sign \
  --targets Id=page-platform,Arn=<platform-alarm-topic-arn>
```

ota-test-unexpected-sign-pattern.json: page when anything other than the staging pipeline role signs (EventBridge event pattern)

```JSON
{
  "source": [
    "aws.kms"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "kms.amazonaws.com"
    ],
    "eventName": [
      "Sign"
    ],
    "userIdentity": {
      "arn": [
        {
          "anything-but": {
            "prefix": "arn:aws:sts::<fleet-ota-staging-account-id>:assumed-role/ota-staging-
pipeline/"
          }
        }
      ]
    }
  }
}
```

Remove the test public key from the 1,240 pilot robots with a job signed by the production key (AWS CLI)

```BASH
set -euo pipefail
# sentinel-slot2-job.json is the firmware team's job document; it is signed with the production
key before this call.
GROUP_ARN=$(aws iot describe-thing-group --region us-east-1 --thing-group-name sf-sentinel-
pilot --query thingGroupArn --output text)
aws iot create-job --region us-east-1 --job-id sentinel-clear-test-key-slot-2 --targets
"$GROUP_ARN" \
  --document file://sentinel-slot2-job.json --target-selection SNAPSHOT \
  --job-executions-rollout-config '{"exponentialRate": {"baseRatePerMinute": 20,
"incrementFactor": 2, "rateIncreaseCriteria": {"numberOfSucceededThings": 50}},
"maximumPerMinute": 200}' \
  --abort-config '{"criteriaList": [{"failureType": "FAILED", "action": "CANCEL",
"thresholdPercentage": 5, "minNumberOfExecutedThings": 20}]}' \
  --timeout-config inProgressTimeoutInMinutes=60
aws iot describe-job --region us-east-1 --job-id sentinel-clear-test-key-slot-2 --query
'job.jobProcessDetails'
```

How it is verified

- `aws kms get-key-policy` shows no principal from fleet-ota-prod.

- A `kms:Sign` call with the production `ota-release-pipeline` role against the test key returns AccessDeniedException, and the EventBridge rule's `State` is `ENABLED_WITH_ALL_CLOUDTRAIL_MANAGEMENT_EVENTS`.
- The next staging pipeline run signs successfully as `ota-staging-pipeline`.
- The job `sentinel-clear-test-key-slot-2` shows every one of the 1,240 pilot units SUCCEEDED, and an image signed with the test key is refused by a pilot unit on the bench.

How to roll it back. Before step 4, the current key policy is saved as `ota-test-key-policy.before.json`. To roll back, restore it with `aws kms put-key-policy --key-id <key-id> --policy-name default --policy file://ota-test-key-policy.before.json` (the key ID from `describe-key`, never the alias), then remove the EventBridge rule with `aws events remove-targets` and `aws events delete-rule`. That policy lets the production pipeline role sign with the test key, so roll back only to unblock a staging release and restore the new policy the same day. Stop the pilot-robot job with `aws iot cancel-job --job-id sentinel-clear-test-key-slot-2`; units that already ran it no longer trust the test key and can get it back only through a new signed job, and the test key must not be scheduled for deletion until every pilot unit reports the new manifest. Verify by running one staging build and reading the policy back.

SF-SEC-022: AWS Config records nothing in 570 account-Regions

High. Benchmark rating Critical (Security Hub CSPM Config.1). How it was rated: benchmark Critical (4) -1 exploitability (limited) +1 blast radius (organization) = 4; rule 2: Critical needs a remote path and a production-or-wider blast radius; capped at High

Where: All 48 accounts in 17 Regions, 570 account-Regions. **Controls:** `Config.1` (AWS Config should be enabled and use the service-linked role for resource recording). **SOC 2:** CC2.1, CC4.1, CC7.1. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 10 hours. **Planned:** Week 5, starting Aug 10. **Depends on:** SF-SEC-039.

Evidence

- Control Tower governs only the six working Regions; unenrolled accounts have no recorder.

Examples: `sf-management us-east-2`, `sf-management us-west-1`, `sf-management ca-central-1`.

Exploitability. Needs a foothold inside, an insider, or physical access. No configuration history to investigate or prove a change. **Blast radius.** Every account in the organization. All 11 unused Regions and the seven unenrolled accounts.

How we verified it. Read from the scan; Control Tower role failures removed as false positives.

Remediation: Record AWS Config in all 17 Regions and every account, including the Kestrel accounts. AWS Config records in only six Regions and not in the Kestrel accounts (Config.1). Extend the Control Tower landing zone's governed Regions to all 17, enroll the accounts that Control Tower does not manage yet, and aggregate everything in sf-security-tooling.

Steps

1. Register sf-security-tooling as the Config delegated administrator.
2. Update the landing zone manifest so `governedRegions` lists all 17 Regions; Control Tower then deploys its Config recorder and rules to them.
3. Enroll the unenrolled accounts (see control-tower-enrollment) so Control Tower records them too.
4. Create the organization configuration aggregator for all Regions in sf-security-tooling.
5. Remember that Control Tower's recorder uses its own role, which is why Config.1 is set to skip the service-linked role check in Security Hub CSPM (see securityhub-org).

Delegated administrator and the landing zone Region change (management account) (AWS CLI)

```bash
aws organizations enable-aws-service-access --service-principal config.amazonaws.com
aws organizations register-delegated-administrator --service-principal config.amazonaws.com --
account-id <security-tooling-account-id>

LZ=$(aws controltower list-landing-zones --query 'landingZones[0].arn' --output text)
aws controltower get-landing-zone --landing-zone-identifier "$LZ" --query
'landingZone.manifest' > lz-manifest.json
# Edit lz-manifest.json so governedRegions lists these 17 Regions:
#   us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-west-1 eu-
west-2
#   eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-southeast-1 ap-
southeast-2 ap-south-1
aws controltower update-landing-zone --landing-zone-identifier "$LZ" --version <landing-zone-
version> --manifest file://lz-manifest.json
aws controltower get-landing-zone-operation --operation-identifier <operation-identifier> --
query 'operationDetails.status'
```

Organization aggregator in sf-security-tooling (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Aggregate AWS Config data from every account and Region.
Resources:
  AggregatorRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: sf-config-aggregator
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AWSConfigRoleForOrganizations
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal: {Service: config.amazonaws.com}
            Action: 'sts:AssumeRole'
  Aggregator:
    Type: AWS::Config::ConfigurationAggregator
    Properties:
      ConfigurationAggregatorName: sf-org-aggregator
      OrganizationAggregationSource:
        RoleArn: !GetAtt AggregatorRole.Arn
        AllAwsRegions: true
```

How it is verified

- `aws configservice describe-configuration-recorder-status` shows a recording recorder in all 17 Regions of a sample account, including a Kestrel account (Security Hub CSPM Config.1).
- `aws configservice describe-configuration-aggregator-sources-status` shows every account and Region as succeeded.
- `aws controltower get-landing-zone` shows `governedRegions` with all 17 Regions and status ACTIVE.

How to roll it back. Revert the landing zone manifest to its previous `governedRegions` list and update the landing zone, which stops Control Tower from recording in the added Regions. Delete the aggregator by reverting its stack. Recordings already delivered stay in the log archive. Verify with the landing zone status.

SF-SEC-023: GuardDuty is off in 558 account-Regions, including every Kestrel account; most protection plans are off

High. Benchmark rating High (Security Hub CSPM GuardDuty.1 and GuardDuty.5–13). How it was rated: benchmark High (3) -1 exploitability (limited) +1 blast radius (organization) = 3

Where: All 48 accounts in 17 Regions, 558 account-Regions. **Controls:** `GuardDuty.1` (GuardDuty should be enabled); `GuardDuty.5` (GuardDuty EKS Audit Log Monitoring should be enabled); `GuardDuty.6` (GuardDuty Lambda Protection should be enabled); `GuardDuty.7` (GuardDuty EKS

Runtime Monitoring should be enabled); `GuardDuty.8` (GuardDuty Malware Protection for EC2 should be enabled); `GuardDuty.9` (GuardDuty RDS Protection should be enabled); `GuardDuty.11` (GuardDuty Runtime Monitoring should be enabled); `GuardDuty.12` (GuardDuty ECS Runtime Monitoring should be enabled); `GuardDuty.13` (GuardDuty EC2 Runtime Monitoring should be enabled). **SOC 2:** CC6.8, CC7.2, CC7.3. **Owner:** Nadia Petrova, Manager, Security Operations. **Repository:** `platform-landing-zone`. **Effort:** 12 hours. **Planned:** Week 5, starting Aug 10.

Evidence

- Detectors in the 6 working Regions of 43 accounts; auto-enable for new members is off.
- 1,872 protection-plan checks fail where GuardDuty runs (Runtime Monitoring, RDS, Lambda, and Malware Protection off).

Examples: `sf-management us-east-2`, `sf-management us-west-1`, `sf-management ca-central-1`.

Exploitability. Needs a foothold inside, an insider, or physical access. A detection gap: an attacker in an unmonitored Region or account is not seen. **Blast radius.** Every account in the organization. All 11 unused Regions and the five Kestrel accounts.

How we verified it. Read from the scan.

Remediation: Run GuardDuty in all 17 Regions for every account with every protection plan on. GuardDuty runs in only six Regions, not in the Kestrel accounts, and with partial protection plans (GuardDuty.1, GuardDuty.5 to GuardDuty.13). Make sf-security-tooling the delegated administrator in all 17 Regions and auto-enable it, with every protection plan, for all members.

Steps

1. From the management account, enable trusted access and make sf-security-tooling the GuardDuty delegated administrator in each of the 17 Regions.
2. In sf-security-tooling, set the organization configuration to auto-enable `ALL` members and every protection plan the API exposes: S3 Protection, EKS Protection, Runtime Monitoring, Malware Protection for EC2, RDS Protection, and Lambda Protection.
3. Turn on Malware Protection for S3 on the sensitive buckets, Malware Protection for AWS Backup on the backup vaults, and AI Protection where Bedrock or SageMaker AI is used; these are set per resource or plan.
4. Confirm the five Kestrel accounts show as members in every Region.
5. Route severity 7 and above findings to the security on-call from every Region (the routing script forwards each Region to us-east-1), and check that Extended Threat Detection findings arrive, since it is on by default.

Where GuardDuty stands today, per Region (read-only, sf-security-tooling) (AWS CLI)

```bash
for region in us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-west-1 eu-west-2 eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-southeast-1 ap-southeast-2 ap-south-1; do
  echo "$region detector: $(aws guardduty list-detectors --profile <security-tooling-profile> --region $region --query 'DetectorIds[0]' --output text)"
done
```

Delegate in every Region, then enable all members and plans (management account, then sf-security-tooling) (AWS CLI)

```bash
REGIONS="us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-west-1 eu-west-2 eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-southeast-1 ap-southeast-2 ap-south-1"

# 1. Management account: trusted access, the list of active accounts, and the delegated administrator in EVERY Region.
#    The designation is per Region; a Region without it rejects update-organization-configuration.
aws organizations enable-aws-service-access --profile <management-profile> --service-principal guardduty.amazonaws.com
aws organizations list-accounts --profile <management-profile> --query 'Accounts[?State==`ACTIVE`].Id' --output text | tr '\t' '\n' > active-accounts.txt
for region in $REGIONS; do
  aws guardduty enable-organization-admin-account --profile <management-profile> --region $region --admin-account-id <security-tooling-account-id>
done

# 2. sf-security-tooling: a detector in each Region, auto-enable for all members, and a count of members that came up.
#    The administrator account is not its own member, so the expected count is one less than the active accounts.
expected=$(( $(wc -l < active-accounts.txt) - 1 ))
for region in $REGIONS; do
  detector=$(aws guardduty list-detectors --profile <security-tooling-profile> --region $region --query 'DetectorIds[0]' --output text)
  [ "$detector" = "None" ] && detector=$(aws guardduty create-detector --profile <security-tooling-profile> --region $region --enable --query DetectorId --output text)
  aws guardduty update-organization-configuration --profile <security-tooling-profile> --region $region --detector-id "$detector" --auto-enable-organization-members ALL --features '[
    {"Name": "S3_DATA_EVENTS", "AutoEnable": "ALL"},
    {"Name": "EKS_AUDIT_LOGS", "AutoEnable": "ALL"},
    {"Name": "EBS_MALWARE_PROTECTION", "AutoEnable": "ALL"},
    {"Name": "RDS_LOGIN_EVENTS", "AutoEnable": "ALL"},
    {"Name": "LAMBDA_NETWORK_LOGS", "AutoEnable": "ALL"},
    {"Name": "RUNTIME_MONITORING", "AutoEnable": "ALL", "AdditionalConfiguration": [
      {"Name": "EKS_ADDON_MANAGEMENT", "AutoEnable": "ALL"},
      {"Name": "ECS_FARGATE_AGENT_MANAGEMENT", "AutoEnable": "ALL"},
      {"Name": "EC2_AGENT_MANAGEMENT", "AutoEnable": "ALL"}]}]'
  enabled=$(aws guardduty list-members --profile <security-tooling-profile> --region $region --detector-id "$detector" --only-associated false \
    --query 'length(Members[?RelationshipStatus==`Enabled`])' --output text)
  [ "$enabled" = "$expected" ] || echo "CHECK $region: $enabled of $expected members are
```

```
Enabled (enrollment can take several minutes; run the count again)"
done
```

High and critical GuardDuty findings (EventBridge event pattern)

```JSON
{
  "source": [
    "aws.guardduty"
  ],
  "detail-type": [
    "GuardDuty Finding"
  ],
  "detail": {
    "severity": [
      {
        "numeric": [
          ">=",
          7
        ]
      }
    ]
  }
}
```

Route GuardDuty findings from every Region to the on-call (sf-security-tooling) (AWS CLI)

```bash
REGIONS="us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-west-1
eu-west-2 eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-southeast-1 ap-
southeast-2 ap-south-1"
BUS=arn:aws:events:us-east-1:<security-tooling-account-id>:event-bus/default
FORWARD_ROLE=arn:aws:iam::<security-tooling-account-id>:role/sf-events-forward

# Once: the role EventBridge uses to put events on the us-east-1 bus.
aws iam create-role --role-name sf-events-forward --assume-role-policy-document \
  '{"Version": "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
"events.amazonaws.com"}, "Action": "sts:AssumeRole"}]}'
aws iam put-role-policy --role-name sf-events-forward --policy-name put-events-to-us-east-1 --
policy-document \
  '{"Version": "2012-10-17", "Statement": [{"Effect": "Allow", "Action": "events:PutEvents",
"Resource": "'$BUS'"}]}'

# Every Region except us-east-1 forwards matching findings; us-east-1 pages directly.
for region in $REGIONS; do
  aws events put-rule --region $region --name sf-guardduty-high-critical --state ENABLED --
event-pattern file://guardduty-high-critical-pattern.json
  if [ "$region" = "us-east-1" ]; then
    aws sns create-topic --region us-east-1 --name sf-guardduty-pagerduty
    aws events put-targets --region us-east-1 --rule sf-guardduty-high-critical \
      --targets Id=PagerDuty,Arn=arn:aws:sns:us-east-1:<security-tooling-account-id>:sf-
guardduty-pagerduty
  else
    aws events put-targets --region $region --rule sf-guardduty-high-critical --targets
Id=ToUsEast1,Arn=$BUS,RoleArn=$FORWARD_ROLE
  fi
done
# The us-east-1 rule matches both its own findings and the forwarded ones, because forwarding
keeps the source and detail.
for region in $REGIONS; do
  echo "$region: $(aws events describe-rule --region $region --name sf-guardduty-high-critical
--query State --output text)"
done
```

How it is verified

- `aws guardduty list-detectors` returns a detector in all 17 Regions, `list-organization-admin-accounts` names sf-security-tooling in each, and `list-members` shows every member account as Enabled, which is every active account except the administrator itself (Security Hub CSPM GuardDuty.1).
- `aws guardduty describe-organization-configuration` shows `AutoEnableOrganizationMembers` ALL and every listed feature on ALL.
- A sample finding generated with `create-sample-findings` in a Region other than us-east-1 reaches the on-call route.

How to roll it back. Turn off a protection plan in the organization configuration with `aws guardduty update-organization-configuration`, or suspend a member in one Region. Rolling back reduces detection, so use it for a cost or noise incident only, and keep the detector in the Region running. Delegating administration to another account is a separate change. Verify by reading the organization configuration.

SF-SEC-024: S3 Block Public Access is off at the account level in 31 accounts and on 47 buckets

High. Benchmark rating High (Security Hub CSPM S3.1 and S3.8). How it was rated: benchmark High (3) -1 exploitability (limited) +1 blast radius (organization) = 3

Where: 37 accounts in 4 Regions, 31 accounts. **Controls:** `S3.1` (S3 general purpose buckets should have block public access settings enabled); `S3.8` (S3 general purpose buckets should block public access). **SOC 2:** CC5.2, CC6.6, C1.1. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 10 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- S3.1 failed in 31 accounts; S3.8 failed on 47 buckets.

Examples: `fleet-analytics-prod (…5093)`, `fleet-ota-staging (…6148)`, `fleet-remote-ops-prod (…5139)`, `arn:aws:s3:::sf-artifactory-artifacts-1188-us-east-1-001`, `arn:aws:s3:::sf-build-cache-build-cache-1356-us-east-1-000`.

Exploitability. Needs a foothold inside, an insider, or physical access. Nothing is public today beyond the Kestrel buckets, but one policy edit would make any bucket public. **Blast radius.** Every account in the organization. Every account without the account-level setting.

How we verified it. Read from the scan.

Remediation: Turn on account-level S3 Block Public Access in every account and lock it. Account-level Block Public Access is off in many accounts, so any bucket can be made public by a policy or ACL (S3.1, S3.8). Set it with `s3control put-public-access-block` in every account, report drift with a Config rule, and deny anyone but the platform role from changing it.

Steps

1. List every active account and run the s3control command in each one; account-level settings apply to all buckets, current and future. Then set each bucket's own block, which is what S3.8 reads, except the buckets that serve public content on purpose.
2. Check for buckets that serve public content on purpose (static websites) before you run it; move them behind CloudFront with origin access control.
3. Deploy the Config rule as a stack set so a later change shows up as non-compliant.

4. Attach `sf-protect-account-defaults` (see scp-guardrails); its KeepS3AccountBlock statement lets only the platform roles change or delete the account setting.
5. Consider the Organizations Amazon S3 policy type as a later step: it sets all four settings across accounts and overrides the account-level values, and detaching it reverts accounts to their own settings.

Set the account-level block and every bucket's own block in every account (AWS CLI)

```bash
# Active accounts come from the State field; the older Status field is deprecated.
# intentionally-public-buckets.txt lists, one name per line, the buckets that serve public
content on purpose.
BLOCK=BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets=t
rue
for account in $(aws organizations list-accounts --query 'Accounts[?State==`ACTIVE`].Id' --
output text); do
  creds=$(aws sts assume-role --role-arn arn:aws:iam::$account:role/
OrganizationAccountAccessRole \
    --role-session-name s3-bpa --query Credentials --output json) || { echo "SKIP $account:
cannot assume the role"; continue; }
  (export AWS_ACCESS_KEY_ID=$(echo "$creds" | jq -r .AccessKeyId) AWS_SECRET_ACCESS_KEY=$(echo
"$creds" | jq -r .SecretAccessKey)
   export AWS_SESSION_TOKEN=$(echo "$creds" | jq -r .SessionToken)
   # Account level (S3.1).
   aws s3control put-public-access-block --account-id "$account" --public-access-block-
configuration $BLOCK
   aws s3control get-public-access-block --account-id "$account" --query
PublicAccessBlockConfiguration
   # Bucket level (S3.8 evaluates each bucket's own setting).
   for bucket in $(aws s3api list-buckets --query 'Buckets[].Name' --output text); do
     grep -qxF "$bucket" intentionally-public-buckets.txt 2>/dev/null && { echo "KEEP $bucket";
continue; }
     aws s3api put-public-access-block --bucket "$bucket" --public-access-block-configuration
$BLOCK \
       || echo "FAILED $account $bucket"
   done)
done
```

Config rule that reports accounts without the block (deploy as a stack set) (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Report accounts where S3 account-level Block Public Access is incomplete.
Resources:
  S3AccountBlockRule:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sf-s3-account-public-access-blocks
      InputParameters:
        BlockPublicAcls: 'True'
        IgnorePublicAcls: 'True'
        BlockPublicPolicy: 'True'
        RestrictPublicBuckets: 'True'
      Source:
        Owner: AWS
        SourceIdentifier: S3_ACCOUNT_LEVEL_PUBLIC_ACCESS_BLOCKS
```

sf-protect-account-defaults, statement KeepS3AccountBlock: only the platform roles may change the account-level block (the whole policy is in the guardrail SCPs) (Service control policy)

```json
{
  "Sid": "KeepS3AccountBlock",
  "Effect": "Deny",
  "Action": [
    "s3:PutAccountPublicAccessBlock"
  ],
  "Resource": "*",
  "Condition": {
    "ArnNotLike": {
      "aws:PrincipalArn": [
        "arn:aws:iam::*:role/PlatformAdmin",
        "arn:aws:iam::*:role/AWSControlTowerExecution",
        "arn:aws:iam::*:role/stacksets-exec-*"
      ]
    }
  }
}
```

How it is verified

- `aws s3control get-public-access-block --account-id <account-id>` shows four true values in every account, and `aws s3api get-public-access-block --bucket <bucket-name>` does for every bucket except the intentionally public ones.
- Security Hub CSPM S3.1 passes in every account and S3.8 passes for every bucket after the next evaluation.

- `aws s3control delete-public-access-block` from an ordinary administrator role is denied by `sf-protect-account-defaults`.

How to roll it back. Turn the account setting off for one account with `aws s3control delete-public-access-block`, after detaching the SCP from that account. For one bucket that must stay public, remove only that bucket's own block with `aws s3api delete-public-access-block --bucket <bucket-name>` so every other bucket keeps it, and move the bucket behind CloudFront with origin access control instead. Revert the Config rule stack set last. Verify with the Config rule's result.

SF-SEC-025: The 2019 organization trail can be stopped from the management account, and its logs can be deleted

High. Benchmark rating High (Vantalect baseline (log archive without Object Lock); Security Hub CSPM CloudTrail.2, .4, .7). How it was rated: benchmark High (3) -1 exploitability (limited) +1 blast radius (organization) = 3

Where: `sf-management` (…0007), `kestrel-prod` (…9031), `kestrel-data` (…9098), `kestrel-legacy-mgmt` (…9120) in us-east-1, 4 trails. **Controls:** `CloudTrail.2` (CloudTrail should have encryption at-rest enabled); `CloudTrail.4` (CloudTrail log file validation should be enabled); `CloudTrail.7` (Ensure S3 bucket access logging is enabled on the CloudTrail S3 bucket). **SOC 2:** CC4.1, CC7.2. **Owner:** Nadia Petrova, Manager, Security Operations. **Repository:** `platform-landing-zone`. **Effort:** 20 hours. **Planned:** Week 5, starting Aug 10.

Evidence

- Log file validation off and no KMS key on the trail; bucket default encryption is SSE-KMS.
- sf-log-archive-cloudtrail-us-east-1-000: versioning on, no Object Lock, bucket policy grants s3:* to the management account.
- Three Kestrel accounts also keep single-Region trails with the same gaps.

Examples: `sf-org-trail (management account)`, `sf-log-archive-cloudtrail-us-east-1-000`.

Exploitability. Needs a foothold inside, an insider, or physical access. Needs administrator access in the management or log-archive account. **Blast radius.** Every account in the organization. The only audit record for all 48 accounts.

How we verified it. Read the trail and bucket configuration on 07/20/2026.

Remediation: Replace the 2019 organization trail with a validated, encrypted trail in an immutable log archive bucket. `sf-org-trail` has no log file validation (CloudTrail.4) or KMS key (CloudTrail.2), and its bucket in sf-log-archive has no Object Lock and can be written and deleted from the management account. Build a new bucket in sf-log-archive with Object Lock in

COMPLIANCE mode for 400 days, then a new organization trail `sf-org-trail-v2` that logs management events and S3 data events for every bucket. Protect the key and the account, and alarm in the management account on any attempt to stop the trail, schedule the key for deletion, or close sf-log-archive.

Steps

1. Deploy the archive stack in sf-log-archive: a KMS key that is retained if the stack is deleted, an access-log bucket, and the Object Lock bucket with a bucket policy that allows only CloudTrail to write, plus the TLS-only Deny.
2. Deploy the trail stack in the management account: a multi-Region organization trail with log file validation and the KMS key, delivering to the new bucket.
3. Run both trails side by side for a week and compare event counts; do not stop the old trail until the new one is complete.
4. Deploy the EventBridge rule in the management account in us-east-1, where these API calls are recorded; it pages on StopLogging, DeleteTrail, UpdateTrail, PutEventSelectors, ScheduleKeyDeletion, DisableKey, and CloseAccount, and exempts no one, so a planned pipeline change pages too and the on-call confirms it.
5. Attach the `sf-protect-log-archive` SCP to the sf-log-archive account, and attach the IAM deny for `organizations:CloseAccount` to every administrator role in the management account. COMPLIANCE-mode objects are deleted if the account is closed, and an SCP cannot stop the management account from closing it, so the deny and the alarm are the controls on that path.
6. Stop the old trail, keep its bucket read-only for the retention period, and tighten its bucket policy to Allow statements for the security team only.
7. COMPLIANCE retention cannot be shortened by anyone, root included, so confirm the 400 days with Legal before the first object is written.

sf-log-archive: key, access-log bucket, and the Object Lock bucket (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Immutable CloudTrail archive for the organization trail.
Parameters:
  OrganizationId: {Type: String, Description: AWS Organizations ID (<org-id>).}
  ManagementAccountId: {Type: String, Description: Management account ID (<management-account-
id>).}
Resources:
  TrailKey:
    Type: AWS::KMS::Key
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      EnableKeyRotation: true
```

```
      PendingWindowInDays: 30
      KeyPolicy:
        Version: '2012-10-17'
        Statement:
          - Sid: AccountAdministration
            Effect: Allow
            Principal: {AWS: !Sub 'arn:aws:iam::${AWS::AccountId}:root'}
            Action: 'kms:*'
            Resource: '*'
          - Sid: CloudTrailDescribesKey
            Effect: Allow
            Principal: {Service: cloudtrail.amazonaws.com}
            Action: 'kms:DescribeKey'
            Resource: '*'
            Condition:
              StringEquals: {'aws:SourceArn': !Sub 'arn:aws:cloudtrail:us-east-1:$
{ManagementAccountId}:trail/sf-org-trail-v2'}
          - Sid: CloudTrailEncrypts
            Effect: Allow
            Principal: {Service: cloudtrail.amazonaws.com}
            Action: 'kms:GenerateDataKey*'
            Resource: '*'
            Condition:
              StringEquals: {'aws:SourceArn': !Sub 'arn:aws:cloudtrail:us-east-1:$
{ManagementAccountId}:trail/sf-org-trail-v2'}
              StringLike: {'kms:EncryptionContext:aws:cloudtrail:arn': !Sub
'arn:aws:cloudtrail:*:${ManagementAccountId}:trail/sf-org-trail-v2'}
          - Sid: OrganizationReadersDescribeKey
            Effect: Allow
            Principal: {AWS: '*'}
            Action: 'kms:DescribeKey'
            Resource: '*'
            Condition:
              StringEquals: {'aws:PrincipalOrgID': !Ref OrganizationId}
          - Sid: OrganizationReadersDecrypt
            Effect: Allow
            Principal: {AWS: '*'}
            Action: 'kms:Decrypt'
            Resource: '*'
            Condition:
              StringEquals: {'aws:PrincipalOrgID': !Ref OrganizationId}
              StringLike: {'kms:EncryptionContext:aws:cloudtrail:arn': !Sub
'arn:aws:cloudtrail:*:${ManagementAccountId}:trail/sf-org-trail-v2'}
  AccessLogBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Sub 'sf-log-archive-access-logs-${AWS::AccountId}-${AWS::Region}'
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault: {SSEAlgorithm: AES256}
      PublicAccessBlockConfiguration: {BlockPublicAcls: true, BlockPublicPolicy: true,
IgnorePublicAcls: true, RestrictPublicBuckets: true}
  AccessLogBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AccessLogBucket
      PolicyDocument:
```

```
        Version: '2012-10-17'
        Statement:
          - Sid: S3ServerAccessLogsDelivery
            Effect: Allow
            Principal: {Service: logging.s3.amazonaws.com}
            Action: 's3:PutObject'
            Resource: !Sub '${AccessLogBucket.Arn}/*'
            Condition: {StringEquals: {'aws:SourceAccount': !Ref 'AWS::AccountId'}}
          - Sid: DenyNonTlsRequests
            Effect: Deny
            Principal: '*'
            Action: 's3:*'
            Resource: [!GetAtt AccessLogBucket.Arn, !Sub '${AccessLogBucket.Arn}/*']
            Condition: {Bool: {'aws:SecureTransport': 'false'}}
  TrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      BucketName: sf-log-archive-org-trail-locked-us-east-1
      ObjectLockEnabled: true
      ObjectLockConfiguration:
        ObjectLockEnabled: Enabled
        Rule: {DefaultRetention: {Mode: COMPLIANCE, Days: 400}}
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - BucketKeyEnabled: true
            ServerSideEncryptionByDefault: {SSEAlgorithm: 'aws:kms', KMSMasterKeyID: !GetAtt
TrailKey.Arn}
      PublicAccessBlockConfiguration: {BlockPublicAcls: true, BlockPublicPolicy: true,
IgnorePublicAcls: true, RestrictPublicBuckets: true}
      LoggingConfiguration: {DestinationBucketName: !Ref AccessLogBucket, LogFilePrefix:
cloudtrail-v2/}
  TrailBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref TrailBucket
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Sid: CloudTrailAclCheck
            Effect: Allow
            Principal: {Service: cloudtrail.amazonaws.com}
            Action: 's3:GetBucketAcl'
            Resource: !GetAtt TrailBucket.Arn
            Condition: {StringEquals: {'aws:SourceArn': !Sub 'arn:aws:cloudtrail:us-east-1:$
{ManagementAccountId}:trail/sf-org-trail-v2'}}
          - Sid: CloudTrailWriteOrganizationLogs
            Effect: Allow
            Principal: {Service: cloudtrail.amazonaws.com}
            Action: 's3:PutObject'
            Resource: [!Sub '${TrailBucket.Arn}/AWSLogs/${ManagementAccountId}/*', !Sub '$
{TrailBucket.Arn}/AWSLogs/${OrganizationId}/*']
            Condition:
              StringEquals:
                's3:x-amz-acl': bucket-owner-full-control
                'aws:SourceArn': !Sub 'arn:aws:cloudtrail:us-east-1:$
```

```
{ManagementAccountId}:trail/sf-org-trail-v2'
          - Sid: DenyNonTlsRequests
            Effect: Deny
            Principal: '*'
            Action: 's3:*'
            Resource: [!GetAtt TrailBucket.Arn, !Sub '${TrailBucket.Arn}/*']
            Condition: {Bool: {'aws:SecureTransport': 'false'}}
Outputs:
  TrailKeyArn: {Value: !GetAtt TrailKey.Arn}
```

Management account: the new organization trail (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Multi-Region organization trail with log file validation, delivered to the
immutable archive.
Parameters:
  TrailKeyArn: {Type: String, Description: KMS key ARN from the sf-log-archive stack.}
Resources:
  OrgTrail:
    Type: AWS::CloudTrail::Trail
    Properties:
      TrailName: sf-org-trail-v2
      S3BucketName: sf-log-archive-org-trail-locked-us-east-1
      KMSKeyId: !Ref TrailKeyArn
      IsLogging: true
      IsMultiRegionTrail: true
      IsOrganizationTrail: true
      IncludeGlobalServiceEvents: true
      EnableLogFileValidation: true
      EventSelectors:
        - ReadWriteType: All
          IncludeManagementEvents: true
          DataResources:
            - {Type: 'AWS::S3::Object', Values: ['arn:aws:s3']}
            - {Type: 'AWS::Lambda::Function', Values: ['arn:aws:lambda']}
```

Page when anyone tries to stop, delete, or change a trail, its key, or the archive account (EventBridge event pattern)

```JSON
{
  "source": [
    "aws.cloudtrail",
    "aws.kms",
    "aws.organizations"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "cloudtrail.amazonaws.com",
      "kms.amazonaws.com",
      "organizations.amazonaws.com"
    ],
    "eventName": [
      "StopLogging",
      "DeleteTrail",
      "UpdateTrail",
      "PutEventSelectors",
      "ScheduleKeyDeletion",
      "DisableKey",
      "CloseAccount"
    ]
  }
}
```

Page on tampering from the management account in us-east-1 (no exemption, so a planned change pages too) (AWS CLI)

```BASH
# Run as an administrator of the management account. These API calls are recorded in the
management account,
# so the rule must live here: a rule in sf-security-tooling never sees them.
aws sns create-topic --region us-east-1 --name sf-trail-tamper-pagerduty
aws events put-rule --region us-east-1 --name sf-trail-tamper --state ENABLED --event-pattern
file://trail-tamper-pattern.json
aws events put-targets --region us-east-1 --rule sf-trail-tamper \
  --targets Id=PagerDuty,Arn=arn:aws:sns:us-east-1:<management-account-id>:sf-trail-tamper-
pagerduty
aws events list-targets-by-rule --region us-east-1 --rule sf-trail-tamper --query
'Targets[].Arn'
```

sf-protect-log-archive: nobody in sf-log-archive can schedule key deletion, disable a key, or delete the buckets (attach to the account itself) (Service control policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ProtectLogArchiveKeysAndBuckets",
      "Effect": "Deny",
      "Resource": "*",
      "Action": [
        "kms:ScheduleKeyDeletion",
        "kms:DisableKey",
        "s3:DeleteBucket",
        "s3:PutBucketObjectLockConfiguration"
      ]
    }
  ]
}
```

Management account: attach to every administrator role so no one can close sf-log-archive (SCPs do not bind the management account) (IAM policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "NobodyClosesMemberAccounts",
      "Effect": "Deny",
      "Action": "organizations:CloseAccount",
      "Resource": "*"
    }
  ]
}
```

Deploy both stacks and check the new trail (AWS CLI)

```bash
aws cloudformation deploy --profile <log-archive-profile> --region us-east-1 --stack-name sf-trail-archive \
  --template-file trail-archive.yaml --parameter-overrides OrganizationId=<org-id> ManagementAccountId=<management-account-id>
KEY=$(aws cloudformation describe-stacks --profile <log-archive-profile> --region us-east-1 --stack-name sf-trail-archive \
  --query 'Stacks[0].Outputs[?OutputKey==`TrailKeyArn`].OutputValue' --output text)
aws cloudformation deploy --profile <management-profile> --region us-east-1 --stack-name sf-org-trail-v2 \
  --template-file org-trail.yaml --parameter-overrides TrailKeyArn="$KEY"
aws cloudtrail get-trail-status --profile <management-profile> --name sf-org-trail-v2 --query '[IsLogging,LatestDeliveryError]'
aws cloudtrail describe-trails --profile <management-profile> --trail-name-list sf-org-trail-v2 \
  --query 'trailList[].[LogFileValidationEnabled,KmsKeyId,IsOrganizationTrail]'
```

How it is verified

- `aws cloudtrail describe-trails` shows `LogFileValidationEnabled` true and a `KmsKeyId` for `sf-org-trail-v2` (Security Hub CSPM CloudTrail.4, CloudTrail.2).
- `aws s3api get-object-lock-configuration` on the new bucket shows COMPLIANCE for 400 days, and `aws s3api delete-object --bucket sf-log-archive-org-trail-locked-us-east-1 --key <log-object-key> --version-id <version-id>` run as an administrator of sf-log-archive fails with AccessDenied because of the retention.
- `aws cloudtrail get-event-selectors --trail-name sf-org-trail-v2` shows S3 data events for all buckets, and `aws kms describe-key` on the trail key shows KeyState Enabled with no deletion date.
- A test `StopLogging` on a scratch trail in the management account pages the security team, and `aws cloudtrail validate-logs` passes for the last 24 hours.

How to roll it back. Restart the old trail with `aws cloudtrail start-logging` while both trails run side by side; stop the new trail only after that. The Object Lock bucket in COMPLIANCE mode cannot be emptied or shortened by anyone, root included, for 400 days, so the bucket stays even if the trail is removed. The key has `DeletionPolicy: Retain`, so deleting the stack leaves it in place; never schedule it for deletion while any object it encrypts is retained. Revert the trail stack and the EventBridge rule in the management account. To lift the protections, detach `sf-protect-log-archive` from sf-log-archive and remove the `organizations:CloseAccount` deny from the management roles, and expect the alarm to page. Verify by checking the event counts of both trails.

SF-SEC-026: Service control policies stop nothing: only FullAWSAccess and DenyLeaveOrganization are attached

High. Benchmark rating High (Vantalect baseline: organization guardrails). How it was rated: benchmark High (3) -1 exploitability (limited) +1 blast radius (organization) = 3

Where: All 48 accounts in 17 Regions, 2 policies. **Controls:** No benchmark control covers this. **SOC 2:** CC5.2, CC6.1. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 24 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- No resource control policies or declarative policies are enabled.
- Control Tower's guardrail SCPs were detached on 03/14/2023 and never reattached.

Examples: `FullAWSAccess`, `DenyLeaveOrganization`.

Exploitability. Needs a foothold inside, an insider, or physical access. Any administrator in any member account can disable logging, detection, or encryption defaults. **Blast radius.** Every account in the organization. Every member account.

How we verified it. Read list-policies and list-targets-for-policy for every policy type.

Remediation: Add the guardrail SCPs, the TLS resource control policy, and the EC2 declarative policy. The organization has only FullAWSAccess and a deny-leave-organization SCP, and no resource control policies or declarative policies. Add four SCPs: one protects the security services and the trail, one denies member-account root, one denies the 11 unused Regions, and one keeps the account defaults on (EBS encryption; S3, snapshot, and AMI Block Public Access; RDS snapshot sharing; KMS key rotation) and stops new IAM users, access keys, and console passwords. Then enable and attach the RCP and declarative policy types.

Steps

1. Create the four SCPs below; each is well under the 10,240-character limit, and an entity can hold 10 SCPs.
2. Attach each to the Sandbox OU first, run a day of normal work and the platform pipelines, then attach to the workload OUs and finally the root.
3. Apply the member-root deny only after the root credential cleanup finishes (see root-credentials). Its `Null` condition on `aws:AssumedRoot` exempts the short root sessions that `sts:AssumeRoot` starts from the management account; without it the deny also blocks the audit, credential-deletion, and bucket-unlock tasks that centralized root access exists for. Before attaching it to the root, start an `IAMAuditRootUserCredentials` session in a sandbox account that already has the policy and confirm it works.

4. Enable the resource control policy type and attach the S3 TLS RCP (see s3-tls-rcp); enable the declarative policy type and attach the EC2 policy (see imdsv2 and ebs-snapshot-public). RCPs do not affect the management account.
5. Keep the exception roles (PlatformAdmin, AWSControlTowerExecution, stack set execution roles) in the policies so pipelines keep working, and review the exceptions every quarter.

sf-protect-security-services: protect CloudTrail, Config, and the detection services (Service control policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ProtectSecurityServices",
      "Effect": "Deny",
      "Resource": "*",
      "Action": [
        "cloudtrail:StopLogging",
        "cloudtrail:DeleteTrail",
        "cloudtrail:UpdateTrail",
        "cloudtrail:PutEventSelectors",
        "config:StopConfigurationRecorder",
        "config:DeleteConfigurationRecorder",
        "config:DeleteDeliveryChannel",
        "config:DeleteConfigurationAggregator",
        "guardduty:DeleteDetector",
        "guardduty:DisassociateFromAdministratorAccount",
        "guardduty:StopMonitoringMembers",
        "securityhub:DisableSecurityHub",
        "securityhub:DisassociateFromAdministratorAccount",
        "securityhub:DeleteMembers",
        "access-analyzer:DeleteAnalyzer",
        "inspector2:Disable"
      ],
      "Condition": {
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::*:role/PlatformAdmin",
            "arn:aws:iam::*:role/AWSControlTowerExecution",
            "arn:aws:iam::*:role/stacksets-exec-*",
            "arn:aws:iam::*:role/PlatformSecurityAdmin"
          ]
        }
      }
    }
  ]
}
```

sf-deny-member-root: deny every action by a member account's root user, except root sessions started with sts:AssumeRoot (Service control policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyMemberRoot",
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aws:PrincipalArn": "arn:aws:iam::*:root"
        },
        "Null": {
          "aws:AssumedRoot": "true"
        }
      }
    }
  ]
}
```

sf-protect-account-defaults: keep the account defaults on, protect sharing and key rotation, and stop new IAM users (Service control policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "KeepEbsEncryption",
      "Effect": "Deny",
      "Action": [
        "ec2:DisableEbsEncryptionByDefault"
      ],
      "Resource": "*"
    },
    {
      "Sid": "KeepS3AccountBlock",
      "Effect": "Deny",
      "Action": [
        "s3:PutAccountPublicAccessBlock"
      ],
      "Resource": "*",
      "Condition": {
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::*:role/PlatformAdmin",
            "arn:aws:iam::*:role/AWSControlTowerExecution",
            "arn:aws:iam::*:role/stacksets-exec-*"
          ]
        }
      }
    },
    {
```

```
      "Sid": "KeepSnapshotAndImageBlocks",
      "Effect": "Deny",
      "Action": [
        "ec2:DisableSnapshotBlockPublicAccess",
        "ec2:DisableImageBlockPublicAccess"
      ],
      "Resource": "*",
      "Condition": {
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::*:role/PlatformAdmin",
            "arn:aws:iam::*:role/AWSControlTowerExecution",
            "arn:aws:iam::*:role/stacksets-exec-*"
          ]
        }
      }
    },
    {
      "Sid": "ProtectRdsSnapshotSharing",
      "Effect": "Deny",
      "Action": [
        "rds:ModifyDBSnapshotAttribute",
        "rds:ModifyDBClusterSnapshotAttribute"
      ],
      "Resource": "*",
      "Condition": {
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::*:role/PlatformAdmin",
            "arn:aws:iam::*:role/AWSControlTowerExecution",
            "arn:aws:iam::*:role/stacksets-exec-*"
          ]
        }
      }
    },
    {
      "Sid": "ProtectKeyRotation",
      "Effect": "Deny",
      "Action": [
        "kms:DisableKeyRotation"
      ],
      "Resource": "*",
      "Condition": {
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::*:role/PlatformAdmin",
            "arn:aws:iam::*:role/AWSControlTowerExecution",
            "arn:aws:iam::*:role/stacksets-exec-*"
          ]
        }
      }
    },
    {
      "Sid": "NoNewIamUsers",
      "Effect": "Deny",
      "Action": [
        "iam:CreateUser",
```

```
        "iam:CreateAccessKey",
        "iam:CreateLoginProfile",
        "iam:UpdateLoginProfile"
      ],
      "Resource": "*",
      "Condition": {
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::*:role/PlatformIdentityAdmin"
          ]
        }
      }
    }
  ]
}
```

sf-deny-unused-regions: deny the 11 unused Regions, with global services and platform roles excepted (Service control policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyUnusedRegions",
      "Effect": "Deny",
      "Resource": "*",
      "NotAction": [
        "acm:*",
        "aws-portal:*",
        "budgets:*",
        "ce:*",
        "cloudfront:*",
        "cur:*",
        "directconnect:*",
        "ec2:DescribeRegions",
        "fms:*",
        "globalaccelerator:*",
        "health:*",
        "iam:*",
        "networkmanager:*",
        "organizations:*",
        "pricing:*",
        "route53:*",
        "route53domains:*",
        "shield:*",
        "sts:*",
        "support:*",
        "trustedadvisor:*",
        "waf:*",
        "wafv2:*",
        "config:*",
        "guardduty:*",
        "securityhub:*",
        "access-analyzer:*",
```

```
        "inspector2:*",
        "cloudtrail:*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:RequestedRegion": [
            "us-east-1",
            "us-west-2",
            "eu-central-1",
            "ap-northeast-1",
            "eu-west-1",
            "ap-southeast-1"
          ]
        },
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::*:role/PlatformAdmin",
            "arn:aws:iam::*:role/AWSControlTowerExecution",
            "arn:aws:iam::*:role/stacksets-exec-*"
          ]
        }
      }
    }
  ]
}
```

Create the policies, enable the types, and attach to Sandbox first (AWS CLI)

```bash
ROOT=<org-root-id>
for type in RESOURCE_CONTROL_POLICY DECLARATIVE_POLICY_EC2; do
  aws organizations enable-policy-type --root-id "$ROOT" --policy-type $type || true
done
for name in sf-protect-security-services sf-deny-member-root sf-protect-account-defaults sf-
deny-unused-regions; do
  file=policies/scp/${name#sf-}.json
  id=$(aws organizations create-policy --name $name --type SERVICE_CONTROL_POLICY --description
"$name" \
    --content file://$file --query Policy.PolicySummary.Id --output text)
  aws organizations attach-policy --policy-id "$id" --target-id <sandbox-ou-id>
done
```

How it is verified

- `aws organizations list-policies-for-target --target-id <org-root-id> --filter SERVICE_CONTROL_POLICY` lists the four new SCPs, and the RCP and the two EC2 declarative policies are attached to the root.
- `cloudtrail:StopLogging` from an administrator who is not the security role is denied, and an EC2 call in `us-east-2` is denied.

- With `sf-deny-member-root` attached, `aws sts assume-root --target-principal <sandbox-account-id> --task-policy-arn arn=arn:aws:iam::aws:policy/root-task/IAMAuditRootUserCredentials` succeeds and its session can list the root user's credentials, while a call made with the account's real root user credentials is denied.
- The platform pipelines and the Control Tower baseline run unchanged for a week after the root attachment.

How to roll it back. Detach the SCP from the narrowest OU that blocks a legitimate action, with `aws organizations detach-policy`, and reattach it after the exception role is added. Detach in the reverse of the attach order: root first, then the workload OUs, then Sandbox. Detach the RCP and the declarative policy the same way. Never roll back by adding `Allow *` to a policy. Verify with the denied action.

SF-SEC-027: SSH bastions accept connections from the whole internet

High. Benchmark rating Critical (Security Hub CSPM EC2.19, EC2.53, EC2.54, and EC2.18). How it was rated: benchmark Critical (4) -1 exploitability (limited) +1 blast radius (organization) = 4; rule 2: Critical needs a remote path and a production-or-wider blast radius; capped at High

Where: `sf-shared-services` (…1188) in us-east-1, 1 security group. **Controls:** `EC2.18` (Security groups should only allow unrestricted incoming traffic for authorized ports); `EC2.19` (Security groups should not allow unrestricted access to ports with high risk); `EC2.53` (EC2 security groups should not allow ingress from 0.0.0.0/0 to remote server administration ports). **SOC 2:** CC6.6. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 6 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- 1 security group admits 0.0.0.0/0 on port 22; 7 instances attached.
- 0 also admit ::/0.

Examples: `sg-e4d355829b0619c3f`, `i-0ac135ca8063abfa2`, `i-03522e38c0602b04a`, `i-01cb1546f5ce3868a`.

Exploitability. Needs a foothold inside, an insider, or physical access. Key-based SSH only; exposure is to SSH exploits and stolen keys. **Blast radius.** Every account in the organization. The bastions reach every VPC through the Transit Gateway.

How we verified it. Matched each rule to its owner and use with the account owner.

Remediation: Close SSH and RDP to the internet and use Session Manager instead. This is the same change as SF-SEC-009, applied to the accounts listed above. Use the steps, the code, the verification, and the rollback under SF-SEC-009, with this finding's accounts in place of that finding's.

SF-SEC-028: SSH is open to the internet on 76 development instances

High. Benchmark rating Critical (Security Hub CSPM EC2.19, EC2.53, EC2.54, and EC2.18). How it was rated: benchmark Critical (4) +0 exploitability (moderate) -1 blast radius (account) = 3

Where: `fleet-dev` (…6037) in us-east-1, 7 security groups. **Controls:** `EC2.18` (Security groups should only allow unrestricted incoming traffic for authorized ports); `EC2.19` (Security groups should not allow unrestricted access to ports with high risk); `EC2.53` (EC2 security groups should not allow ingress from 0.0.0.0/0 to remote server administration ports); `EC2.54` (EC2 security groups should not allow ingress from ::/0 to remote server administration ports). **SOC 2:** CC6.6. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 18 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- 7 security groups admit 0.0.0.0/0 on port 22; 76 instances attached.
- 6 also admit ::/0.

Examples: `sg-1651637bd778bf555`, `sg-4765828c5ce65574c`, `sg-8fea77f453540ad59`, `sg-9777ee769b7645f88`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Shared keys circulate among developers. **Blast radius.** One non-production account. Development data and credentials.

How we verified it. Matched each rule to its owner and use with the account owner.

Remediation: Close SSH and RDP to the internet and use Session Manager instead. This is the same change as SF-SEC-009, applied to the accounts listed above. Use the steps, the code, the verification, and the rollback under SF-SEC-009, with this finding's accounts in place of that finding's.

SF-SEC-029: 4 more member-account root users have no MFA; 40 use virtual MFA

High. Benchmark rating Critical (Security Hub CSPM IAM.9 and IAM.6). How it was rated: benchmark Critical (4) -1 exploitability (limited) +0 blast radius (production) = 3

Where: 44 accounts in us-east-1, 44 root users. **Controls:** `IAM.6` (Hardware MFA should be enabled for the root user); `IAM.9` (MFA should be enabled for the root user). **SOC 2:** CC6.1, CC6.2. **Owner:** Grace Liu, Manager, Identity and Access. **Repository:** `platform-landing-zone`. **Effort:** 6 hours. **Planned:** Week 4, starting Aug 3.

Evidence

- IAM.9 failed in 4 accounts; IAM.6 (hardware MFA) failed in 44.
- Member-account root passwords exist everywhere because centralized root access management is off.

Examples: `kestrel-staging (…9054)`, `kestrel-dev (…9076)`, `sandbox-hackathon (…4078)`, `suspended-legacy-ci (…4099)`.

Exploitability. Needs a foothold inside, an insider, or physical access. A password reset needs the root mailbox; two Kestrel root mailboxes are on a domain IT plans to retire. **Blast radius.** Production systems or customer data. Full control of each account, including its billing and support settings.

How we verified it. Checked each account's root MFA device type in the credential report on 07/20/2026.

Remediation: Delete member-account root credentials centrally and protect the management root with hardware MFA. This is the same change as SF-SEC-010, applied to the accounts listed above. Use the steps, the code, the verification, and the rollback under SF-SEC-010, with this finding's accounts in place of that finding's.

SF-SEC-030: 23 robot certificates are shared by 1,840 robots at 3 customer sites

High. Benchmark rating Critical (AWS IoT Device Defender DEVICE_CERTIFICATE_SHARED_CHECK). How it was rated: benchmark Critical (4) -1 exploitability (limited) +0 blast radius (production) = 3

Where: `fleet-ingest-prod` (…5012) in 2 Regions, 23 certificates. **Controls:** No Security Hub CSPM control covers this. Found by: DEVICE_CERTIFICATE_SHARED_CHECK; CONFLICTING_CLIENT_IDS_CHECK. **SOC 2:** CC6.1, CC6.6. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 32 hours. **Planned:** Week 3, starting Jul 27; the rest (Re-provision the robots at the Bremen and Ota sites during their maintenance windows) is due September 30, 2026.

Evidence

- Shared certificates by site: Hanse Cold Chain, Bremen; Kanto Precision Components, Ota; Ridgeline Logistics, Columbus distribution center.
- CONFLICTING_CLIENT_IDS_CHECK (High) fires on the same devices: robots knock each other offline several times an hour.
- Root cause: a 2024 refurbishment run cloned the provisioning image, certificate included.

Examples: `1f9f302f3683a3c0`, `4c8e5a61e0bc7e8d`, `4d19ad8a665bc973`, `Hanse Cold Chain, Bremen`, `Kanto Precision Components, Ota`.

Exploitability. Needs a foothold inside, an insider, or physical access. Needs physical access to one robot, but then impersonates every robot that shares its certificate. **Blast radius.** Production systems or customer data. No single robot can be revoked without disconnecting dozens of others.

How we verified it. Matched connection logs to robot serial numbers with fleet support on 07/23/2026; 2 lab rigs removed as false positives.

Remediation: Issue one certificate per robot through fleet provisioning and revoke the shared ones. Some certificates are shared by many robots (DEVICE_CERTIFICATE_SHARED_CHECK and CONFLICTING_CLIENT_IDS_CHECK), so one robot's connection kicks another off and revoking a stolen certificate would take all of them offline. Give each robot its own certificate through fleet provisioning by claim, with a pre-provisioning hook that lets only listed serial numbers through, and revoke the shared ones.

Steps

1. List the shared certificates and the robots on each from the Device Defender findings, and save the robots' serial numbers.
2. Deploy the pre-provisioning hook below and load the saved serial numbers into its allow list, so a stolen claim certificate cannot register a robot that is not on the list.
3. Create the claim certificate, attach a policy that allows only the provisioning topics, and create the provisioning template with the hook; the template's override settings let it adopt a robot that already exists as a thing.
4. Ship a firmware job (it reaches these robots through the jobs topics that `sf-robot-legacy-v1-restricted` allows) that generates a key on the robot, requests its own certificate with a CSR, and stores it.
5. Roll the job out by thing group; confirm each robot reconnects with its own certificate, then detach the shared certificate from that robot's thing.

6. Revoke each shared certificate only when no thing is attached to it, then rotate the claim certificate.

List shared certificates and the robots on them (AWS CLI)

```bash
TASK=$(aws iot start-on-demand-audit-task --target-check-names DEVICE_CERTIFICATE_SHARED_CHECK
CONFLICTING_CLIENT_IDS_CHECK --query taskId --output text)
# Wait until `aws iot describe-audit-task --task-id "$TASK" --query taskStatus` returns
COMPLETED.
aws iot list-audit-findings --task-id "$TASK" --check-name DEVICE_CERTIFICATE_SHARED_CHECK \
  --query 'findings[].nonCompliantResource.resourceIdentifier.deviceCertificateId' --output
text | tr '\t' '\n' | sort -u > shared-certificate-ids.txt
: > shared-robots.txt
for id in $(cat shared-certificate-ids.txt); do
  arn=$(aws iot describe-certificate --certificate-id "$id" --query
certificateDescription.certificateArn --output text)
  echo "$id $(aws iot list-principal-things --principal "$arn" --query 'things | length(@)')"
  aws iot list-principal-things --principal "$arn" --query 'things[]' --output text | tr '\t'
'\n' >> shared-robots.txt
done
```

Pre-provisioning hook and serial-number allow list (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Pre-provisioning hook that lets only listed robot serial numbers claim a
certificate.
Resources:
  SerialAllowList:
    Type: AWS::DynamoDB::Table
    Properties:
      TableName: sf-robot-serial-allowlist
      BillingMode: PAY_PER_REQUEST
      AttributeDefinitions: [{AttributeName: serial, AttributeType: S}]
      KeySchema: [{AttributeName: serial, KeyType: HASH}]
  HookRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: sf-robot-provisioning-hook
      ManagedPolicyArns: [arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole]
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - {Effect: Allow, Principal: {Service: lambda.amazonaws.com}, Action:
'sts:AssumeRole'}
      Policies:
        - PolicyName: read-allow-list
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - {Effect: Allow, Action: 'dynamodb:GetItem', Resource: !GetAtt
SerialAllowList.Arn}
  Hook:
```

```
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: sf-robot-provisioning-hook
      Runtime: python3.13
      Handler: index.handler
      Role: !GetAtt HookRole.Arn
      Timeout: 5
      Environment:
        Variables:
          TABLE: !Ref SerialAllowList
      Code:
        ZipFile: |
          import os
          import boto3

          table = boto3.resource("dynamodb").Table(os.environ["TABLE"])

          def handler(event, context):
              serial = (event.get("parameters") or {}).get("SerialNumber", "")
              found = bool(serial) and "Item" in table.get_item(Key={"serial": serial})
              return {"allowProvisioning": found}
  IotMayInvokeHook:
    Type: AWS::Lambda::Permission
    Properties:
      FunctionName: !Ref Hook
      Action: lambda:InvokeFunction
      Principal: iot.amazonaws.com
      SourceAccount: !Ref AWS::AccountId
Outputs:
  HookArn:
    Value: !GetAtt Hook.Arn
```

Claim certificate policy: provisioning topics only (AWS IoT policy)

```
JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:*:<fleet-ingest-prod-account-id>:client/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topic/$aws/certificates/create-from-csr/
*",
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topic/$aws/provisioning-templates/sf-
robot-fleet-provisioning/provision/*"
      ]
```

```
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topicfilter/$aws/certificates/create-
from-csr/*",
        "arn:aws:iot:*:<fleet-ingest-prod-account-id>:topicfilter/$aws/provisioning-templates/
sf-robot-fleet-provisioning/provision/*"
      ]
    }
  ]
}
```

Load the allow list, create the claim certificate and template, and retire shared certificates once empty (AWS CLI)

```bash
set -euo pipefail
while read -r serial; do
  aws dynamodb put-item --table-name sf-robot-serial-allowlist --item "{\"serial\": {\"S\":
\"$serial\"}}"
done < <(sort -u shared-robots.txt)

aws iot create-policy --policy-name sf-robot-claim --policy-document file://sf-robot-claim.json
umask 077
CLAIM_ARN=$(aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
claim-cert.pem \
  --public-key-outfile claim-public.key --private-key-outfile claim-private.key --query
certificateArn --output text)
aws iot attach-policy --policy-name sf-robot-claim --target "$CLAIM_ARN"
aws iot create-thing-group --thing-group-name sf-fleet-provisioned

cat > provisioning-template.json <<'JSON'
{"Parameters": {"SerialNumber": {"Type": "String"}, "AWS::IoT::Certificate::Id": {"Type":
"String"}},
 "Resources": {
   "thing": {"Type": "AWS::IoT::Thing",
             "Properties": {"ThingName": {"Ref": "SerialNumber"}, "ThingGroups": ["sf-fleet-
provisioned"]},
             "OverrideSettings": {"AttributePayload": "MERGE", "ThingTypeName": "REPLACE",
"ThingGroups": "DO_NOTHING"}},
   "certificate": {"Type": "AWS::IoT::Certificate", "Properties": {"CertificateId": {"Ref":
"AWS::IoT::Certificate::Id"}, "Status": "Active"}},
   "policy": {"Type": "AWS::IoT::Policy", "Properties": {"PolicyName": "sf-robot-scoped-v2"}}}}
JSON
HOOK_ARN=$(aws cloudformation describe-stacks --stack-name sf-robot-provisioning-hook --query
'Stacks[0].Outputs[?OutputKey==`HookArn`].OutputValue' --output text)
aws iot create-provisioning-template --template-name sf-robot-fleet-provisioning \
  --provisioning-role-arn arn:aws:iam::<fleet-ingest-prod-account-id>:role/
IotFleetProvisioningRole \
  --pre-provisioning-hook targetArn="$HOOK_ARN" \
  --template-body file://provisioning-template.json --enabled

# moved-robots.txt: robots that have reconnected with their own certificate. Detach the shared
```

```
one, and revoke it only when empty.
while read -r thing shared_arn; do
  aws iot detach-thing-principal --thing-name "$thing" --principal "$shared_arn"
done < moved-robots.txt
for id in $(cat shared-certificate-ids.txt); do
  arn=$(aws iot describe-certificate --certificate-id "$id" --query
certificateDescription.certificateArn --output text)
  left=$(aws iot list-principal-things --principal "$arn" --query 'things | length(@)')
  if [ "$left" = "0" ]; then aws iot update-certificate --certificate-id "$id" --new-status
REVOKED; else echo "keep $id: $left robots left"; fi
done
```

How it is verified

- A new on-demand audit shows no DEVICE_CERTIFICATE_SHARED_CHECK or CONFLICTING_CLIENT_IDS_CHECK finding.
- `aws iot list-principal-things` returns at most one thing for every active certificate.
- Every shared certificate shows status REVOKED, and no robot has connected with one since.
- A provisioning request from the claim certificate with a serial number that is not in `sf-robot-serial-allowlist` is refused by the hook.

How to roll it back. A revoked certificate cannot be used again, so revoke the shared certificate only after its last robot has moved. If a robot fails to reconnect on its own certificate, cancel the firmware job for its thing group with `aws iot cancel-job`, re-attach the shared certificate to its thing with `aws iot attach-thing-principal` if it was already detached and not yet revoked, and keep the certificate active until the robot is fixed. Delete the `sf-robot-provisioning-hook` stack and the provisioning template only after the claim certificate is deactivated. Rotate the claim certificate afterward. Verify with the robots' connection counts.

Medium findings

15 Medium findings, 462 hours between them.

SF-SEC-031: 1,247 Lambda functions run on deprecated runtimes

Medium. Benchmark rating Medium (Security Hub CSPM Lambda.2). How it was rated: benchmark Medium (2) +0 exploitability (moderate) +0 blast radius (production) = 2

Where: 5 accounts in 2 Regions, 1,247 Lambda functions. **Controls:** `Lambda.2` (Lambda functions should use supported runtimes). **SOC 2:** CC7.1, CC8.1. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 143 hours. **Planned:** Backlog, due January 29, 2027.

Evidence

- By runtime: nodejs20.x 418, python3.9 265, nodejs18.x 198, python3.8 158, nodejs16.x 124, go1.x 84.

Examples: `telemetry-5204f6`, `telemetry-d7390c`, `telemetry-5cd72b`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Deprecated runtimes receive no security patches. **Blast radius.** Production systems or customer data. Fleet API and ingest functions behind public APIs.

How we verified it. Read from the scan.

Remediation: Move Lambda functions off deprecated runtimes to nodejs24.x, python3.13, and provided.al2023. Many functions run deprecated runtimes: nodejs20.x, nodejs18.x, nodejs16.x, python3.9, python3.8, and go1.x (Lambda.2). They keep running, but creates are blocked from July 29, 2027 and updates from August 31, 2027, so they stop receiving fixes. Move them to nodejs24.x, python3.13, and provided.al2023 for Go.

Steps

1. List every function by runtime and rank by traffic and owner; the deprecation dates are go1.x January 8, 2024, nodejs16.x June 12, 2024, python3.8 October 14, 2024, nodejs18.x September 1, 2025, python3.9 December 15, 2025, and nodejs20.x April 30, 2026.
2. Upgrade Node.js functions to nodejs24.x and Python functions to python3.13, run the unit tests, and deploy to an alias with weighted traffic.
3. Rebuild each Go function as a custom runtime: compile for Linux ARM64, name the binary `bootstrap`, and set the runtime to provided.al2023.
4. Skip python3.10 as a target: it is deprecated on October 31, 2026.
5. Set the runtime in the CDK constructs so the next deploy keeps it, and stop new functions from using a deprecated runtime.

Inventory functions on deprecated runtimes (AWS CLI)

```bash
for region in us-east-1 us-west-2 eu-central-1 ap-northeast-1 eu-west-1 ap-southeast-1; do
  aws lambda list-functions --region $region \
    --query "Functions[?Runtime=='nodejs20.x' || Runtime=='nodejs18.x' || Runtime=='nodejs16.x'
|| Runtime=='python3.9' || Runtime=='python3.8' || Runtime=='go1.x'].[FunctionName,Runtime]" \
    --output text | sed "s/^/$region\t/"
done > deprecated-runtimes.tsv
cut -f3 deprecated-runtimes.tsv | sort | uniq -c
```

Rebuild a Go function for provided.al2023 and switch its runtime (AWS CLI)

```bash
GOOS=linux GOARCH=arm64 CGO_ENABLED=0 go build -tags lambda.norpc -o bootstrap main.go
zip function.zip bootstrap
aws lambda update-function-code --function-name <function-name> --zip-file fileb://function.zip
--architectures arm64
aws lambda wait function-updated --function-name <function-name>
aws lambda update-function-configuration --function-name <function-name> --runtime
provided.al2023 --handler bootstrap
aws lambda wait function-updated --function-name <function-name>
aws lambda publish-version --function-name <function-name>
```

Current runtimes in code (AWS CDK, TypeScript)

```typescript
import { Duration, Stack, StackProps, aws_lambda as lambda } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class TelemetryFunctionsStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new lambda.Function(this, 'Ingest', { runtime: lambda.Runtime.NODEJS_24_X, handler:
'index.handler', code: lambda.Code.fromAsset('dist/ingest'), timeout: Duration.seconds(30) });
    new lambda.Function(this, 'Report', { runtime: lambda.Runtime.PYTHON_3_13, handler:
'app.handler', code: lambda.Code.fromAsset('dist/report'), timeout: Duration.seconds(60) });
    new lambda.Function(this, 'Mapper', { runtime: lambda.Runtime.PROVIDED_AL2023,
architecture: lambda.Architecture.ARM_64, handler: 'bootstrap', code:
lambda.Code.fromAsset('dist/mapper') });
  }
}
```

How it is verified

- The inventory returns no function on a deprecated runtime (Security Hub CSPM Lambda.2).
- Each migrated function passed its tests and ran a full day of production traffic on the new runtime with unchanged error rates.
- A CDK deploy of the stacks leaves every function on its new runtime.

How to roll it back. Repoint the alias to the previous function version with `aws lambda update-alias`, which switches the traffic at once. A deprecated runtime still runs, so the rollback is safe until the dates in the steps. Revert the CDK change after the alias is back. Verify with the function's error rate and duration.

SF-SEC-032: EBS encryption by default is off in 762 account-Regions; 1,245 volumes are unencrypted

Medium. Benchmark rating Medium (Security Hub CSPM EC2.7). How it was rated: benchmark Medium (2) +0 exploitability (moderate) +0 blast radius (production) = 2

Where: All 48 accounts in 17 Regions, 1,245 EBS volumes. **Controls:** `EC2.7` (EBS default encryption should be enabled). **SOC 2:** CC6.1, C1.1. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 78 hours. **Planned:** Week 3, starting Jul 27; the rest (Migrate the remaining unencrypted volumes through encrypted snapshot copies) is due December 18, 2026.

Evidence

- 1,245 of 4,712 volumes unencrypted (435,126 GB).
- EBS encryption by default is on in only 54 account-Regions.

Examples: `vol-0890e81465a174ae9`, `vol-0baeb3e7e6d043055`, `vol-02c3bfe1524250024`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). With ec2:ModifySnapshotAttribute, an attacker copies an unencrypted volume out through a snapshot; a KMS key would also need a grant. **Blast radius.** Production systems or customer data. Unencrypted volumes in 38 accounts, including production databases on EC2.

How we verified it. Read from the scan.

Remediation: Turn on EBS encryption by default in every Region and replace every unencrypted volume. EBS encryption by default is off (EC2.7) and many volumes are unencrypted. Turn the default on in every account and Region, block anyone from turning it off, then replace each unencrypted volume through an encrypted snapshot copy.

Steps

1. Create the customer managed EBS key `alias/sf-ebs` in every account and Region (the stack set template below), with grants for the Auto Scaling service-linked role and for the forensics account. Snapshots encrypted with the AWS managed key `aws/ebs` can never be shared with another account, so the forensic share in the incident runbooks needs this key. Service-managed stack sets skip the management account, so deploy the same template there yourself.
2. Set it as the default key and run `enable-ebs-encryption-by-default` in every Region of every account.
3. Attach `sf-protect-account-defaults` (see scp-guardrails); its KeepEbsEncryption statement denies `ec2:DisableEbsEncryptionByDefault` to everyone, so the setting stays on.
4. List unencrypted volumes and group them by owning team; plan each replacement in a maintenance window.
5. For each volume: stop the instance, snapshot the volume, copy the snapshot with encryption, create a volume from it, and swap it onto the same device name.

6. Delete the old volume and snapshot only after the owner confirms the instance boots and the data is intact.

Customer managed EBS key per account and Region, shareable with the forensics account (deploy as a stack set) (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Default EBS key (alias/sf-ebs). Grants the Auto Scaling service-linked role and
the forensics account.
Parameters:
  ForensicsAccountId: {Type: String, Description: Forensics account ID (<forensics-account-
id>).}
Resources:
  EbsKey:
    Type: AWS::KMS::Key
    Properties:
      EnableKeyRotation: true
      KeyPolicy:
        Version: '2012-10-17'
        Statement:
          - Sid: AccountAdministration
            Effect: Allow
            Principal: {AWS: !Sub 'arn:aws:iam::${AWS::AccountId}:root'}
            Action: 'kms:*'
            Resource: '*'
          - Sid: AutoScalingUsesKey
            Effect: Allow
            Principal: {AWS: !Sub 'arn:aws:iam::${AWS::AccountId}:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling'}
            Action: ['kms:Encrypt', 'kms:Decrypt', 'kms:ReEncrypt*', 'kms:GenerateDataKey*',
'kms:DescribeKey']
            Resource: '*'
          - Sid: AutoScalingCreatesGrants
            Effect: Allow
            Principal: {AWS: !Sub 'arn:aws:iam::${AWS::AccountId}:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling'}
            Action: 'kms:CreateGrant'
            Resource: '*'
            Condition: {Bool: {'kms:GrantIsForAWSResource': 'true'}}
          - Sid: ForensicsAccountCopiesSnapshots
            Effect: Allow
            Principal: {AWS: !Sub 'arn:aws:iam::${ForensicsAccountId}:root'}
            Action: ['kms:DescribeKey', 'kms:Decrypt', 'kms:ReEncrypt*', 'kms:CreateGrant']
            Resource: '*'
  EbsKeyAlias:
    Type: AWS::KMS::Alias
    Properties:
      AliasName: alias/sf-ebs
      TargetKeyId: !Ref EbsKey
```

Deploy the key everywhere, then enable default encryption in every Region of the current account (AWS CLI)

```bash
# A key policy can name the Auto Scaling service-linked role only once it exists in that
account. Create it everywhere
# first; the call fails harmlessly where the role is already there.
for account in $(aws organizations list-accounts --query 'Accounts[?State==`ACTIVE`].Id' --
output text); do
  creds=$(aws sts assume-role --role-arn arn:aws:iam::$account:role/
OrganizationAccountAccessRole \
    --role-session-name ebs-key --query Credentials --output json) || { echo "SKIP $account:
cannot assume the role"; continue; }
  (export AWS_ACCESS_KEY_ID=$(echo "$creds" | jq -r .AccessKeyId) AWS_SECRET_ACCESS_KEY=$(echo
"$creds" | jq -r .SecretAccessKey)
   export AWS_SESSION_TOKEN=$(echo "$creds" | jq -r .SessionToken)
   aws iam create-service-linked-role --aws-service-name autoscaling.amazonaws.com 2>/dev/null
|| true)
done
aws cloudformation create-stack-set --region us-east-1 --stack-set-name sf-ebs-key --template-
body file://ebs-key.yaml \
  --parameters ParameterKey=ForensicsAccountId,ParameterValue=<forensics-account-id> \
  --permission-model SERVICE_MANAGED --auto-deployment
Enabled=true,RetainStacksOnAccountRemoval=false
aws cloudformation create-stack-instances --region us-east-1 --stack-set-name sf-ebs-key \
  --deployment-targets OrganizationalUnitIds=<org-root-id> --regions us-east-1 us-east-2 us-
west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-west-1 eu-west-2 eu-west-3 eu-north-1
ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-southeast-1 ap-southeast-2 ap-south-1
# Service-managed stack sets never deploy to the management account: create the same stack
there, in every Region.
for region in us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-
west-1 eu-west-2 eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-
southeast-1 ap-southeast-2 ap-south-1; do
  aws ec2 modify-ebs-default-kms-key-id --region $region --kms-key-id alias/sf-ebs
  aws ec2 enable-ebs-encryption-by-default --region $region
  aws ec2 get-ebs-encryption-by-default --region $region --query EbsEncryptionByDefault
done
```

sf-protect-account-defaults, statement KeepEbsEncryption: nobody may turn EBS default encryption off (the whole policy is in the guardrail SCPs) (Service control policy)

```json
{
  "Sid": "KeepEbsEncryption",
  "Effect": "Deny",
  "Action": [
    "ec2:DisableEbsEncryptionByDefault"
  ],
  "Resource": "*"
}
```

Replace one unencrypted volume with an encrypted copy (AWS CLI)

```bash
REGION=us-east-1
VOLUME=<unencrypted-volume-id>
INSTANCE=<instance-id>
DEVICE=<device-name>
AZ=$(aws ec2 describe-volumes --region $REGION --volume-ids $VOLUME --query
'Volumes[0].AvailabilityZone' --output text)
aws ec2 stop-instances --region $REGION --instance-ids $INSTANCE && aws ec2 wait instance-
stopped --region $REGION --instance-ids $INSTANCE
SNAP=$(aws ec2 create-snapshot --region $REGION --volume-id $VOLUME --query SnapshotId --output
text)
aws ec2 wait snapshot-completed --region $REGION --snapshot-ids $SNAP
ENC=$(aws ec2 copy-snapshot --region $REGION --source-region $REGION --source-snapshot-id $SNAP
--encrypted \
        --kms-key-id alias/sf-ebs --query SnapshotId --output text)
aws ec2 wait snapshot-completed --region $REGION --snapshot-ids $ENC
NEW=$(aws ec2 create-volume --region $REGION --snapshot-id $ENC --availability-zone $AZ --query
VolumeId --output text)
aws ec2 wait volume-available --region $REGION --volume-ids $NEW
aws ec2 detach-volume --region $REGION --volume-id $VOLUME && aws ec2 wait volume-available --
region $REGION --volume-ids $VOLUME
aws ec2 attach-volume --region $REGION --volume-id $NEW --instance-id $INSTANCE --device
$DEVICE
aws ec2 start-instances --region $REGION --instance-ids $INSTANCE
```

How it is verified

- `aws ec2 get-ebs-encryption-by-default` returns true in every Region of every account (Security Hub CSPM EC2.7).
- `aws ec2 describe-volumes --filters Name=encrypted,Values=false` returns nothing in the account.
- `ec2:DisableEbsEncryptionByDefault` from an administrator role is denied by `sf-protect-account-defaults`.
- `aws ec2 get-ebs-default-kms-key-id` returns `alias/sf-ebs` in every Region, and a snapshot encrypted with it can be shared with the forensics account (a snapshot under `aws/ebs` cannot).

How to roll it back. For a replaced volume, reattach the original volume to the instance at the same device name; it is kept until step 5. Turn the default off in a Region with `aws ec2 disable-ebs-encryption-by-default` only after detaching the SCP, and only for an incident. Volumes already encrypted stay encrypted. Verify by booting the instance and checking the volume's `Encrypted` flag.

SF-SEC-033: 556 running instances are not managed by Systems Manager; 2,380 run AMIs over a year old

Medium. Benchmark rating Medium (Security Hub CSPM SSM.1). How it was rated: benchmark Medium (2) +0 exploitability (moderate) +0 blast radius (production) = 2

Where: 37 accounts in 5 Regions, 556 instances. **Controls:** `SSM.1` (Amazon EC2 instances should be managed by AWS Systems Manager). **SOC 2:** CC7.1, CC6.8. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 20 hours. **Planned:** Backlog, due October 30, 2026.

Evidence

- No SSM Agent registration or no instance profile with the managed-instance policy.

Examples: `i-07f84b51b5a841da7`, `i-0f50f34e9fe5b0d89`, `i-04dbeca4c521a56e5`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Unpatched hosts with known vulnerabilities. **Blast radius.** Production systems or customer data. Mostly Kestrel, sandboxes, and build hosts.

How we verified it. Read from the scan.

Remediation: Bring every instance under Systems Manager and patch on a schedule. Instances are not managed by Systems Manager (SSM.1) and some AMIs are more than a year old. Turn on Default Host Management Configuration so instances register without per-instance setup, patch on a schedule, and rebuild old AMIs.

Steps

1. Deploy the default host management role to each account, then enable the setting in every Region the account uses.
2. List instances that still do not show as managed nodes; they usually lack the agent or outbound access to the Systems Manager endpoints.
3. Create the organization patch policy with Systems Manager Quick Setup, or per account a State Manager association that runs `AWS-RunPatchBaseline`, scanning weekly and installing in the maintenance window.
4. List AMIs and running instances older than a year, rebuild them from current base images with an EC2 Image Builder pipeline, and roll the instances.
5. Deregister AMIs nobody uses after a hold, and keep the patch compliance report in the monthly platform review.

Default host management role (deploy as a stack set) (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Role Systems Manager uses to manage instances without an instance profile.
Resources:
  DefaultHostManagementRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: AWSSystemsManagerDefaultEC2InstanceManagementRole
      Path: /service-role/
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonSSMManagedEC2InstanceDefaultPolicy
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal: {Service: ssm.amazonaws.com}
            Action: 'sts:AssumeRole'
```

Enable the setting and schedule patching in one Region (AWS CLI)

```bash
REGION=us-east-1
aws ssm update-service-setting --region $REGION \
  --setting-id /ssm/managed-instance/default-ec2-instance-management-role \
  --setting-value AWSSystemsManagerDefaultEC2InstanceManagementRole
aws ssm create-association --region $REGION --name AWS-RunPatchBaseline --association-name sf-
weekly-patch-scan \
  --targets Key=InstanceIds,Values='*' --parameters Operation=Scan --schedule-expression
'cron(0 3 ? * SUN *)'
aws ssm describe-instance-information --region $REGION --query 'InstanceInformationList[?
PingStatus!=`Online`].InstanceId' --output text
```

Find AMIs and instances older than a year (AWS CLI)

```bash
CUTOFF=$(date -u -d '365 days ago' +%Y-%m-%dT%H:%M:%S.000Z)
aws ec2 describe-images --owners self --query "Images[?CreationDate<'${CUTOFF}'].
[ImageId,Name,CreationDate]" --output text
aws ec2 describe-instances --query "Reservations[].Instances[?LaunchTime<'${CUTOFF}'].
[InstanceId,ImageId,LaunchTime]" --output text
```

How it is verified

- `aws ssm describe-instance-information` lists every running instance with `PingStatus` Online (Security Hub CSPM SSM.1).
- The Patch Manager compliance summary shows no instance missing critical patches for more than 30 days.
- No running instance uses an AMI older than a year.

How to roll it back. Turn off Default Host Management Configuration with `aws ssm reset-service-setting`, and delete the patch association or the Quick Setup configuration. Patches that are installed stay installed; restore an instance from its pre-patch snapshot if one breaks an application. Verify by listing the managed nodes.

SF-SEC-034: 39 internet-facing ALBs have no AWS WAF; 84 ALBs do not redirect HTTP to HTTPS

Medium. Benchmark rating Medium (Security Hub CSPM ELB.1). How it was rated: benchmark Medium (2) +0 exploitability (moderate) +0 blast radius (production) = 2

Where: 8 accounts in 5 Regions, 123 load balancers. **Controls:** `ELB.1` (Application Load Balancer should be configured to redirect all HTTP requests to HTTPS). **SOC 2:** CC6.6, CC6.7. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 16 hours. **Planned:** Backlog, due October 30, 2026.

Evidence

- 17 Classic Load Balancers remain (Kestrel and legacy).

Examples: `telemetry-lb-91e29d`, `telemetry-lb-60132f`, `telemetry-lb-49a374`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Application attacks reach targets unfiltered. **Blast radius.** Production systems or customer data. Fleet API and staging endpoints.

How we verified it. Read from the scan.

Remediation: Redirect HTTP to HTTPS, put AWS WAF in front of public ALBs, and migrate Classic Load Balancers. Application Load Balancers serve HTTP without redirecting (ELB.1), internet-facing ALBs have no AWS WAF web ACL, and Classic Load Balancers remain. Add the redirect, attach a web ACL with AWS managed rule groups, and replace each Classic Load Balancer with an ALB or NLB.

Steps

1. Change every HTTP listener's default action to a permanent redirect to HTTPS on 443.
2. Deploy the web ACL as a stack set and attach it to each internet-facing ALB; start with the rule groups in count mode for a week, then block.
3. For each Classic Load Balancer, create an ALB (HTTP and HTTPS) or NLB (TCP) with matching target groups and health checks.
4. Move traffic with weighted Route 53 records, watch errors for a day, then retire the Classic Load Balancer.

5. Put the redirect in the CDK load balancer constructs so a rebuild keeps it.

Redirect HTTP listeners and list the Classic Load Balancers (AWS CLI)

```bash
REGION=us-east-1
for lb in $(aws elbv2 describe-load-balancers --region $REGION --query 'LoadBalancers[?
Type==`application`].LoadBalancerArn' --output text); do
  listener=$(aws elbv2 describe-listeners --region $REGION --load-balancer-arn $lb --query
'Listeners[?Port==`80`].ListenerArn | [0]' --output text)
  [ "$listener" = "None" ] && continue
  aws elbv2 modify-listener --region $REGION --listener-arn $listener \
    --default-actions
'Type=redirect,RedirectConfig={Protocol=HTTPS,Port=443,StatusCode=HTTP_301}'
done
aws elb describe-load-balancers --region $REGION --query 'LoadBalancerDescriptions[].
[LoadBalancerName,DNSName]' --output text
```

Web ACL with AWS managed rule groups, attached to one ALB (deploy as a stack set) (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Regional web ACL with AWS managed rule groups for public ALBs.
Parameters:
  LoadBalancerArn:
    Type: String
    Description: ARN of the internet-facing Application Load Balancer.
Resources:
  WebAcl:
    Type: AWS::WAFv2::WebACL
    Properties:
      Name: sf-public-alb
      Scope: REGIONAL
      DefaultAction: {Allow: {}}
      VisibilityConfig: {SampledRequestsEnabled: true, CloudWatchMetricsEnabled: true,
MetricName: sf-public-alb}
      Rules:
        - Name: AWSManagedRulesCommonRuleSet
          Priority: 10
          OverrideAction: {None: {}}
          Statement: {ManagedRuleGroupStatement: {VendorName: AWS, Name:
AWSManagedRulesCommonRuleSet}}
          VisibilityConfig: {SampledRequestsEnabled: true, CloudWatchMetricsEnabled: true,
MetricName: common}
        - Name: AWSManagedRulesKnownBadInputsRuleSet
          Priority: 20
          OverrideAction: {None: {}}
          Statement: {ManagedRuleGroupStatement: {VendorName: AWS, Name:
AWSManagedRulesKnownBadInputsRuleSet}}
          VisibilityConfig: {SampledRequestsEnabled: true, CloudWatchMetricsEnabled: true,
MetricName: bad-inputs}
        - Name: AWSManagedRulesAmazonIpReputationList
          Priority: 30
```

```
          OverrideAction: {None: {}}
          Statement: {ManagedRuleGroupStatement: {VendorName: AWS, Name:
AWSManagedRulesAmazonIpReputationList}}
          VisibilityConfig: {SampledRequestsEnabled: true, CloudWatchMetricsEnabled: true,
MetricName: ip-reputation}
  WebAclAssociation:
    Type: AWS::WAFv2::WebACLAssociation
    Properties:
      ResourceArn: !Ref LoadBalancerArn
      WebACLArn: !GetAtt WebAcl.Arn
```

ALB with the redirect in code (AWS CDK, TypeScript)

```
TYPESCRIPT
import { Stack, StackProps, aws_ec2 as ec2, aws_elasticloadbalancingv2 as elbv2 } from 'aws-
cdk-lib';
import { Construct } from 'constructs';

export class PublicAlbStack extends Stack {
  constructor(scope: Construct, id: string, props: StackProps & { vpc: ec2.IVpc }) {
    super(scope, id, props);
    const alb = new elbv2.ApplicationLoadBalancer(this, 'Alb', { vpc: props.vpc,
internetFacing: true, dropInvalidHeaderFields: true });
    alb.addRedirect({
      sourceProtocol: elbv2.ApplicationProtocol.HTTP, sourcePort: 80,
      targetProtocol: elbv2.ApplicationProtocol.HTTPS, targetPort: 443,
    });
    alb.addListener('Https', {
      port: 443,
      certificates: [elbv2.ListenerCertificate.fromArn('<acm-certificate-arn>')],
      defaultAction: elbv2.ListenerAction.fixedResponse(404),
    });
  }
}
```

How it is verified

- `curl -sI http://<alb-host>/` returns `301` with a `Location` header on https (Security Hub CSPM ELB.1).
- `aws wafv2 list-resources-for-web-acl --web-acl-arn <web-acl-arn> --resource-type APPLICATION_LOAD_BALANCER` lists every internet-facing ALB.
- `aws elb describe-load-balancers` returns no Classic Load Balancers in any Region.

How to roll it back. Set the HTTP listener's default action back to forward with `aws elbv2 modify-listener`, and detach the web ACL with `aws wafv2 disassociate-web-acl`, or switch its rule groups to count mode. Shift the weighted Route 53 records back to the Classic Load Balancer, which is kept for a day after cutover. Verify with the ALB's 5xx rate and the WAF logs.

SF-SEC-035: 22 database instances store data unencrypted

Medium. Benchmark rating Medium (Security Hub CSPM RDS.3). How it was rated: benchmark Medium (2) +0 exploitability (moderate) +0 blast radius (production) = 2

Where: 11 accounts in 3 Regions, 22 DB instances. **Controls:** `RDS.3` (RDS DB instances should have encryption at-rest enabled). **SOC 2:** CC6.1, C1.1. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 74 hours. **Planned:** Backlog, due November 20, 2026.

Evidence

- StorageEncrypted false; encryption cannot be turned on in place.

Examples: `db-mission-planner-cafdc5`, `db-ota-4b4f68`, `db-fleet-api-eu-286bf3`, `db-fleet-api-jp-b409e4`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Snapshots of unencrypted databases can be shared out. **Blast radius.** Production systems or customer data. Includes Aurora PostgreSQL writers for the Fleet API and the OTA service.

How we verified it. Read from the scan.

Remediation: Move every unencrypted database onto encrypted storage by snapshot, copy, and restore. Several RDS databases, including Aurora PostgreSQL writers, store data unencrypted (RDS.3), and encryption cannot be turned on in place. Snapshot each database, then encrypt it with a customer managed key: copy the snapshot for an RDS instance, or restore the snapshot directly for an Aurora cluster (AWS cannot encrypt an Aurora cluster snapshot in a copy). Cut the application over to the restored database.

Steps

1. Schedule a cutover window per database and tell the owning team; writes stop between the final snapshot and the cutover.
2. Create a customer managed KMS key for databases (rotation on) and allow the application roles to use it where they read snapshots.
3. Take a final snapshot. For an RDS instance, copy the snapshot with `--kms-key-id` and restore from the encrypted copy. For an Aurora cluster, do not copy: AWS cannot encrypt an unencrypted cluster snapshot in a copy, so restore the final cluster snapshot directly with `--kms-key-id`.
4. Keep the engine's default Extended Support setting when you restore a MySQL 5.7 or PostgreSQL 11–13 database; opting out on a database already past standard support forces a major version upgrade, so plan upgrades as their own change.

5. Point the application at the new endpoint (change the Route 53 CNAME or the secret), watch for 24 hours, then retire the old database after a final backup.

RDS instance: snapshot, copy encrypted, restore (AWS CLI)

```bash
DB=<unencrypted-db-instance-identifier>
aws rds create-db-snapshot --db-instance-identifier $DB --db-snapshot-identifier $DB-final
aws rds wait db-snapshot-available --db-snapshot-identifier $DB-final
aws rds copy-db-snapshot --source-db-snapshot-identifier $DB-final --target-db-snapshot-identifier $DB-encrypted --kms-key-id alias/sf-rds
aws rds wait db-snapshot-available --db-snapshot-identifier $DB-encrypted
aws rds restore-db-instance-from-db-snapshot --db-instance-identifier $DB-enc --db-snapshot-identifier $DB-encrypted \
  --db-subnet-group-name <db-subnet-group-name> --vpc-security-group-ids <security-group-id> --no-publicly-accessible
```

Aurora cluster: snapshot, restore encrypted from the unencrypted snapshot, add the writer (AWS CLI)

```bash
CLUSTER=<unencrypted-aurora-cluster-identifier>
aws rds create-db-cluster-snapshot --db-cluster-identifier $CLUSTER --db-cluster-snapshot-identifier $CLUSTER-final
aws rds wait db-cluster-snapshot-available --db-cluster-snapshot-identifier $CLUSTER-final
# Aurora cannot copy an unencrypted cluster snapshot into an encrypted one; the restore is what applies the key.
aws rds restore-db-cluster-from-snapshot --db-cluster-identifier $CLUSTER-enc --snapshot-identifier $CLUSTER-final \
  --engine aurora-postgresql --kms-key-id alias/sf-rds --db-subnet-group-name <db-subnet-group-name> --vpc-security-group-ids <security-group-id>
aws rds create-db-instance --db-cluster-identifier $CLUSTER-enc --db-instance-identifier $CLUSTER-enc-writer \
  --db-instance-class <db-instance-class> --engine aurora-postgresql
```

The same Aurora restore in the CDK app (use it instead of the CLI restore, not in addition to it) (AWS CDK, TypeScript)

```typescript
import { Stack, StackProps, aws_ec2 as ec2, aws_kms as kms, aws_rds as rds } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class FleetDatabaseStack extends Stack {
  constructor(scope: Construct, id: string, props: StackProps & { vpc: ec2.IVpc }) {
    super(scope, id, props);
    new rds.DatabaseClusterFromSnapshot(this, 'Cluster', {
      engine: rds.DatabaseClusterEngine.auroraPostgres({ version: rds.AuroraPostgresEngineVersion.of('<aurora-postgres-full-version>', '<aurora-postgres-major-version>') }),
      snapshotIdentifier: '<cluster-snapshot-identifier>',
      storageEncrypted: true,
      storageEncryptionKey: kms.Key.fromKeyArn(this, 'DatabaseKey', '<rds-kms-key-arn>'),
      writer: rds.ClusterInstance.provisioned('Writer', { instanceType: new ec2.InstanceType('<db-instance-class>') }),
      vpc: props.vpc,
      deletionProtection: true,
    });
  }
}
```

How it is verified

- `aws rds describe-db-instances --query 'DBInstances[?StorageEncrypted==false].DBInstanceIdentifier'` returns an empty list in each account (Security Hub CSPM RDS.3).
- `aws rds describe-db-clusters` shows `StorageEncrypted` true and the customer managed `KmsKeyId` for every Aurora cluster.
- The application's error rate and latency are unchanged for 24 hours after cutover.

How to roll it back. Point the application back at the old endpoint by reverting the Route 53 CNAME or the secret; the old database stays until the final backup in step 5. Writes made to the new database after cutover must be exported and replayed on the old one, so roll back inside the first hour or accept the loss. Keep the final snapshot. Verify by reading the application's error rate.

SF-SEC-036: 9 ElastiCache replication groups lack encryption at rest or in transit

Medium. Benchmark rating Medium (Vantalect baseline (no Security Hub CSPM control in the verified set)). How it was rated: benchmark Medium (2) +0 exploitability (moderate) +0 blast radius (production) = 2

Where: `fleet-api-prod` (…5044), `fleet-eu-prod` (…5160), `fleet-staging` (…6014) in 3 Regions, 9 clusters and domains. **Controls:** No benchmark control covers this. **SOC 2:** CC6.1, CC6.7, C1.1. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 33 hours. **Planned:** Backlog, due November 20, 2026.

Evidence

- 7 without in-transit encryption; 3 without at-rest encryption.

Examples: `fleet-api-cache-8f78c`, `fleet-api-cache-bc196`, `fleet-api-cache-116fa`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Anyone inside the VPC can read cache traffic in clear text. **Blast radius.** Production systems or customer data. Fleet API session and mission caches.

How we verified it. Read from the scan.

Remediation: Rebuild ElastiCache groups with encryption, upgrade them to Valkey, and enable fine-grained access on OpenSearch. ElastiCache replication groups lack at-rest and in-transit encryption, and one OpenSearch domain has no fine-grained access control. Encryption at rest cannot be added to an existing group, so restore a backup into a new encrypted group, then upgrade it in place to Valkey; turn on fine-grained access control for the domain.

Steps

1. Take a manual backup of each replication group and create a new group from it with at-rest and in-transit encryption and a customer managed key.
2. Update clients to use TLS endpoints and test them against the new group before cutover; the endpoint changes, so update the application configuration.
3. Cut over, then upgrade the new group in place from Redis OSS to Valkey; endpoints stay the same and downtime is a short failover.
4. On the OpenSearch domain, confirm node-to-node encryption, encryption at rest, and HTTPS are on, then enable fine-grained access control with an IAM master role.
5. Replace the open domain access policy with one that names the application roles, and map those roles in OpenSearch security.

ElastiCache: backup, encrypted group from backup, then Valkey in place (AWS CLI)

```bash
OLD=<old-replication-group-id>
NEW=<new-replication-group-id>
aws elasticache create-snapshot --replication-group-id $OLD --snapshot-name $OLD-pre-encryption
aws elasticache create-replication-group --replication-group-id $NEW --replication-group-
description "$NEW encrypted" \
  --snapshot-name $OLD-pre-encryption --cache-node-type <cache-node-type> --num-cache-clusters
2 \
  --automatic-failover-enabled --multi-az-enabled --at-rest-encryption-enabled --transit-
encryption-enabled \
  --kms-key-id <kms-key-arn> --cache-subnet-group-name <cache-subnet-group-name> --security-
group-ids <security-group-id>
aws elasticache wait replication-group-available --replication-group-id $NEW
# After cutover, upgrade in place (add --cache-parameter-group-name <valkey-parameter-group> if
a custom group is attached):
aws elasticache modify-replication-group --replication-group-id $NEW --engine valkey --engine-
version 9.0 --apply-immediately
```

OpenSearch: enable fine-grained access control with an IAM master role (AWS CLI)

```bash
aws opensearch describe-domain --domain-name <domain-name> \
  --query 'DomainStatus.
[NodeToNodeEncryptionOptions.Enabled,EncryptionAtRestOptions.Enabled,DomainEndpointOptions.Enfo
rceHTTPS]'
aws opensearch update-domain-config --domain-name <domain-name> --advanced-security-options '{
  "Enabled": true,
  "InternalUserDatabaseEnabled": false,
  "MasterUserOptions": {"MasterUserARN": "arn:aws:iam::<fleet-analytics-prod-account-id>:role/
<opensearch-master-role-name>"}
}'
aws opensearch describe-domain-config --domain-name <domain-name> --query
'DomainConfig.AdvancedSecurityOptions.Status.State'
```

How it is verified

- `aws elasticache describe-replication-groups` shows `AtRestEncryptionEnabled` and `TransitEncryptionEnabled` true and engine `valkey` for every group.
- The old groups carry no traffic for 24 hours before they are deleted.
- `describe-domain` shows `AdvancedSecurityOptions.Enabled` true, and an unsigned request to the domain is rejected.

How to roll it back. Point clients back at the old replication group's endpoint, which is kept until the cutover is accepted. The Valkey upgrade cannot be reversed in place; restore from the backup taken in step 1 into a new Redis OSS group instead. Fine-grained access control on OpenSearch cannot be turned off once on, so restore the earlier domain access policy and map the roles in OpenSearch security. Verify with a client connection test.

SF-SEC-037: 5 EFS file systems are not encrypted at rest

Medium. Benchmark rating Medium (Security Hub CSPM EFS.1 and EFS.8). How it was rated: benchmark Medium (2) +0 exploitability (moderate) +0 blast radius (production) = 2

Where: 5 accounts in us-east-1, 5 file systems. **Controls:** `EFS.1` (Elastic File System should be configured to encrypt file data at-rest using AWS KMS); `EFS.8` (EFS file systems should be encrypted at rest). **SOC 2:** CC6.1, C1.1. **Owner:** Hiro Tanaka, Director, Crucible Platform. **Repository:** `crucible-infra`. **Effort:** 12 hours. **Planned:** Backlog, due November 20, 2026.

Evidence

- Encryption at rest can only be chosen when a file system is created.

Examples: `fs-28d55d83b768295ba`, `fs-5bc14dc41f1e09c2a`, `fs-704b4ebcdd219d132`, `fs-9f2b462a7e2a76f5d`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Backups and replicas inherit the gap. **Blast radius.** Production systems or customer data. Includes Kestrel model artifacts and researcher home directories.

How we verified it. Read from the scan.

Remediation: Copy unencrypted EFS file systems into new encrypted ones with DataSync. Unencrypted EFS file systems (EFS.1, EFS.8) cannot be encrypted in place. Create an encrypted file system with a customer managed key and a TLS-only policy, copy the data with DataSync, and switch the mounts over.

Steps

1. Create the encrypted file system in CDK with automatic backups and the TLS-only file system policy.
2. Create DataSync locations for the old and new file systems and a task that preserves ownership and permissions.
3. Run the task once while the workload is live, then stop writers, run it again for the last changes, and verify counts and checksums.
4. Update the mounts (ECS task definitions, EKS persistent volumes, fstab entries) to the new file system, using the TLS mount option.
5. Keep the old file system for two weeks, then take a final AWS Backup recovery point and delete it.

Encrypted file system with backups (AWS CDK, TypeScript)

```typescript
import { RemovalPolicy, Stack, StackProps, aws_ec2 as ec2, aws_efs as efs, aws_kms as kms }
from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class SimulationStorageStack extends Stack {
  constructor(scope: Construct, id: string, props: StackProps & { vpc: ec2.IVpc }) {
    super(scope, id, props);
    const key = new kms.Key(this, 'EfsKey', { enableKeyRotation: true });
    new efs.FileSystem(this, 'SimulationData', {
      vpc: props.vpc,
      encrypted: true,
      kmsKey: key,
      enableAutomaticBackups: true,
      allowAnonymousAccess: false,
      removalPolicy: RemovalPolicy.RETAIN,
    });
  }
}
```

EFS file system policy: refuse mounts that do not use TLS (IAM policy)

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyNonTlsMounts",
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": "*",
      "Condition": {
        "Bool": {
          "aws:SecureTransport": "false"
        }
      }
    }
  ]
}
```

DataSync copy from the old file system to the new one (AWS CLI)

```bash
SRC=$(aws datasync create-location-efs --efs-filesystem-arn arn:aws:elasticfilesystem:us-east-1:<crucible-account-id>:file-system/<old-file-system-id> \
  --ec2-config SubnetArn=<subnet-arn>,SecurityGroupArns=<security-group-arn> --query LocationArn --output text)
DST=$(aws datasync create-location-efs --efs-filesystem-arn arn:aws:elasticfilesystem:us-east-1:<crucible-account-id>:file-system/<new-file-system-id> \
  --ec2-config SubnetArn=<subnet-arn>,SecurityGroupArns=<security-group-arn> --in-transit-encryption TLS1_2 --query LocationArn --output text)
TASK=$(aws datasync create-task --source-location-arn "$SRC" --destination-location-arn "$DST" --name efs-to-encrypted \
  --options VerifyMode=POINT_IN_TIME_CONSISTENT,Uid=INT_VALUE,Gid=INT_VALUE,PreserveDeletedFiles=REMOVE --query TaskArn --output text)
aws datasync start-task-execution --task-arn "$TASK"
```

How it is verified

- `aws efs describe-file-systems --query 'FileSystems[?Encrypted==false]'` returns an empty list (Security Hub CSPM EFS.1, EFS.8).
- The DataSync task reports zero files with verification errors and the file counts match.
- The workload runs against the new file system for two weeks with no mount errors.

How to roll it back. Switch the mounts back to the old file system, which is kept for two weeks, and stop the DataSync task. Data written to the new file system after cutover must be copied back with a reverse DataSync task. Delete the new file system only after that copy. Verify with file counts and a checksum sample.

SF-SEC-038: Security Hub CSPM runs in only 3 Regions, with FSBP only and no CIS standard

Medium. Benchmark rating Medium (Vantalect baseline: posture monitoring). How it was rated: benchmark Medium (2) -1 exploitability (limited) +1 blast radius (organization) = 2

Where: All 48 accounts in 17 Regions, 687 account-Regions. **Controls:** No benchmark control covers this. **SOC 2:** CC2.2, CC4.1, CC7.1, CC7.3, CC7.4. **Owner:** Nadia Petrova, Manager, Security Operations. **Repository:** `platform-landing-zone`. **Effort:** 8 hours. **Planned:** Week 5, starting Aug 10. **Depends on:** SF-SEC-022.

Evidence

- Kestrel accounts are not members.
- Findings are not routed to on-call; a Slack channel receives some GuardDuty findings.

Examples: `us-east-1, us-west-2, eu-central-1 only`, `no finding aggregation Region`.

Exploitability. Needs a foothold inside, an insider, or physical access. Failed controls go unnoticed between audits. **Blast radius.** Every account in the organization. Every account and Region.

How we verified it. Read from describe-hub and get-enabled-standards.

Remediation: Enable Security Hub CSPM centrally with FSBP and CIS v5.0.0 and route serious findings to PagerDuty. AWS Security Hub CSPM runs in a few Regions with only the AWS Foundational Security Best Practices standard. Use central configuration from sf-security-tooling to enable FSBP v1.0.0 and CIS v5.0.0 everywhere, aggregate findings in us-east-1, and send CRITICAL and HIGH findings to PagerDuty.

Steps

1. Make sf-security-tooling the Security Hub CSPM delegated administrator, turn Security Hub CSPM on there without the default standards, and create a finding aggregator in us-east-1 that links all Regions; central configuration needs both.
2. Switch the organization to central configuration and stop per-account auto-enable.
3. Create one configuration policy that enables FSBP v1.0.0 and CIS v5.0.0 and associate it with the organization root.
4. Disable EC2.172 by agreement, add port 3478 to EC2.18's authorized TCP and UDP ports, and set Config.1's `includeConfigServiceLinkedRoleCheck` to false, because Control Tower uses its own recorder role.
5. Create the EventBridge rule for new, failed CRITICAL and HIGH findings that Security Hub CSPM itself generated, and point it at the PagerDuty topic. A finding stays NEW until someone triages it, so an update to an untriaged finding pages again; the on-call sets it to NOTIFIED when acknowledging the page, which ends the repeats.

Where Security Hub CSPM stands today (read-only, sf-security-tooling) (AWS CLI)

```bash
aws securityhub describe-organization-configuration --profile <security-tooling-profile> --region us-east-1
aws securityhub list-finding-aggregators --profile <security-tooling-profile> --region us-east-1
```

Delegated administrator, aggregator, central configuration, and the sf-baseline policy (AWS CLI)

```bash
REGIONS="us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-west-1 eu-west-2 eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-southeast-1 ap-southeast-2 ap-south-1"
```

```
# 1. Management account: make sf-security-tooling the administrator from the home Region.
aws securityhub enable-organization-admin-account --profile <management-profile> --region us-
east-1 --admin-account-id <security-tooling-account-id>

# 2. sf-security-tooling: turn Security Hub CSPM on without the default standards, aggregate
every Region into us-east-1,
#    and switch the organization to central configuration (it needs auto-enable off).
aws securityhub enable-security-hub --profile <security-tooling-profile> --region us-east-1 --
no-enable-default-standards
aws securityhub create-finding-aggregator --profile <security-tooling-profile> --region us-
east-1 --region-linking-mode ALL_REGIONS
aws securityhub update-organization-configuration --profile <security-tooling-profile> --region
us-east-1 --no-auto-enable --auto-enable-standards NONE \
  --organization-configuration ConfigurationType=CENTRAL

# 3. The configuration policy. Integer lists are plain numbers; EC2.18 takes TCP and UDP port
lists, and TURN relays use 3478 on both.
cat > sf-baseline-policy.json <<'JSON'
{"SecurityHub": {
  "ServiceEnabled": true,
  "EnabledStandardIdentifiers": [
    "arn:aws:securityhub:us-east-1::standards/aws-foundational-security-best-practices/v/
1.0.0",
    "arn:aws:securityhub:us-east-1::standards/cis-aws-foundations-benchmark/v/5.0.0"],
  "SecurityControlsConfiguration": {
    "DisabledSecurityControlIdentifiers": ["EC2.172"],
    "SecurityControlCustomParameters": [
      {"SecurityControlId": "EC2.18", "Parameters": {
        "authorizedTcpPorts": {"ValueType": "CUSTOM", "Value": {"IntegerList": [80, 443,
3478]}},
        "authorizedUdpPorts": {"ValueType": "CUSTOM", "Value": {"IntegerList": [3478]}}}},
      {"SecurityControlId": "Config.1", "Parameters": {"includeConfigServiceLinkedRoleCheck":
{"ValueType": "CUSTOM", "Value": {"Boolean": false}}}}]}}}
JSON
POLICY=$(aws securityhub create-configuration-policy --profile <security-tooling-profile> --
region us-east-1 --name sf-baseline \
  --configuration-policy file://sf-baseline-policy.json --query Id --output text)
aws securityhub start-configuration-policy-association --profile <security-tooling-profile> --
region us-east-1 \
  --configuration-policy-identifier "$POLICY" --target RootId=<org-root-id>

# 4. Every linked Region must list sf-security-tooling as the administrator. Where one does
not, run
#    enable-organization-admin-account in that Region too before you trust the coverage
numbers.
for region in $REGIONS; do
  echo "$region: $(aws securityhub list-organization-admin-accounts --profile <management-
profile> --region $region --query 'AdminAccounts[].[AccountId,Status]' --output text)"
done
```

New, failed CRITICAL and HIGH findings that Security Hub CSPM itself generated (EventBridge event pattern)

```
JSON
```

```
{
  "source": [
    "aws.securityhub"
  ],
  "detail-type": [
    "Security Hub Findings - Imported"
  ],
  "detail": {
    "findings": {
      "ProductName": [
        "Security Hub"
      ],
      "Severity": {
        "Label": [
          "CRITICAL",
          "HIGH"
        ]
      },
      "Compliance": {
        "Status": [
          "FAILED"
        ]
      },
      "Workflow": {
        "Status": [
          "NEW"
        ]
      },
      "RecordState": [
        "ACTIVE"
      ]
    }
  }
}
```

Send matching findings to the PagerDuty topic (AWS CLI)

```bash
aws events put-rule --profile <security-tooling-profile> --region us-east-1 --name sf-securityhub-critical-high --state ENABLED \
  --event-pattern file://securityhub-critical-high-pattern.json
aws events put-targets --profile <security-tooling-profile> --region us-east-1 --rule sf-securityhub-critical-high \
  --targets Id=PagerDuty,Arn=arn:aws:sns:us-east-1:<security-tooling-account-id>:sf-securityhub-pagerduty
```

How it is verified

- `aws securityhub describe-organization-configuration` shows `ConfigurationType` CENTRAL, and `list-configuration-policy-associations` shows the root associated with `sf-baseline`.
- FSBP v1.0.0 and CIS v5.0.0 show as enabled in every account and Region, and findings aggregate in us-east-1.

- A test CRITICAL finding created with `batch-import-findings` pages the on-call engineer.

How to roll it back. Disassociate the configuration policy from the root with `aws securityhub start-configuration-policy-disassociation`, and disable a standard or a control in the policy. Switching back to local configuration is one call to `aws securityhub update-organization-configuration` and re-enables per-account settings. Remove the EventBridge rule to stop the pages. Verify by reading the policy association.

SF-SEC-039: 7 accounts are outside Control Tower, and the landing zone has drifted

Medium. Benchmark rating Medium (Vantalect baseline: account governance). How it was rated: benchmark Medium (2) -1 exploitability (limited) +1 blast radius (organization) = 2

Where: 7 accounts in 6 Regions, 7 accounts. **Controls:** No benchmark control covers this. **SOC 2:** CC1.3, CC3.4, CC5.2. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 20 hours. **Planned:** Week 5, starting Aug 10.

Evidence

- The landing zone's guardrail SCPs were detached from all six registered OUs on 03/14/2023 to unblock a failed pipeline deployment and were never reattached.
- sandbox-hackathon was created in the Organizations console in April 2024 and placed in Sandbox outside Account Factory.

Examples: `kestrel-prod (…9031)`, `kestrel-staging (…9054)`, `kestrel-dev (…9076)`, `kestrel-data (…9098)`.

Exploitability. Needs a foothold inside, an insider, or physical access. New and acquired accounts miss the baseline. **Blast radius.** Every account in the organization. Five Kestrel accounts, a sandbox, and a suspended account.

How we verified it. Read the Control Tower drift report on 07/20/2026.

Remediation: Repair the Control Tower landing zone and enroll the unmanaged accounts. The 2021 Control Tower landing zone has drifted: guardrail SCPs were detached in 2023, some accounts were moved outside Control Tower, and others were never enrolled (the Kestrel accounts, sandbox-hackathon, and suspended-legacy-ci). Add the execution role to each account, move accounts back, repair the landing zone, re-register the OUs, and enroll.

Steps

1. Read the landing zone status and drift report to list what Control Tower thinks is wrong.

2. Deploy the `AWSControlTowerExecution` role to every unenrolled account; it trusts only the management account.
3. Move the accounts that left Control Tower back into their registered OUs, and move the unenrolled accounts into registered OUs.
4. Run a landing zone repair (`reset-landing-zone`) and wait for it to finish; re-register each OU so detached guardrail SCPs come back.
5. Enroll each remaining account and watch its enrollment operation; enroll `suspended-legacy-ci` only if it will be kept, or close it instead.

AWSControlTowerExecution role for an unenrolled account (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Role Control Tower assumes to enroll this account.
Parameters:
  ManagementAccountId: {Type: String, Description: Management account ID (<management-account-
id>).}
Resources:
  ControlTowerExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: AWSControlTowerExecution
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AdministratorAccess
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal: {AWS: !Sub 'arn:aws:iam::${ManagementAccountId}:root'}
            Action: 'sts:AssumeRole'
```

Status, role deployment, account moves, and landing zone repair (AWS CLI)

```bash
LZ=$(aws controltower list-landing-zones --query 'landingZones[0].arn' --output text)
aws controltower get-landing-zone --landing-zone-identifier "$LZ" --query 'landingZone.
[status,driftStatus.status,version,latestAvailableVersion]'

# In each unenrolled account (profile <account-profile>):
aws cloudformation deploy --profile <account-profile> --region us-east-1 --stack-name sf-
control-tower-execution \
  --template-file control-tower-execution-role.yaml --capabilities CAPABILITY_NAMED_IAM \
  --parameter-overrides ManagementAccountId=<management-account-id>

aws organizations move-account --account-id <account-id> --source-parent-id <current-parent-id>
--destination-parent-id <registered-ou-id>

aws controltower reset-landing-zone --landing-zone-identifier "$LZ"
aws controltower get-landing-zone-operation --operation-identifier <operation-identifier> --
query 'operationDetails.status'
```

Re-register an OU and enroll an account (AWS CLI)

```bash
aws controltower list-enabled-baselines --query 'enabledBaselines[].
[arn,targetIdentifier,statusSummary.status]' --output table
BASELINE=$(aws controltower list-baselines --query "baselines[?
name=='AWSControlTowerBaseline'].arn" --output text)
# <identity-center-enabled-baseline-arn>: the IdentityCenterBaseline row from list-enabled-
baselines above.
# Registering the Kestrel OU enrolls every account in it.
aws controltower enable-baseline \
  --baseline-identifier "$BASELINE" --baseline-version <baseline-version> \
  --target-identifier arn:aws:organizations::<management-account-id>:ou/<org-id>/<kestrel-ou-
id> \
  --parameters "[{\"key\": \"IdentityCenterEnabledBaselineArn\", \"value\": \"<identity-center-
enabled-baseline-arn>\"}]"
aws controltower get-baseline-operation --operation-identifier <operation-identifier> --query
'baselineOperation.status'
```

How it is verified

- `aws controltower get-landing-zone` shows status ACTIVE, drift status IN_SYNC, and the latest available version.
- `aws controltower list-enabled-baselines` shows every account and OU as SUCCEEDED, and the guardrail SCPs are attached to each registered OU.
- None of the previously unenrolled accounts remain outside a registered OU, and each shows as enrolled in the Control Tower console.

How to roll it back. A landing zone repair cannot be undone, and the guardrail SCPs it restores should stay. To take an enrolled account back out of Control Tower management, disable its baseline with `aws controltower disable-baseline`. Move an account back to its earlier OU with `aws organizations move-account`. Delete the `AWSControlTowerExecution` role only after the account is unmanaged. Verify with the landing zone drift status.

SF-SEC-040: Break-glass access is not alarmed and was last tested in April 2025

Medium. Benchmark rating Medium (Vantalect baseline: emergency access). How it was rated: benchmark Medium (2) -1 exploitability (limited) +1 blast radius (organization) = 2

Where: `sf-management` (…0007) in us-east-1, 3 emergency identities. **Controls:** No benchmark control covers this. **SOC 2:** CC6.1, CC7.2. **Owner:** Nadia Petrova, Manager, Security Operations. **Repository:** `platform-landing-zone`. **Effort:** 8 hours. **Planned:** Week 4, starting Aug 3.

Evidence

- No EventBridge rule or alarm on ConsoleLogin by the root user or the break-glass users.
- Last documented test: 04/02/2025.

Examples: `sf-management root`, `breakglass-01`, `breakglass-02`.

Exploitability. Needs a foothold inside, an insider, or physical access. Misuse needs the sealed credentials or the root mailbox. **Blast radius.** Every account in the organization. The management account's root and two break-glass users administer the organization.

How we verified it. Reviewed the break-glass procedure with Nadia Petrova on 07/22/2026.

Remediation: Alarm every root and break-glass sign-in in all 17 Regions and test it each quarter. The management root (hardware MFA in a safe) and `breakglass-01` and `breakglass-02` are the last-resort identities, but nothing pages anyone when they are used. An EventBridge rule on console sign-in sends them to SNS and PagerDuty from every Region, and a quarterly test proves the path works.

Steps

1. Create the stack set below in the management account and deploy it to all 17 Regions.
2. Create the PagerDuty service with an HTTPS integration and pass its URL as the template parameter.
3. Confirm the SNS subscription from PagerDuty and send a test message.
4. Once a quarter, with two people present, sign in as `breakglass-01`, confirm the page arrives within five minutes, sign out, and record the result.

5. Rotate the break-glass passwords after every real use and every test.

Alarm template for the stack set (one instance per Region) (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Page on root and break-glass console sign-in.
Parameters:
  PagerDutyEndpoint:
    Type: String
    Description: The PagerDuty integration URL (<pagerduty-eventbridge-integration-url>).
Resources:
  AlarmKey:
    Type: AWS::KMS::Key
    Properties:
      EnableKeyRotation: true
      KeyPolicy:
        Version: '2012-10-17'
        Statement:
          - Sid: AccountAdministration
            Effect: Allow
            Principal: {AWS: !Sub 'arn:aws:iam::${AWS::AccountId}:root'}
            Action: 'kms:*'
            Resource: '*'
          - Sid: EventBridgePublishesToEncryptedTopic
            Effect: Allow
            Principal: {Service: events.amazonaws.com}
            Action: ['kms:Decrypt', 'kms:GenerateDataKey*']
            Resource: '*'
  AlarmTopic:
    Type: AWS::SNS::Topic
    Properties:
      TopicName: sf-breakglass-alarm
      KmsMasterKeyId: !Ref AlarmKey
  AlarmTopicPolicy:
    Type: AWS::SNS::TopicPolicy
    Properties:
      Topics: [!Ref AlarmTopic]
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal: {Service: events.amazonaws.com}
            Action: 'sns:Publish'
            Resource: !Ref AlarmTopic
  PagerDutySubscription:
    Type: AWS::SNS::Subscription
    Properties:
      TopicArn: !Ref AlarmTopic
      Protocol: https
      Endpoint: !Ref PagerDutyEndpoint
  SignInRule:
    Type: AWS::Events::Rule
    Properties:
      Name: sf-root-and-breakglass-signin
      State: ENABLED
```

```
      EventPattern:
        source: [aws.signin]
        detail-type: [AWS Console Sign In via CloudTrail]
        $or:
          - detail: {userIdentity: {type: [Root]}}
          - detail: {userIdentity: {userName: [breakglass-01, breakglass-02]}}
      Targets:
        - Id: PageSecurity
          Arn: !Ref AlarmTopic
```

The same pattern, for a test with `aws events test-event-pattern` (EventBridge event pattern)

```JSON
{
  "source": [
    "aws.signin"
  ],
  "detail-type": [
    "AWS Console Sign In via CloudTrail"
  ],
  "$or": [
    {
      "detail": {
        "userIdentity": {
          "type": [
            "Root"
          ]
        }
      }
    },
    {
      "detail": {
        "userIdentity": {
          "userName": [
            "breakglass-01",
            "breakglass-02"
          ]
        }
      }
    }
  ]
}
```

Deploy to all 17 Regions from the management account (AWS CLI)

```bash
aws cloudformation create-stack-set --stack-set-name sf-breakglass-alarm --template-body
file://breakglass-alarm.yaml \
  --capabilities CAPABILITY_IAM \
  --parameters ParameterKey=PagerDutyEndpoint,ParameterValue=<pagerduty-eventbridge-
integration-url> \
  --permission-model SELF_MANAGED \
  --administration-role-arn arn:aws:iam::<management-account-id>:role/
AWSCloudFormationStackSetAdministrationRole \
  --execution-role-name AWSCloudFormationStackSetExecutionRole
aws cloudformation create-stack-instances --stack-set-name sf-breakglass-alarm --accounts
<management-account-id> \
  --regions us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-
west-1 eu-west-2 eu-west-3 \
            eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-southeast-1 ap-
southeast-2 ap-south-1 \
  --operation-preferences MaxConcurrentCount=5,FailureToleranceCount=1
```

How it is verified

- `aws cloudformation list-stack-instances --stack-set-name sf-breakglass-alarm` shows CURRENT instances in all 17 Regions.
- The quarterly sign-in as `breakglass-01` pages the on-call engineer within five minutes, and the page time is logged with the test.
- `aws events test-event-pattern` returns true for a sample Root sign-in event and false for an ordinary user.

How to roll it back. Delete the stack set instances for the alarm template, or revert the stack set in `platform-landing-zone`, which removes the EventBridge rules, the SNS topics, and the PagerDuty subscription. Rolling back leaves root and break-glass sign-ins unmonitored, so do it only to replace the integration, and keep one Region's alarm running meanwhile. Verify with a quarterly test sign-in and a page within five minutes.

SF-SEC-041: SSH and RDP are open to the internet on 27 sandbox instances

Medium. Benchmark rating Critical (Security Hub CSPM EC2.19, EC2.53, EC2.54, and EC2.18). How it was rated: benchmark Critical (4) +0 exploitability (moderate) -1 blast radius (contained) = 3; rule 3: theoretical exploit or contained blast radius; capped at Medium

Where: `sandbox-platform` (…4011), `sandbox-hackathon` (…4078) in us-east-1, 3 security groups. **Controls:** `EC2.18` (Security groups should only allow unrestricted incoming traffic for authorized ports); `EC2.19` (Security groups should not allow unrestricted access to ports with high risk); `EC2.53` (EC2 security groups should not allow ingress from 0.0.0.0/0 to remote server

administration ports); `EC2.54` (EC2 security groups should not allow ingress from ::/0 to remote server administration ports). **SOC 2:** CC6.6. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 10 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- 3 security groups admit 0.0.0.0/0 on port 22, 80, and 443; 27 instances attached.
- 1 also admits ::/0.

Examples: `sg-7be9797e12be88753`, `sg-e613674bd82574a85`, `sg-e79a5c4cf81a10069`, `i-010042e56ca498c2a`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). Default images with shared keys. **Blast radius.** A sandbox with no production data. Sandbox accounts with no production data.

How we verified it. Matched each rule to its owner and use with the account owner.

Remediation: Close SSH and RDP to the internet and use Session Manager instead. This is the same change as SF-SEC-009, applied to the accounts listed above. Use the steps, the code, the verification, and the rollback under SF-SEC-009, with this finding's accounts in place of that finding's.

SF-SEC-042: Amazon Inspector is off almost everywhere; 132 ECR repositories have no image scanning

Medium. Benchmark rating High (Security Hub CSPM Inspector.1–4 and ECR.1). How it was rated: benchmark High (3) -1 exploitability (limited) +0 blast radius (production) = 2

Where: All 48 accounts in 17 Regions, 3,380 failed checks. **Controls:** `ECR.1` (ECR private repositories should have image scanning configured); `Inspector.1` (Amazon Inspector EC2 scanning should be enabled); `Inspector.2` (Amazon Inspector ECR scanning should be enabled); `Inspector.3` (Amazon Inspector Lambda code scanning should be enabled); `Inspector.4` (Amazon Inspector Lambda standard scanning should be enabled). **SOC 2:** CC6.8, CC7.1. **Owner:** Nadia Petrova, Manager, Security Operations. **Repository:** `platform-landing-zone`. **Effort:** 8 hours. **Planned:** Week 5, starting Aug 10.

Evidence

- Inspector EC2 and ECR scanning only in fleet production accounts in us-east-1.

Examples: `sf-cicd/repo-007`, `sf-cicd/repo-010`, `sf-cicd/repo-012`.

Exploitability. Needs a foothold inside, an insider, or physical access. Known vulnerabilities in images, hosts, and functions go unreported. **Blast radius.** Production systems or customer data. Container images every workload deploys.

How we verified it. Read from the scan.

Remediation: Turn on Amazon Inspector for EC2, ECR, Lambda, and Lambda code across the organization. Amazon Inspector is off (Inspector.1 to Inspector.4) and ECR repositories have no scanning (ECR.1). Make sf-security-tooling the delegated administrator, auto-enable every scan type for all members, and switch ECR to continuous enhanced scanning.

Steps

1. From the management account, designate sf-security-tooling as the Inspector delegated administrator in each of the 17 Regions; the designation is per Region.
2. From sf-security-tooling, run `inspector2 enable` for every active account in each Region (auto-enable reaches only accounts that join later), then turn on auto-enable for EC2, ECR, Lambda, and Lambda code, and list any account that is not fully enabled.
3. Make sure instances are managed by Systems Manager (see ssm-patching), because agent-based EC2 scanning needs the agent; agentless scanning covers instances without it.
4. Set the ECR registry scanning configuration to enhanced, continuous scanning for all repositories in every account.
5. Route CRITICAL findings to the security queue from every Region (the routing script forwards each Region to us-east-1), and give each team a weekly list of findings by repository.

Delegate in every Region and activate all accounts (management account, then sf-security-tooling) (AWS CLI)

```bash
REGIONS="us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-west-1
eu-west-2 eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-southeast-1 ap-
southeast-2 ap-south-1"

# 1. Management account: trusted access, the active accounts, and the delegated administrator
in EVERY Region.
#    Inspector's delegated administrator is set per Region.
aws organizations enable-aws-service-access --profile <management-profile> --service-principal
inspector2.amazonaws.com
aws organizations list-accounts --profile <management-profile> --query 'Accounts[?
State==`ACTIVE`].Id' --output text > active-accounts.txt
for region in $REGIONS; do
  aws inspector2 enable-delegated-admin-account --profile <management-profile> --region $region
--delegated-admin-account-id <security-tooling-account-id>
done

# 2. sf-security-tooling: activate every existing account in every Region (auto-enable reaches
only accounts that join later),
#    turn on auto-enable for new members, and list any account that is not fully enabled.
for region in $REGIONS; do
  aws inspector2 enable --profile <security-tooling-profile> --region $region --account-ids $
(cat active-accounts.txt) \
    --resource-types EC2 ECR LAMBDA LAMBDA_CODE
  aws inspector2 update-organization-configuration --profile <security-tooling-profile> --
region $region \
    --auto-enable ec2=true,ecr=true,lambda=true,lambdaCode=true
  not_enabled=$(aws inspector2 batch-get-account-status --profile <security-tooling-profile> --
region $region --account-ids $(cat active-accounts.txt) \
    --query 'accounts[?state.status!=`ENABLED`].accountId' --output text)
  [ -z "$not_enabled" ] || echo "CHECK $region: not fully enabled yet: $not_enabled"
done
```

Continuous enhanced scanning for ECR (run in each account and Region) (AWS CLI)

```bash
for region in us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-
west-1 eu-west-2 eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-
southeast-1 ap-southeast-2 ap-south-1; do
  aws ecr put-registry-scanning-configuration --region $region --scan-type ENHANCED \
    --rules '[{"scanFrequency": "CONTINUOUS_SCAN", "repositoryFilters": [{"filter": "*",
"filterType": "WILDCARD"}]}]'
  aws ecr get-registry-scanning-configuration --region $region --query
scanningConfiguration.scanType --output text
done
```

Critical Inspector findings (EventBridge event pattern)

```JSON
{
  "source": [
    "aws.inspector2"
  ],
  "detail-type": [
    "Inspector2 Finding"
  ],
  "detail": {
    "severity": [
      "CRITICAL"
    ],
    "status": [
      "ACTIVE"
    ]
  }
}
```

Route Inspector findings from every Region to the on-call (sf-security-tooling) (AWS CLI)

```bash
REGIONS="us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-west-1
eu-west-2 eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-southeast-1 ap-
southeast-2 ap-south-1"
BUS=arn:aws:events:us-east-1:<security-tooling-account-id>:event-bus/default
FORWARD_ROLE=arn:aws:iam::<security-tooling-account-id>:role/sf-events-forward

# Once: the role EventBridge uses to put events on the us-east-1 bus.
aws iam create-role --role-name sf-events-forward --assume-role-policy-document \
  '{"Version": "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
"events.amazonaws.com"}, "Action": "sts:AssumeRole"}]}'
aws iam put-role-policy --role-name sf-events-forward --policy-name put-events-to-us-east-1 --
policy-document \
  '{"Version": "2012-10-17", "Statement": [{"Effect": "Allow", "Action": "events:PutEvents",
"Resource": "'$BUS'"}]}'

# Every Region except us-east-1 forwards matching findings; us-east-1 pages directly.
for region in $REGIONS; do
  aws events put-rule --region $region --name sf-inspector-critical --state ENABLED --event-
pattern file://inspector-critical-pattern.json
  if [ "$region" = "us-east-1" ]; then
    aws sns create-topic --region us-east-1 --name sf-inspector-pagerduty
    aws events put-targets --region us-east-1 --rule sf-inspector-critical \
      --targets Id=PagerDuty,Arn=arn:aws:sns:us-east-1:<security-tooling-account-id>:sf-
inspector-pagerduty
  else
    aws events put-targets --region $region --rule sf-inspector-critical --targets
Id=ToUsEast1,Arn=$BUS,RoleArn=$FORWARD_ROLE
  fi
done
# The us-east-1 rule matches both its own findings and the forwarded ones, because forwarding
keeps the source and detail.
for region in $REGIONS; do
  echo "$region: $(aws events describe-rule --region $region --name sf-inspector-critical --
query State --output text)"
done
```

How it is verified

- `aws inspector2 batch-get-account-status` returns no account that is not ENABLED, in each of the 17 Regions, for every active account, including the accounts that existed before the change (Security Hub CSPM Inspector.1 to Inspector.4).
- `aws ecr get-registry-scanning-configuration` shows `ENHANCED` with `CONTINUOUS_SCAN` in each account (ECR.1).
- A known vulnerable test image pushed to ECR produces a finding within minutes.

How to roll it back. Disable a scan type for the organization with `aws inspector2 update-organization-configuration`, set ECR scanning back to basic with `aws ecr put-registry-scanning-configuration`, and remove the delegated administrator. Findings stay available for the retention period. Verify by reading the organization configuration.

SF-SEC-043: 750 default VPCs, most in Regions nobody uses, have open default security groups and network ACLs

Medium. Benchmark rating High (Security Hub CSPM EC2.2, EC2.6, and EC2.21). How it was rated: benchmark High (3) -1 exploitability (limited) +0 blast radius (production) = 2

Where: All 48 accounts in 17 Regions, 2,350 failed checks. **Controls:** `EC2.2` (VPC default security groups should not allow inbound or outbound traffic); `EC2.6` (VPC flow logging should be enabled in all VPCs); `EC2.21` (Network ACLs should not allow ingress from 0.0.0.0/0 to port 22 or port 3389). **SOC 2:** CC6.6, CC7.2. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 16 hours. **Planned:** Week 5, starting Aug 10; the rest (Replace the open default network ACLs in the working Regions' VPCs) is due October 30, 2026.

Evidence

- EC2.2 787, EC2.6 769, EC2.21 794 failed.

Examples: `EC2.2: sg-0c3f3f01f4453e4d8`, `EC2.2: sg-85ae1d0c3fc93f4a6`, `EC2.2: sg-3e7b1cf76171e31ac`.

Exploitability. Needs a foothold inside, an insider, or physical access. Only matters when someone launches into them, which nothing would notice in an unused Region. **Blast radius.** Production systems or customer data. Default VPCs in all 11 unused Regions of all 48 accounts.

How we verified it. Read from the scan.

Remediation: Remove default VPCs from unused Regions and lock down the ones that remain. Every Region has a default VPC with an open default security group (EC2.2), network ACLs open to 22 and 3389 (EC2.21), and no flow logs (EC2.6). Delete the default VPCs in the 11 Regions nobody uses, then restrict the default security group, add deny entries for the two admin ports to the ACLs, and send flow logs to the log archive in the rest.

Steps

1. Confirm the six Regions in use (us-east-1, us-west-2, eu-central-1, ap-northeast-1, eu-west-1, ap-southeast-1) and that nothing runs in the other 11.
2. In each unused Region, delete the default VPC's internet gateway, subnets, and VPC.
3. In the used Regions, save the default security group rules and network ACL entries, remove every rule from the default security group, and add deny entries for ports 22 and 3389 ahead of the ACL's allow-all rule, which stays because network ACLs are stateless.
4. Set `restrictDefaultSecurityGroup` and flow logs to the log archive bucket in the CDK VPC constructs, and in the StackSet for default VPCs that remain.

5. Add a service control policy later (see scp-guardrails) that denies actions in the unused Regions so a default VPC cannot return.

Delete the default VPC in each unused Region (AWS CLI)

```bash
for region in us-east-2 us-west-1 ca-central-1 sa-east-1 eu-west-2 eu-west-3 eu-north-1 ap-northeast-2 ap-northeast-3 ap-southeast-2 ap-south-1; do
  vpc=$(aws ec2 describe-vpcs --region $region --filters Name=isDefault,Values=true --query 'Vpcs[0].VpcId' --output text)
  [ "$vpc" = "None" ] && continue
  [ "$(aws ec2 describe-network-interfaces --region $region --filters Name=vpc-id,Values=$vpc --query 'length(NetworkInterfaces)' --output text)" = "0" ] || { echo "SKIP $region $vpc has interfaces"; continue; }
  for igw in $(aws ec2 describe-internet-gateways --region $region --filters Name=attachment.vpc-id,Values=$vpc --query 'InternetGateways[].InternetGatewayId' --output text); do
    aws ec2 detach-internet-gateway --region $region --internet-gateway-id $igw --vpc-id $vpc
    aws ec2 delete-internet-gateway --region $region --internet-gateway-id $igw
  done
  for subnet in $(aws ec2 describe-subnets --region $region --filters Name=vpc-id,Values=$vpc --query 'Subnets[].SubnetId' --output text); do
    aws ec2 delete-subnet --region $region --subnet-id $subnet
  done
  aws ec2 delete-vpc --region $region --vpc-id $vpc
done
```

Empty the default security group and close the ACL entries in a used Region (AWS CLI)

```bash
REGION=us-east-1
VPC=<default-vpc-id>
SG=$(aws ec2 describe-security-groups --region $REGION --filters Name=vpc-id,Values=$VPC
Name=group-name,Values=default --query 'SecurityGroups[0].GroupId' --output text)
# Save the rules first (the rollback restores from this file), and revoke only a list that is
not empty.
aws ec2 describe-security-groups --region $REGION --group-ids $SG > default-sg-before-$SG.json
INGRESS=$(aws ec2 describe-security-groups --region $REGION --group-ids $SG --query
'SecurityGroups[0].IpPermissions' --output json)
EGRESS=$(aws ec2 describe-security-groups --region $REGION --group-ids $SG --query
'SecurityGroups[0].IpPermissionsEgress' --output json)
[ "$INGRESS" = "[]" ] || aws ec2 revoke-security-group-ingress --region $REGION --group-id $SG
--ip-permissions "$INGRESS"
[ "$EGRESS" = "[]" ] || aws ec2 revoke-security-group-egress --region $REGION --group-id $SG --
ip-permissions "$EGRESS"
NACL=$(aws ec2 describe-network-acls --region $REGION --filters Name=vpc-id,Values=$VPC
Name=default,Values=true --query 'NetworkAcls[0].NetworkAclId' --output text)
# Network ACLs are stateless, so do not touch the allow-all rule 100: replacing it would drop
return traffic for every
# subnet on this ACL. Add deny entries with LOWER rule numbers, which are evaluated first, for
the two admin ports.
aws ec2 describe-network-acls --region $REGION --network-acl-ids $NACL > nacl-before-$NACL.json
n=10
for port in 22 3389; do
  aws ec2 create-network-acl-entry --region $REGION --network-acl-id $NACL --ingress --rule-
number $n \
    --protocol tcp --port-range From=$port,To=$port --cidr-block 0.0.0.0/0 --rule-action deny
  aws ec2 create-network-acl-entry --region $REGION --network-acl-id $NACL --ingress --rule-
number $((n + 1)) \
    --protocol tcp --port-range From=$port,To=$port --ipv6-cidr-block ::/0 --rule-action deny
  n=$((n + 10))
done
aws ec2 describe-network-acls --region $REGION --network-acl-ids $NACL \
  --query 'NetworkAcls[0].Entries[?Egress==`false`].
[RuleNumber,RuleAction,Protocol,PortRange.From,CidrBlock,Ipv6CidrBlock]' --output table
```

VPC with a restricted default security group and flow logs to the log archive (AWS CDK, TypeScript)

```typescript
import { Stack, StackProps, aws_ec2 as ec2, aws_s3 as s3 } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class WorkloadVpcStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const vpc = new ec2.Vpc(this, 'Vpc', { restrictDefaultSecurityGroup: true });
    vpc.addFlowLog('ToLogArchive', {
      destination: ec2.FlowLogDestination.toS3(s3.Bucket.fromBucketArn(this, 'Archive',
'arn:aws:s3:::<log-archive-flow-logs-bucket-name>'), 'vpc-flow-logs/'),
      trafficType: ec2.FlowLogTrafficType.ALL,
    });
  }
}
```

How it is verified

- `aws ec2 describe-vpcs --filters Name=isDefault,Values=true` returns nothing in the 11 unused Regions of every account.
- Security Hub CSPM EC2.2, EC2.21, and EC2.6 pass in the used Regions after the next evaluation.
- A flow log record for a test connection appears in the log archive bucket.

How to roll it back. A deleted default VPC comes back with `aws ec2 create-default-vpc` in that Region, with a new VPC ID. The change saves the default security group rules to `default-sg-before-<group-id>.json` and the network ACL to `nacl-before-<acl-id>.json`. Restore the group from the first copy with `aws ec2 authorize-security-group-ingress` and `authorize-security-group-egress`, and remove the four added ACL entries with `aws ec2 delete-network-acl-entry --ingress --rule-number` for 10, 11, 20, and 21; the allow-all rule 100 is never changed. Detach the Region deny SCP if a team has to work in an unused Region. Verify by describing the VPC and its flow logs.

SF-SEC-044: IAM Access Analyzer covers only us-east-1; 768 account-Regions have no external access analyzer

Medium. Benchmark rating High (Security Hub CSPM IAM.28). How it was rated: benchmark High (3) -1 exploitability (limited) +0 blast radius (production) = 2

Where: All 48 accounts in 16 Regions, 768 account-Regions. **Controls:** `IAM.28` (IAM Access Analyzer external access analyzer should be enabled). **SOC 2:** CC6.3, CC7.2. **Owner:** Nadia Petrova, Manager, Security Operations. **Repository:** `platform-landing-zone`. **Effort:** 4 hours. **Planned:** Week 5, starting Aug 10.

Evidence

- One organization analyzer in us-east-1.

Examples: `sf-management us-east-2`, `sf-management us-west-1`, `sf-management us-west-2`.

Exploitability. Needs a foothold inside, an insider, or physical access. External sharing of KMS keys, buckets, and roles outside us-east-1 goes unreported. **Blast radius.** Production systems or customer data. Resources in the five other working Regions.

How we verified it. Read from the scan.

Remediation: Run organization IAM Access Analyzer in all 17 Regions from the delegated administrator. The organization external access analyzer exists only in us-east-1 (IAM.28), so a public bucket or shared snapshot in any other Region goes unnoticed. Create the organization analyzer in all 17 Regions from sf-security-tooling and route active findings to the security queue.

Steps

1. Enable trusted access for Access Analyzer and register sf-security-tooling as delegated administrator.
2. Create an organization analyzer named `sf-org-external-access` in each Region from the delegated administrator.
3. Archive findings that are intended (CloudFront-fronted buckets, vendor roles with an ExternalId) with archive rules that name the resource.
4. Send active findings to the security queue with the event pattern below.
5. Deploy the analyzer through the stack set so a new Region is covered the day it is enabled.

Delegate and create the analyzer in every Region (AWS CLI)

```bash
# Management account:
aws organizations enable-aws-service-access --service-principal access-analyzer.amazonaws.com
aws organizations register-delegated-administrator --service-principal access-analyzer.amazonaws.com --account-id <security-tooling-account-id>
# sf-security-tooling:
for region in us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-west-1 eu-west-2 eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-southeast-1 ap-southeast-2 ap-south-1; do
  aws accessanalyzer create-analyzer --region $region --analyzer-name sf-org-external-access --type ORGANIZATION
done
```

The same analyzer as a stack set template (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Organization external access analyzer.
Resources:
  ExternalAccessAnalyzer:
    Type: AWS::AccessAnalyzer::Analyzer
    Properties:
      AnalyzerName: sf-org-external-access
      Type: ORGANIZATION
```

Active Access Analyzer findings (EventBridge event pattern)

```json
{
  "source": [
    "aws.access-analyzer"
  ],
  "detail-type": [
    "Access Analyzer Finding"
  ],
  "detail": {
    "status": [
      "ACTIVE"
    ]
  }
}
```

How it is verified

- `aws accessanalyzer list-analyzers --type ORGANIZATION` returns `sf-org-external-access` in all 17 Regions (Security Hub CSPM IAM.28).
- A test bucket policy that grants access to an outside account produces a finding within a few minutes.
- Every active finding has an owner or an archive rule with a written reason.

How to roll it back. Delete the analyzer in a Region with `aws accessanalyzer delete-analyzer`, or revert the stack set. Archive rules can be removed with `delete-archive-rule`, which makes the archived findings active again. Verify by listing the analyzers.

SF-SEC-045: 31 databases keep one day of backups; 112 resilience checks fail

Medium. Benchmark rating High (Security Hub CSPM RDS.13, RDS.5, and RDS.15). How it was rated: benchmark High (3) -1 exploitability (theoretical) +0 blast radius (production) = 2

Where: 15 accounts in 4 Regions, 143 database settings. **Controls:** `RDS.5` (RDS DB instances should be configured with multiple Availability Zones); `RDS.13` (RDS automatic minor version upgrades should be enabled). **SOC 2:** A1.2, A1.3, CC7.5, CC9.1. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 12 hours. **Planned:** Backlog, due October 30, 2026.

Evidence

- Backup retention of one day on 31 instances.
- 72 with automatic minor version upgrades off.

Examples: `db-dev-environments-1e064c`, `db-dev-environments-878b56`, `db-dev-environments-67520f`.

Exploitability. A defense-in-depth gap with no direct path on its own. An availability and recovery gap rather than an access path. **Blast radius.** Production systems or customer data. Production databases, including SAP-adjacent Oracle and Fleet API Aurora.

How we verified it. Read from the scan.

Remediation: Make production databases Multi-AZ, auto-patched, and backed up for 14 days. Production databases run in a single AZ (RDS.5, RDS.15), automatic minor version upgrades are off (RDS.13), and backups are kept for one day. Convert them to Multi-AZ, turn minor upgrades on, extend retention, and add an AWS Backup plan with an immutable copy.

Steps

1. Convert single-AZ instances to Multi-AZ in a maintenance window; the conversion takes a snapshot and may add I/O load while it runs.
2. For Aurora, add a reader in a second AZ with a promotion tier of 1.
3. Turn on automatic minor version upgrades and set the maintenance window to the team's quiet time.
4. Raise backup retention to 14 days and deploy the AWS Backup plan with daily and weekly rules.
5. Copy recovery points to a logically air-gapped vault in sf-backup, so a compromised account cannot delete them.

RDS instance: Multi-AZ, minor upgrades, 14-day backups (AWS CLI)

```bash
aws rds modify-db-instance --db-instance-identifier <db-instance-identifier> \
  --multi-az --auto-minor-version-upgrade --backup-retention-period 14 \
  --preferred-maintenance-window sun:07:00-sun:08:00 --apply-immediately
```

Aurora cluster and the air-gapped vault in the backup account (AWS CLI)

```bash
aws rds modify-db-cluster --db-cluster-identifier <aurora-cluster-identifier> --backup-retention-period 14 --apply-immediately
aws rds create-db-instance --db-cluster-identifier <aurora-cluster-identifier> --db-instance-identifier <aurora-cluster-identifier>-reader \
  --db-instance-class <db-instance-class> --engine aurora-postgresql --availability-zone <second-availability-zone> --promotion-tier 1
aws rds modify-db-instance --db-instance-identifier <aurora-cluster-identifier>-reader --auto-minor-version-upgrade --apply-immediately
# In sf-backup:
aws backup create-logically-air-gapped-backup-vault --backup-vault-name sf-fleet-airgapped \
  --min-retention-days 7 --max-retention-days 365
```

AWS Backup plan for tagged databases (AWS CDK, TypeScript)

```typescript
import { Duration, Stack, StackProps, aws_backup as backup, aws_events as events } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class DatabaseBackupStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const vault = new backup.BackupVault(this, 'Vault', { backupVaultName: 'sf-fleet-databases' });
    const plan = new backup.BackupPlan(this, 'Plan', { backupPlanName: 'sf-fleet-databases', backupVault: vault });
    plan.addRule(new backup.BackupPlanRule({ ruleName: 'daily', scheduleExpression: events.Schedule.cron({ minute: '0', hour: '5' }), deleteAfter: Duration.days(35) }));
    plan.addRule(new backup.BackupPlanRule({ ruleName: 'weekly', scheduleExpression: events.Schedule.cron({ minute: '0', hour: '6', weekDay: 'SUN' }), deleteAfter: Duration.days(365) }));
    plan.addSelection('TaggedDatabases', { resources: [backup.BackupResource.fromTag('backup', 'true')] });
  }
}
```

How it is verified

- `aws rds describe-db-instances --query 'DBInstances[?MultiAZ==false]'` returns nothing for production databases; Aurora clusters show instances in two AZs (RDS.5, RDS.15).
- `describe-db-instances` shows `AutoMinorVersionUpgrade` true (RDS.13) and `BackupRetentionPeriod` 14.
- AWS Backup lists a completed recovery point for each tagged database and a copy in the air-gapped vault.

How to roll it back. Convert an instance back to a single AZ with `aws rds modify-db-instance --no-multi-az`, which takes effect in the maintenance window. Reduce backup retention and turn off minor version upgrades with the same call. Delete an Aurora reader with `delete-db-instance`. Copies already in the air-gapped vault cannot be removed before their retention ends. Verify by describing the instance.

Low findings

9 Low findings, 128 hours between them.

SF-SEC-046: 130 customer managed policies use service-wide wildcards such as s3:*

Low. Benchmark rating Low (Security Hub CSPM IAM.21). How it was rated: benchmark Low (1) +0 exploitability (moderate) +0 blast radius (production) = 1

Where: 14 accounts in us-east-1, 130 IAM policies. **Controls:** `IAM.21` (IAM customer managed policies that you create should not allow wildcard actions for services). **SOC 2:** CC6.3. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `fleet-infra`. **Effort:** 40 hours. **Planned:** Backlog, due December 18, 2026.

Evidence

- 130 policies grant service:* actions (IAM.21).

Examples: `fleet-dev-svc-wildcard-00 (fleet-dev)`, `fleet-dev-svc-wildcard-01 (fleet-dev)`, `fleet-dev-svc-wildcard-02 (fleet-dev)`, `fleet-dev-svc-wildcard-03 (fleet-dev)`.

Exploitability. Remote, with one precondition (stolen credentials or an application flaw). A compromised role can do anything within the named service. **Blast radius.** Production systems or customer data. Spread across 14 accounts.

How we verified it. Read from the scan.

Remediation: Replace wildcard workload policies with scoped ones generated from CloudTrail. This is the same change as SF-SEC-017, applied to the accounts listed above. Use the steps, the code, the verification, and the rollback under SF-SEC-017, with this finding's accounts in place of that finding's.

SF-SEC-047: 2,850 of 14,464 resources have no Team tag, so no owner can be found for them

Low. Benchmark rating Low (Vantalect baseline: asset ownership). How it was rated: benchmark Low (1) -1 exploitability (theoretical) +0 blast radius (production) = 0

Where: All 48 accounts in 6 Regions, 2,850 resources. **Controls:** No benchmark control covers this. **SOC 2:** CC1.3, CC2.1. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 12 hours. **Planned:** Backlog, due December 18, 2026.

Evidence

- No Organizations tag policy; Team and Application tags applied by CDK only.

Examples: `no tag policy`, `no required-tags rule`.

Exploitability. A defense-in-depth gap with no direct path on its own. Slows response: nobody knows whom to call. **Blast radius.** Production systems or customer data. Mostly Kestrel and sandboxes.

How we verified it. Counted from the inventory.

Remediation: Standardize Team and Application tags with an Organizations tag policy and report missing ones. Many resources have no Team or Application owner tag, so nobody can say whose they are, what they cost, or who to call. A tag policy fixes the keys and allowed values, an AWS Config required-tags rule reports resources that lack them, and the CDK apps apply the tags by default.

Steps

1. Agree the allowed values for `Team` and `Application` with the four business units; start with the team list in the findings register.
2. Enable the tag policy type and attach the policy to the Sandbox OU first, then to the workload OUs.
3. Deploy the Config required-tags rule as a stack set; a tag policy checks the value of a tag that exists, so the rule is what finds the missing ones.
4. Apply `Team` and `Application` to every stack in the four CDK repositories with `Tags.of(app)`, so new resources arrive tagged.
5. Review the untagged-resource report weekly with the owning directors until it is empty, then require tags at creation with a later SCP.

Create and attach the tag policy (AWS CLI)

```bash
ROOT=<org-root-id>
aws organizations enable-policy-type --root-id "$ROOT" --policy-type TAG_POLICY
cat > sf-tag-policy.json <<'JSON'
{"tags": {
  "Team": {
    "tag_key": {"@@assign": "Team"},
    "tag_value": {"@@assign": ["fleet", "crucible", "enterprise", "platform", "security",
"kestrel"]},
    "enforced_for": {"@@assign": ["ec2:instance", "ec2:volume", "rds:db", "s3:bucket",
"lambda:function", "eks:cluster"]}},
  "Application": {
    "tag_key": {"@@assign": "Application"},
    "enforced_for": {"@@assign": ["ec2:instance", "ec2:volume", "rds:db", "s3:bucket",
"lambda:function", "eks:cluster"]}}}}
JSON
POLICY=$(aws organizations create-policy --name sf-team-application-tags --type TAG_POLICY \
  --description "Standard Team and Application tags" --content file://sf-tag-policy.json \
  --query Policy.PolicySummary.Id --output text)
aws organizations attach-policy --policy-id "$POLICY" --target-id <sandbox-ou-id>
```

Config rule that reports resources missing the tags (deploy as a stack set) (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Report resources without Team and Application tags.
Resources:
  RequiredTags:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sf-required-team-application-tags
      InputParameters:
        tag1Key: Team
        tag2Key: Application
      Scope:
        ComplianceResourceTypes:
          - AWS::EC2::Instance
          - AWS::EC2::Volume
          - AWS::RDS::DBInstance
          - AWS::S3::Bucket
          - AWS::Lambda::Function
      Source:
        Owner: AWS
        SourceIdentifier: REQUIRED_TAGS
```

Tag every stack in a CDK app (AWS CDK, TypeScript)

```typescript
import { App, Tags } from 'aws-cdk-lib';
import { FleetApiStack } from '../lib/fleet-api-stack';

const app = new App();
new FleetApiStack(app, 'FleetApi');
Tags.of(app).add('Team', 'fleet');
Tags.of(app).add('Application', 'fleet-api');
```

How it is verified

- `aws organizations list-policies-for-target --target-id <sandbox-ou-id> --filter TAG_POLICY` lists `sf-team-application-tags`.
- The Config rule `sf-required-team-application-tags` shows no non-compliant resource in the pilot accounts, and the count falls each week elsewhere.
- `aws resourcegroupstaggingapi get-resources --tag-filters Key=Team` returns an owner for a sample of resources from every business unit.

How to roll it back. Detach the tag policy with `aws organizations detach-policy`, and delete the Config rule stack set. Tags that were applied stay on the resources. Revert the CDK tagging change in the four repositories if a stack fails on a tag limit. Verify with the tag compliance report.

SF-SEC-048: 1,922 workload roles have not been used in 90 days or more

Low. Benchmark rating Low (Vantalect baseline: unused access). How it was rated: benchmark Low (1) -1 exploitability (theoretical) +0 blast radius (production) = 0

Where: 47 accounts in us-east-1, 1,922 IAM roles. **Controls:** No benchmark control covers this. **SOC 2:** CC6.3. **Owner:** Grace Liu, Manager, Identity and Access. **Repository:** `platform-landing-zone`. **Effort:** 16 hours. **Planned:** Backlog, due September 4, 2026.

Evidence

- IAM last-accessed data: 460 never used, 691 unused for over a year.

Examples: `sf-management-workload-003 (sf-management)`, `sf-management-workload-005 (sf-management)`, `sf-management-workload-009 (sf-management)`.

Exploitability. A defense-in-depth gap with no direct path on its own. Unused roles widen what a compromised principal can assume. **Blast radius.** Production systems or customer data. Spread across 48 accounts.

How we verified it. Last-accessed data read with iam generate-service-last-accessed-details.

Remediation: Find roles unused for 90 days, export and tag them in the account that owns them, and delete them after a 14-day hold. Roles nobody has assumed in 90 days keep trust relationships and permissions that an attacker can still use. IAM Access Analyzer's unused access analyzer finds them across the organization; each finding names the account that owns the role (`resourceOwnerAccount`), so every check, export, tag, and deletion runs in that account under an assumed role, never in sf-security-tooling by bare role name.

Steps

1. From the delegated administrator (sf-security-tooling), create one organization unused access analyzer with a 90-day age. IAM is global, so one analyzer in us-east-1 covers every Region.
2. List UnusedIAMRole findings with their owning account and role ARN, and skip service-linked roles (path `/aws-service-role/`), the Identity Center roles (path `/aws-reserved/`), the Control Tower and StackSet roles, the remediation role itself, and roles used only for quarterly or annual jobs.
3. For each finding, assume the remediation role in the owning account, re-check `RoleLastUsed` there, export the role's trust policy, attached policies, and inline policies to disk, and tag the role with `sf-retire-after` set 14 days out. If any export step fails, do not tag the role.
4. Tell the owning teams, then after the hold run the deletion script: in each owning account it re-checks the tag, the last-used date, and the export, detaches policies and instance profiles, and deletes the role.
5. Keep the analyzer running and review new findings every month.

Organization unused access analyzer (deploy in sf-security-tooling, us-east-1) (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: IAM Access Analyzer organization analyzer for unused access, 90-day window.
Resources:
  UnusedAccessAnalyzer:
    Type: AWS::AccessAnalyzer::Analyzer
    Properties:
      AnalyzerName: sf-org-unused-access
      Type: ORGANIZATION_UNUSED_ACCESS
      AnalyzerConfiguration:
        UnusedAccessConfiguration:
          UnusedAccessAge: 90
```

List unused roles by owning account, then export, re-check, and tag each one in that account (AWS CLI)

```bash
```

```
set -uo pipefail
ANALYZER=arn:aws:access-analyzer:us-east-1:<security-tooling-account-id>:analyzer/sf-org-
unused-access
ROLE_IN_MEMBER=<member-account-remediation-role-name>
# GNU date first, then macOS date.
RETIRE_AFTER=$(date -u -d '+14 days' +%F 2>/dev/null || date -u -v+14d +%F)
CUTOFF=$(date -u -d '90 days ago' +%Y-%m-%d 2>/dev/null || date -u -v-90d +%Y-%m-%d)

aws accessanalyzer list-findings-v2 --analyzer-arn "$ANALYZER" \
  --filter '{"findingType": {"eq": ["UnusedIAMRole"]}, "status": {"eq": ["ACTIVE"]}}' \
  --query 'findings[].[resourceOwnerAccount,resource]' --output text > unused-roles.txt
: > retire-queue.csv

while read -r account role_arn; do
  role="${role_arn##*/}"; path="${role_arn#*:role}"
  case "$path" in /aws-service-role/*|/aws-reserved/*) continue;; esac
  case "$role" in AWSControlTower*|stacksets-exec-*|
OrganizationAccountAccessRole|"$ROLE_IN_MEMBER") continue;; esac
  creds=$(aws sts assume-role --role-arn "arn:aws:iam::${account}:role/${ROLE_IN_MEMBER}" --
role-session-name sf-retire-unused-roles \
    --duration-seconds 3600 --query Credentials --output json) || { echo "NO ACCESS to
$account; $role untouched" >&2; continue; }
  (
    export AWS_ACCESS_KEY_ID=$(echo "$creds" | jq -r .AccessKeyId) AWS_SECRET_ACCESS_KEY=$(echo
"$creds" | jq -r .SecretAccessKey)
    export AWS_SESSION_TOKEN=$(echo "$creds" | jq -r .SessionToken)
    # Re-check in the owning account: the role must exist and must not have been used since the
cutoff.
    last=$(aws iam get-role --role-name "$role" --query 'Role.RoleLastUsed.LastUsedDate' --
output text) || exit 1
    if [ "$last" != "None" ] && [ "${last:0:10}" \> "$CUTOFF" ]; then echo "SKIP $account
$role: used $last" >&2; exit 0; fi
    dir="retired-roles/$account/$role"; mkdir -p "$dir"
    aws iam get-role --role-name "$role" --output json > "$dir/role.json" && test -s "$dir/
role.json" || exit 1
    aws iam list-attached-role-policies --role-name "$role" --output json > "$dir/
attached.json" && test -s "$dir/attached.json" || exit 1
    for name in $(aws iam list-role-policies --role-name "$role" --query 'PolicyNames[]' --
output text); do
      aws iam get-role-policy --role-name "$role" --policy-name "$name" --output json > "$dir/
inline-$name.json" && test -s "$dir/inline-$name.json" || exit 1
    done
    aws iam tag-role --role-name "$role" --tags Key=sf-retire-after,Value="$RETIRE_AFTER" ||
exit 1
    echo "$account,$role,$RETIRE_AFTER" >> retire-queue.csv
  ) || echo "NOT TAGGED $account $role: an export or tag step failed" >&2
done < unused-roles.txt
```

After the hold, delete each queued role in the account that owns it (AWS CLI)

```bash
set -uo pipefail
ROLE_IN_MEMBER=<member-account-remediation-role-name>
TODAY=$(date -u +%F)
CUTOFF=$(date -u -d '90 days ago' +%Y-%m-%d 2>/dev/null || date -u -v-90d +%Y-%m-%d)
while IFS=, read -r account role due; do
  [ "$due" \> "$TODAY" ] && continue
  creds=$(aws sts assume-role --role-arn "arn:aws:iam::${account}:role/${ROLE_IN_MEMBER}" --
role-session-name sf-delete-unused-roles \
    --duration-seconds 3600 --query Credentials --output json) || { echo "NO ACCESS to
$account; $role kept" >&2; continue; }
  (
    export AWS_ACCESS_KEY_ID=$(echo "$creds" | jq -r .AccessKeyId) AWS_SECRET_ACCESS_KEY=$(echo
"$creds" | jq -r .SecretAccessKey)
    export AWS_SESSION_TOKEN=$(echo "$creds" | jq -r .SessionToken)
    tag=$(aws iam list-role-tags --role-name "$role" --query 'Tags[?Key==`sf-retire-
after`].Value | [0]' --output text) || exit 1
    [ "$tag" = "$due" ] || { echo "KEEP $account $role: tag removed or changed" >&2; exit 0; }
    last=$(aws iam get-role --role-name "$role" --query 'Role.RoleLastUsed.LastUsedDate' --
output text) || exit 1
    if [ "$last" != "None" ] && [ "${last:0:10}" \> "$CUTOFF" ]; then echo "KEEP $account
$role: used $last" >&2; exit 0; fi
    test -s "retired-roles/$account/$role/role.json" || { echo "KEEP $account $role: no export
on disk" >&2; exit 0; }
    for arn in $(aws iam list-attached-role-policies --role-name "$role" --query
'AttachedPolicies[].PolicyArn' --output text); do
      aws iam detach-role-policy --role-name "$role" --policy-arn "$arn" || exit 1
    done
    for name in $(aws iam list-role-policies --role-name "$role" --query 'PolicyNames[]' --
output text); do
      aws iam delete-role-policy --role-name "$role" --policy-name "$name" || exit 1
    done
    for profile in $(aws iam list-instance-profiles-for-role --role-name "$role" --query
'InstanceProfiles[].InstanceProfileName' --output text); do
      aws iam remove-role-from-instance-profile --instance-profile-name "$profile" --role-name
"$role" || exit 1
    done
    aws iam delete-role --role-name "$role"
  ) || echo "FAILED $account $role" >&2
done < retire-queue.csv
```

How it is verified

- The unused access analyzer shows no ACTIVE UnusedIAMRole finding older than the 14-day hold, in any of the member accounts that owned a finding.
- Every deleted role appears in a CloudTrail `DeleteRole` event in its owning account that matches the change ticket, and `retired-roles/<account>/<role>/` holds its export.
- No production workload reports an `AssumeRole` AccessDenied in the week after the deletions, and no role in sf-security-tooling was touched unless a finding named that account.

How to roll it back. Cancel a deletion by removing the tag in the owning account with `aws iam untag-role --role-name <role> --tag-keys sf-retire-after` and deleting the line from `retire-queue.csv`. A deleted role cannot be restored, but step 3 wrote its `role.json`, `attached.json`, and inline policy files to `retired-roles/<account>/<role>/`: recreate it with `aws iam create-role` from the saved trust policy, re-attach the saved managed policies, and `put-role-policy` the inline ones. The new role has a new unique ID, so fix any resource policy that named the old ID. Revert the analyzer stack to stop the findings. Verify that the role can be assumed and that the analyzer lists no finding for it.

SF-SEC-049: MFA delete is off on every bucket

Low. Benchmark rating Low (Security Hub CSPM S3.20). How it was rated: benchmark Low (1) -1 exploitability (theoretical) +0 blast radius (production) = 0

Where: 23 accounts in 5 Regions, 317 S3 buckets. **Controls:** `S3.20` (S3 general purpose buckets should have MFA delete enabled). **SOC 2:** CC7.2. **Owner:** Nadia Petrova, Manager, Security Operations. **Repository:** `platform-landing-zone`. **Effort:** 4 hours. **Planned:** Week 5, starting Aug 10. **Depends on:** SF-SEC-025.

Evidence

- MFA delete needs the root user's credentials, which week four removes; Object Lock is the stronger control.
- The organization trail has logged S3 data events on every bucket since 2023 (S3.22 and S3.23 pass); at about $42,005 a month for 42,005.3 million S3 requests it is the largest logging cost, a question for the cost review rather than a security gap.

Examples: `arn:aws:s3:::sf-billing-cur-0007-us-east-1-000`, `arn:aws:s3:::sf-billing-cur-0007-us-east-1-001`, `arn:aws:s3:::sf-billing-cur-0007-us-east-1-002`, `arn:aws:s3:::sf-log-archive-cloudtrail-2291-us-east-1-000`.

Exploitability. A defense-in-depth gap with no direct path on its own. Affects recovery from a deletion, not access. **Blast radius.** Production systems or customer data. Every bucket, including the OTA distribution and dataset buckets.

How we verified it. Read from the scan.

Remediation: Protect critical buckets with versioning and Object Lock instead of MFA delete. MFA delete is off on every bucket (S3.20). Only a bucket owner's root user can turn it on, and centralized root access management removes those credentials, so protect the OTA distribution bucket and the log archive with versioning and Object Lock, keep the organization trail's S3 data events, and record S3.20 as an accepted exception.

Steps

1. Keep the organization trail's S3 data events on every bucket: they have been on since 2023 and make S3.22 and S3.23 pass. Whether to narrow them for cost is a decision for the cost review, not a security fix.
2. Do not enable MFA delete: only the bucket owner's root user can turn it on, and the member-account root credentials are being deleted (see root-credentials).
3. Turn on versioning and Object Lock in governance mode on the OTA distribution bucket; the new organization trail bucket already uses compliance mode (see cloudtrail-log-archive).
4. Record S3.20 as an accepted exception, with versioning, Object Lock, and data events as the compensating controls, and suppress the S3.20 findings with that note.

Confirm the organization trail records S3 data events for every bucket (AWS CLI)

```bash
aws cloudtrail get-event-selectors --trail-name sf-org-trail-v2 \
  --query 'AdvancedEventSelectors[].{name: Name, fields: FieldSelectors[].Field}' --output json
aws cloudtrail get-trail-status --name sf-org-trail-v2 --query '[IsLogging,LatestDeliveryTime]'
```

Versioning and Object Lock on the OTA distribution bucket (AWS CLI)

```bash
b=<ota-distribution-bucket>
aws s3api put-bucket-versioning --bucket "$b" --versioning-configuration Status=Enabled
aws s3api put-object-lock-configuration --bucket "$b" \
  --object-lock-configuration '{"ObjectLockEnabled": "Enabled", "Rule": {"DefaultRetention":
{"Mode": "GOVERNANCE", "Days": 30}}}'
aws s3api get-object-lock-configuration --bucket "$b"
```

Suppress S3.20 with the exception note (in the Security Hub CSPM administrator account) (AWS CLI)

```bash
aws securityhub get-findings --region us-east-1 \
  --filters '{"ComplianceSecurityControlId": [{"Value": "S3.20", "Comparison": "EQUALS"}],
"WorkflowStatus": [{"Value": "NEW", "Comparison": "EQUALS"}]}' \
  --query 'Findings[].[Id,ProductArn]' --output text > s3-20-findings.tsv
while read -r id product; do
  aws securityhub batch-update-findings --region us-east-1 \
    --finding-identifiers Id="$id",ProductArn="$product" \
    --workflow Status=SUPPRESSED \
    --note Text="Accepted exception: MFA delete needs root credentials; versioning, Object
Lock, and data events instead",UpdatedBy=<your-name>
done < s3-20-findings.tsv
```

How it is verified

- Security Hub CSPM S3.22 and S3.23 pass in every account.
- `aws s3api get-object-lock-configuration` returns an enabled configuration on the OTA distribution bucket.
- S3.20 findings show workflow status SUPPRESSED with the exception note, and the exception is in the risk-acceptance log.

How to roll it back. Object Lock in governance mode can be bypassed by a principal with `s3:BypassGovernanceRetention`, so the lock can be undone in an emergency, but versioning cannot be turned off once on, only suspended. Remove the S3.20 suppression note with the Security Hub CSPM workflow update and the finding returns. Verify by reading the bucket's versioning and Object Lock configuration.

SF-SEC-050: Account baseline gaps: no security contact in 42 accounts, no support role in 20, CloudShell full access on 195 identities

Low. Benchmark rating Medium (Security Hub CSPM Account.1, IAM.15, IAM.16, IAM.18, IAM.26, and IAM.27). How it was rated: benchmark Medium (2) -1 exploitability (theoretical) +0 blast radius (production) = 1

Where: All 48 accounts in us-east-1, 275 failed checks. **Controls:** `Account.1` (Security contact information should be provided for an AWS account); `IAM.15` (Ensure IAM password policy requires minimum password length of 14 or greater); `IAM.16` (Ensure IAM password policy prevents password reuse); `IAM.18` (Ensure a support role has been created to manage incidents with AWS Support); `IAM.26` (Expired SSL/TLS certificates managed in IAM should be removed); `IAM.27` (IAM identities should not have the AWSCloudShellFullAccess policy attached). **SOC 2:** CC2.3, CC6.1. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 10 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- Account.1: 42 failed
- IAM.18: 20 failed
- IAM.15: 8 failed
- IAM.16: 7 failed
- IAM.26: 3 failed
- IAM.27: 195 failed

Examples: `Account.1: sf-network-hub`, `Account.1: sf-shared-services`, `Account.1: sf-identity`, `Account.1: sf-cicd`, `Account.1: sf-backup`.

Exploitability. A defense-in-depth gap with no direct path on its own. Settings that slow an investigation or widen a compromise rather than open a path. **Blast radius.** Production systems or customer data. Every account the settings are missing from.

How we verified it. Read from the scan; no manual check needed.

Remediation: Set the account baseline: password policy, support role, security contact, CloudShell, and expired certificates. Several accounts miss the basics that Security Hub CSPM checks first: a strong password policy (IAM.15, IAM.16), a support role (IAM.18), a security contact (Account.1), no AWSCloudShellFullAccess (IAM.27), and no expired server certificates (IAM.26).

Steps

1. Deploy the support role as a stack set to every account; CloudFormation has no resource for the password policy, so set that with the CLI.
2. Apply the password policy in every account: 14 characters minimum and the last 24 passwords remembered.
3. Set the security alternate contact for every account from the management account.
4. Detach AWSCloudShellFullAccess from every user, group, and role, and grant narrower CloudShell actions where they are needed.
5. Delete expired IAM server certificates after confirming no load balancer uses them; ACM replaces them.

Support role for the stack set (IAM.18) (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Role that can manage AWS Support cases.
Parameters:
  SupportTrustedPrincipalArn:
    Type: String
    Description: ARN of the role that may assume the support role (the Identity Center support
permission set role).
Resources:
  SupportRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: sf-aws-support-access
      ManagedPolicyArns: [arn:aws:iam::aws:policy/AWSSupportAccess]
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal: {AWS: !Ref SupportTrustedPrincipalArn}
            Action: 'sts:AssumeRole'
```

Password policy and security contact (AWS CLI)

```bash
aws iam get-account-password-policy --output json > password-policy.before.json || echo '{}' >
password-policy.before.json
aws iam update-account-password-policy --minimum-password-length 14 --require-symbols --
require-numbers \
  --require-uppercase-characters --require-lowercase-characters --max-password-age 90 --
password-reuse-prevention 24
# Run from the management account for each member account id.
aws account put-alternate-contact --account-id <member-account-id> --alternate-contact-type
SECURITY \
  --name "StrataForge Security Operations" --title "Security Operations" \
  --email-address security@strataforge.example --phone-number "<security-team-phone-number>"
```

Detach AWSCloudShellFullAccess and delete expired server certificates (AWS CLI)

```bash
POLICY=arn:aws:iam::aws:policy/AWSCloudShellFullAccess
for user in $(aws iam list-entities-for-policy --policy-arn "$POLICY" --query
'PolicyUsers[].UserName' --output text); do
  aws iam detach-user-policy --user-name "$user" --policy-arn "$POLICY"
done
for group in $(aws iam list-entities-for-policy --policy-arn "$POLICY" --query
'PolicyGroups[].GroupName' --output text); do
  aws iam detach-group-policy --group-name "$group" --policy-arn "$POLICY"
done
for role in $(aws iam list-entities-for-policy --policy-arn "$POLICY" --query
'PolicyRoles[].RoleName' --output text); do
  aws iam detach-role-policy --role-name "$role" --policy-arn "$POLICY"
done
NOW=$(date -u +%Y-%m-%dT%H:%M:%SZ)
for name in $(aws iam list-server-certificates --query "ServerCertificateMetadataList[?
Expiration<'${NOW}'].ServerCertificateName" --output text); do
  aws iam delete-server-certificate --server-certificate-name "$name"
done
```

How it is verified

- Security Hub CSPM IAM.15, IAM.16, IAM.18, IAM.26, IAM.27, and Account.1 pass in every account after the next evaluation.
- `aws iam get-account-password-policy` shows MinimumPasswordLength 14 and PasswordReusePrevention 24.
- `aws account get-alternate-contact --alternate-contact-type SECURITY` returns the security team address.

How to roll it back. Revert the stack set for the support role. Set the previous password policy from `password-policy.before.json`, which step 2 saves, with `aws iam update-account-password-policy` and the values it holds (for example `--minimum-password-length`), or run `aws iam delete-account-password-policy` to return to the AWS defaults. Re-attach `AWSCloudShellFullAccess` to a

user or role that needs it with `aws iam attach-role-policy`. A deleted server certificate cannot come back, so keep its export until the load balancers are confirmed clean. Verify with Security Hub CSPM controls IAM.15, IAM.16, IAM.18, and IAM.27.

SF-SEC-051: Server access logging is off on 233 buckets

Low. Benchmark rating Medium (Security Hub CSPM S3.9). How it was rated: benchmark Medium (2) -1 exploitability (theoretical) +0 blast radius (production) = 1

Where: 23 accounts in 5 Regions, 233 S3 buckets. **Controls:** `S3.9` (S3 general purpose buckets should have server access logging enabled). **SOC 2:** CC4.1, CC7.2. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 12 hours. **Planned:** Week 5, starting Aug 10.

Evidence

- get-bucket-logging returns no target.

Examples: `arn:aws:s3:::sf-billing-cur-0007-us-east-1-001`, `arn:aws:s3:::sf-log-archive-elb-access-logs-2291-us-east-1-001`, `arn:aws:s3:::sf-log-archive-elb-access-logs-2291-us-east-1-002`, `arn:aws:s3:::sf-log-archive-config-history-2291-us-east-1`.

Exploitability. A defense-in-depth gap with no direct path on its own. Slows an investigation; opens no path. **Blast radius.** Production systems or customer data. The Kestrel incident above cannot be scoped for this reason.

How we verified it. Read from the scan.

Remediation: Turn on S3 server access logging with a log bucket in each account and Region. Server access logging is off on most buckets (S3.9), including the CloudTrail bucket (CloudTrail.7). S3 requires the log bucket in the same account and Region as the source bucket, so create one per account and Region and replicate it to the log archive.

Steps

1. Deploy the log bucket stack to every account and Region that has buckets; it grants the S3 logging service write access and refuses non-TLS requests.
2. Enable logging on every existing bucket with a prefix of the bucket name, and set the CloudTrail bucket first.
3. Replicate each log bucket to the sf-log-archive bucket; the replication role and the versioning requirement are in the template.
4. Make new buckets log by default by adding the logging property to the CDK bucket constructs in all four repositories.

5. Re-scan for S3.9 and CloudTrail.7.

Per-account log bucket with replication to the log archive (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: S3 server access log bucket, replicated to the log archive.
Parameters:
  ArchiveBucketArn:
    Type: String
    Description: ARN of the destination bucket in sf-log-archive (arn:aws:s3:::<log-archive-
bucket-name>).
Resources:
  AccessLogBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Sub 'sf-s3-access-logs-${AWS::AccountId}-${AWS::Region}'
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault: {SSEAlgorithm: AES256}
      PublicAccessBlockConfiguration: {BlockPublicAcls: true, BlockPublicPolicy: true,
IgnorePublicAcls: true, RestrictPublicBuckets: true}
      VersioningConfiguration: {Status: Enabled}
      ReplicationConfiguration:
        Role: !GetAtt ReplicationRole.Arn
        Rules:
          - Id: ToLogArchive
            Status: Enabled
            Destination: {Bucket: !Ref ArchiveBucketArn}
  AccessLogBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AccessLogBucket
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Sid: S3ServerAccessLogsDelivery
            Effect: Allow
            Principal: {Service: logging.s3.amazonaws.com}
            Action: 's3:PutObject'
            Resource: !Sub '${AccessLogBucket.Arn}/*'
            Condition:
              StringEquals: {'aws:SourceAccount': !Ref 'AWS::AccountId'}
          - Sid: DenyNonTlsRequests
            Effect: Deny
            Principal: '*'
            Action: 's3:*'
            Resource: [!GetAtt AccessLogBucket.Arn, !Sub '${AccessLogBucket.Arn}/*']
            Condition: {Bool: {'aws:SecureTransport': 'false'}}
  ReplicationRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - {Effect: Allow, Principal: {Service: s3.amazonaws.com}, Action: 'sts:AssumeRole'}
```

```
      Policies:
        - PolicyName: replicate-access-logs
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action: ['s3:GetReplicationConfiguration', 's3:ListBucket']
                Resource: !Sub 'arn:aws:s3:::sf-s3-access-logs-${AWS::AccountId}-$
{AWS::Region}'
              - Effect: Allow
                Action: ['s3:GetObjectVersionForReplication', 's3:GetObjectVersionAcl',
's3:GetObjectVersionTagging']
                Resource: !Sub 'arn:aws:s3:::sf-s3-access-logs-${AWS::AccountId}-$
{AWS::Region}/*'
              - Effect: Allow
                Action: ['s3:ReplicateObject', 's3:ReplicateDelete', 's3:ReplicateTags']
                Resource: !Sub '${ArchiveBucketArn}/*'
```

Enable logging on every bucket in the current Region (AWS CLI)

```bash
REGION=us-east-1
ACCOUNT=$(aws sts get-caller-identity --query Account --output text)
LOGS=sf-s3-access-logs-$ACCOUNT-$REGION
for b in $(aws s3api list-buckets --query "Buckets[?starts_with(Name, 'sf-')].Name" --output
text); do
  [ "$b" = "$LOGS" ] && continue
  [ "$(aws s3api get-bucket-location --bucket "$b" --query 'LocationConstraint' --output text |
sed 's/^None$/us-east-1/')" = "$REGION" ] || continue
  aws s3api put-bucket-logging --bucket "$b" \
    --bucket-logging-status "{\"LoggingEnabled\": {\"TargetBucket\": \"$LOGS\",
\"TargetPrefix\": \"$b/\"}}"
done
```

How it is verified

- `aws s3api get-bucket-logging --bucket sf-log-archive-org-trail-locked-us-east-1` shows a target bucket (CloudTrail.7).
- Security Hub CSPM S3.9 passes for every general purpose bucket after the next evaluation.
- A test request to a logged bucket produces an object in the log bucket within a few hours and in the log archive after replication.

How to roll it back. Disable logging on a bucket with `aws s3api put-bucket-logging` and an empty logging status, and revert the log bucket stack. Leave the log objects in place; they are evidence. Verify by reading the bucket's logging configuration.

SF-SEC-052: 220 buckets accept requests without TLS

Low. Benchmark rating Medium (Security Hub CSPM S3.5). How it was rated: benchmark Medium (2) -1 exploitability (theoretical) +0 blast radius (production) = 1

Where: 22 accounts in 5 Regions, 220 S3 buckets. **Controls:** `S3.5` (S3 general purpose buckets should require requests to use TLS). **SOC 2:** CC6.7. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 6 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- No bucket policy statement denies aws:SecureTransport false.

Examples: `arn:aws:s3:::sf-billing-cur-0007-us-east-1-000`, `arn:aws:s3:::sf-billing-cur-0007-us-east-1-001`, `arn:aws:s3:::sf-billing-cur-0007-us-east-1-002`, `arn:aws:s3:::sf-log-archive-vpc-flow-logs-2291-us-east-1-001`.

Exploitability. A defense-in-depth gap with no direct path on its own. Needs a client that sends plain HTTP and an attacker on its path. **Blast radius.** Production systems or customer data. Includes telemetry and sensor-log buckets.

How we verified it. Read from the scan.

Remediation: Require TLS on S3, KMS, SQS, Secrets Manager, and STS requests with a resource control policy. Buckets across the organization accept plain HTTP requests (S3.5). A resource control policy that denies requests when `aws:SecureTransport` is false covers every bucket in every member account at once, including buckets nobody has reviewed, and the same statement covers the KMS keys, queues, secrets, and role trust policies the RCP type supports.

Steps

1. Confirm no client still uses HTTP to S3 (old on-premises tools, public static sites); CloudTrail's `tlsDetails` field and S3 access logs show it.
2. Enable the resource control policy type on the organization root.
3. Create the RCP, attach it to a sandbox OU first, test an HTTP request, and then attach it to the root.
4. RCPs do not apply to resources in the management account, so put the same TLS-only bucket policy on the few buckets that live there.
5. Re-scan: if Security Hub CSPM S3.5 still reports buckets that the RCP covers because the control reads bucket policies, add the bucket policy to those buckets with a stack set.

sf-rcp-enforce-tls: deny requests that do not use TLS (Resource control policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceTls",
      "Effect": "Deny",
      "Principal": "*",
      "Action": [
        "s3:*",
        "kms:*",
        "sqs:*",
        "secretsmanager:*",
        "sts:*"
      ],
      "Resource": "*",
      "Condition": {
        "BoolIfExists": {
          "aws:SecureTransport": "false"
        }
      }
    }
  ]
}
```

Enable the policy type, create the RCP, and attach it (AWS CLI)

```BASH
ROOT=<org-root-id>
aws organizations enable-policy-type --root-id "$ROOT" --policy-type RESOURCE_CONTROL_POLICY
POLICY=$(aws organizations create-policy --name sf-rcp-enforce-tls --type
RESOURCE_CONTROL_POLICY \
  --description "Deny requests that do not use TLS" --content file://policies/rcp/enforce-
tls.json \
  --query Policy.PolicySummary.Id --output text)
aws organizations attach-policy --policy-id "$POLICY" --target-id <sandbox-ou-id>
# After the sandbox test passes:
aws organizations attach-policy --policy-id "$POLICY" --target-id "$ROOT"
```

TLS-only bucket policy for buckets in the management account (S3 bucket policy)

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyNonTlsRequests",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::<management-account-bucket-name>",
        "arn:aws:s3:::<management-account-bucket-name>/*"
      ],
      "Condition": {
        "Bool": {
          "aws:SecureTransport": "false"
        }
      }
    }
  ]
}
```

How it is verified

- `aws s3api get-object --endpoint-url http://s3.us-east-1.amazonaws.com --bucket <bucket-name> --key <object-key> /dev/null` against a member-account bucket fails with AccessDenied.
- `aws organizations list-policies-for-target --target-id <org-root-id> --filter RESOURCE_CONTROL_POLICY` lists `sf-rcp-enforce-tls`.
- Security Hub CSPM S3.5 passes for every bucket after the next evaluation, or the remaining findings are covered by the bucket policy.

How to roll it back. Detach `sf-rcp-enforce-tls` from the OU or root with `aws organizations detach-policy`, starting at the narrowest level where a client fails. Detaching returns the organization to accepting plain HTTP, so fix the client that still uses HTTP first and reattach within the day. Remove the bucket policy from the management account buckets by reverting its stack. Verify by repeating the failing request over HTTPS.

SF-SEC-053: 78 customer managed keys do not rotate

Low. Benchmark rating Medium (Security Hub CSPM KMS.4). How it was rated: benchmark Medium (2) -1 exploitability (theoretical) +0 blast radius (production) = 1

Where: 36 accounts in 6 Regions, 78 KMS keys. **Controls:** `KMS.4` (AWS KMS key rotation should be enabled). **SOC 2:** CC6.1. **Owner:** Elena Vasquez, Vice President, Cloud Platform. **Repository:** `platform-landing-zone`. **Effort:** 4 hours. **Planned:** Week 3, starting Jul 27.

Evidence

- get-key-rotation-status: KeyRotationEnabled false.

Examples: `d404d9ee-a951`, `8d326869-640c`, `9fe15aa1-8a7c`.

Exploitability. A defense-in-depth gap with no direct path on its own. Limits exposure of old key material; opens no path. **Blast radius.** Production systems or customer data. Keys behind production databases and buckets.

How we verified it. Read from the scan; two keys with imported material removed as false positives.

Remediation: Enable automatic rotation on every customer managed KMS key that supports it. Customer managed keys without rotation fail KMS.4. Turn rotation on for every symmetric key, make it the default in CDK, and block anyone from turning it off. Asymmetric keys cannot rotate automatically and need a scheduled replacement.

Steps

1. List customer managed keys in every account and Region, and keep the symmetric encryption keys whose key material comes from KMS.
2. Enable rotation on each with a 365-day period.
3. For asymmetric keys, such as the firmware signing keys, record a replacement schedule: create a new key, publish the new public key to the fleet, and move the alias.
4. Set `enableKeyRotation` in every CDK key construct in the four repositories.
5. Attach `sf-protect-account-defaults` (see scp-guardrails); its ProtectKeyRotation statement denies `kms:DisableKeyRotation` outside the platform roles.

Enable rotation on every eligible key in the current account (AWS CLI)

```bash
# Every key that should rotate and does not is written to rotation-failures.txt; the file must
end up empty.
: > rotation-failures.txt
for region in us-east-1 us-east-2 us-west-1 us-west-2 ca-central-1 sa-east-1 eu-central-1 eu-
west-1 eu-west-2 eu-west-3 eu-north-1 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-
southeast-1 ap-southeast-2 ap-south-1; do
  for key in $(aws kms list-keys --region $region --query 'Keys[].KeyId' --output text); do
    # Four fields, one word each: manager, key spec, origin, state.
    eligible=$(aws kms describe-key --region $region --key-id "$key" \
      --query 'KeyMetadata.[KeyManager,KeySpec,Origin,KeyState]' --output text | tr '\t' ' ')
    [ "$eligible" = "CUSTOMER SYMMETRIC_DEFAULT AWS_KMS Enabled" ] || continue
    aws kms enable-key-rotation --region $region --key-id "$key" --rotation-period-in-days 365
\
      || { echo "$region $key enable-key-rotation failed" >> rotation-failures.txt; continue; }
    [ "$(aws kms get-key-rotation-status --region $region --key-id "$key" --query
KeyRotationEnabled --output text)" = "True" ] \
      || echo "$region $key is not rotating" >> rotation-failures.txt
  done
done
test ! -s rotation-failures.txt && echo "every eligible key rotates" || cat rotation-
failures.txt
```

A key with rotation on by default (AWS CDK, TypeScript)

```typescript
import { Duration, Stack, StackProps, aws_kms as kms } from 'aws-cdk-lib';
import { Construct } from 'constructs';

export class DataKeyStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new kms.Key(this, 'DataKey', {
      alias: 'alias/sf-data',
      enableKeyRotation: true,
      rotationPeriod: Duration.days(365),
    });
  }
}
```

sf-protect-account-defaults, statement ProtectKeyRotation: nobody outside the platform roles may disable key rotation (the whole policy is in the guardrail SCPs) (Service control policy)

```JSON
{
  "Sid": "ProtectKeyRotation",
  "Effect": "Deny",
  "Action": [
    "kms:DisableKeyRotation"
  ],
  "Resource": "*",
  "Condition": {
    "ArnNotLike": {
      "aws:PrincipalArn": [
        "arn:aws:iam::*:role/PlatformAdmin",
        "arn:aws:iam::*:role/AWSControlTowerExecution",
        "arn:aws:iam::*:role/stacksets-exec-*"
      ]
    }
  }
}
```

How it is verified

- `rotation-failures.txt` is empty after the loop, and `aws kms get-key-rotation-status` returns `KeyRotationEnabled` true for every eligible key.
- Security Hub CSPM KMS.4 passes in every account after the next evaluation.
- Asymmetric keys have a documented replacement date in the platform runbook.

How to roll it back. Rotation can be turned off with `aws kms disable-key-rotation` after detaching the SCP, but there is rarely a reason to; rotated key material stays available for decryption either way. For an asymmetric key, point the alias back at the old key until the fleet has the new public key. Revert the CDK default last. Verify with `aws kms get-key-rotation-status`.

SF-SEC-054: EKS control-plane logging is off on 15 of 16 clusters, and every cluster still uses the aws-auth ConfigMap

Low. Benchmark rating Medium (Vantalect baseline: Kubernetes operations). How it was rated: benchmark Medium (2) -1 exploitability (theoretical) +0 blast radius (production) = 1

Where: 10 accounts in 5 Regions, 16 EKS clusters. **Controls:** No benchmark control covers this. **SOC 2:** CC7.2, CC8.1. **Owner:** Sam Okafor, Director, Fleet Cloud Engineering. **Repository:** `fleet-infra`. **Effort:** 24 hours. **Planned:** Week 5, starting Aug 10; the rest (Move every cluster to access entries and upgrade 1.31 and 1.32 clusters) is due November 13, 2026.

Evidence

- authenticationMode CONFIG_MAP on every cluster (the aws-auth ConfigMap is deprecated).

- 13 clusters on 1.31 or 1.32 are in extended support; 1.31 is upgraded automatically after 11/26/2026.

Examples: `telemetry-us-east-1-c37`, `telemetry-eu-central-1-c9d`, `telemetry-ap-northeast-1-732`.

Exploitability. A defense-in-depth gap with no direct path on its own. Slows detection and access review. **Blast radius.** Production systems or customer data. All production clusters.

How we verified it. Read from the scan.

Remediation: Turn on EKS control-plane logging, move to access entries, and upgrade the old clusters. Control-plane logging is off, clusters still authenticate through the deprecated `aws-auth` ConfigMap, and some clusters run Kubernetes 1.31 or 1.32, which are in extended support (EKS.2). Enable logging, migrate to access entries, and upgrade one minor version at a time, 1.31 first.

Steps

1. Enable all five control-plane log types on every cluster and set a retention period on the log groups.
2. Switch each cluster to `API_AND_CONFIG_MAP`, recreate every `aws-auth` mapping as an access entry with an access policy, remove the old mappings, and then set `API` (a one-way change).
3. Upgrade 1.31 clusters before November 26, 2026, when 1.31 reaches the end of extended support and EKS upgrades the control plane for you with no rollback.
4. Upgrade each cluster one minor version at a time, then its node groups and add-ons; in-place upgrades can be rolled back within 7 days.
5. Choose the target from the versions in standard support on the day you upgrade, and prefer the newest that your add-ons support; 1.34 leaves standard support on December 2, 2026.

Logging, then the access entry migration (AWS CLI)

```bash
CLUSTER=<cluster-name>
aws eks update-cluster-config --name $CLUSTER \
  --logging '{"clusterLogging": [{"types": ["api", "audit", "authenticator",
"controllerManager", "scheduler"], "enabled": true}]}'
aws logs put-retention-policy --log-group-name /aws/eks/$CLUSTER/cluster --retention-in-days
365

aws eks update-cluster-config --name $CLUSTER --access-config
authenticationMode=API_AND_CONFIG_MAP
kubectl get configmap aws-auth -n kube-system -o yaml > aws-auth-$CLUSTER.yaml
# Recreate each mapRoles entry with the access it had. Only a mapping in system:masters becomes
cluster admin;
# a mapping with other groups keeps those groups (the existing RoleBindings still match them);
node roles are skipped,
# because EKS creates their access entries when the node group is created (self-managed nodes
use --type EC2_LINUX).
kubectl get configmap aws-auth -n kube-system -o jsonpath='{.data.mapRoles}' | yq -o=json -I=0
'.[]' | while read -r mapping; do
  arn=$(echo "$mapping" | jq -r .rolearn)
  groups=$(echo "$mapping" | jq -r '(.groups // []) | join(" ")')
  case " $groups " in
    *" system:nodes "*|*" system:bootstrappers "*) echo "SKIP node role $arn"; continue ;;
    *" system:masters "*)
      aws eks create-access-entry --cluster-name $CLUSTER --principal-arn "$arn" --type
STANDARD
      aws eks associate-access-policy --cluster-name $CLUSTER --principal-arn "$arn" \
        --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy --
access-scope type=cluster ;;
    *)
      aws eks create-access-entry --cluster-name $CLUSTER --principal-arn "$arn" --type
STANDARD \
        ${groups:+--kubernetes-groups $groups} ;;
  esac
done
# mapUsers entries are IAM users: recreate the ones still needed the same way, and retire the
rest with the IAM user clean-up.
# After every mapping works as an access entry and the aws-auth entries are removed:
aws eks update-cluster-config --name $CLUSTER --access-config authenticationMode=API
```

Upgrade one minor version at a time, then the node group (AWS CLI)

```bash
CLUSTER=<cluster-name>
aws eks describe-cluster --name $CLUSTER --query cluster.version --output text
aws eks update-cluster-version --name $CLUSTER --kubernetes-version <next-minor-version>
aws eks wait cluster-active --name $CLUSTER
aws eks update-nodegroup-version --cluster-name $CLUSTER --nodegroup-name <nodegroup-name>
aws eks list-addons --cluster-name $CLUSTER --output text
```

How it is verified

- `aws eks describe-cluster --query cluster.logging` shows all five log types enabled, and `cluster.accessConfig.authenticationMode` is `API`.
- Security Hub CSPM EKS.2 passes: no cluster runs a version in extended support, and none is on 1.31 after November 26, 2026.
- Every engineer and pipeline that used an `aws-auth` mapping still reaches its cluster through its access entry.

How to roll it back. Logging can be turned off with `aws eks update-cluster-config --logging`, but leave it on. Moving authentication from `API_AND_CONFIG_MAP` to `API` is one-way, so keep `API_AND_CONFIG_MAP` until every mapping is verified; to undo one access entry, run `aws eks delete-access-entry`, which leaves its `aws-auth` mapping in force. An in-place upgrade can be rolled back within 7 days; after that, restore the workloads to a new cluster at the old version. Verify with the cluster version and the node status.

© 2026 Vantalect, LLC. Sample deliverable; StrataForge Robotics is fictional.

AWS, Amazon Web Services, AWS Graviton, and the AWS service names are trademarks of Amazon.com, Inc. or its affiliates. Kubernetes is a registered trademark of The Linux Foundation. GitHub is a trademark of GitHub, Inc. NVIDIA and Isaac Sim are trademarks of NVIDIA Corporation. SAP and SAP S/4HANA are trademarks of SAP SE. Okta is a trademark of Okta, Inc. PagerDuty is a trademark of PagerDuty, Inc. SOC 2 is a registered trademark of the American Institute of Certified Public Accountants. CIS Benchmarks is a trademark of the Center for Internet Security, Inc. Vantalect is not affiliated with or endorsed by any of them. StrataForge Robotics is fictional.