FIELD MANUAL · VOLUME II

The *build* is the breach.

SUPPLY CHAIN, SIGNING, AND THE PARTS YOU CAN'T SEE

Six lessons about the other half of cloud security: the pipeline that produced the thing you're running. Identity is how attackers get in. Supply chain is how they were already inside before you deployed.

Raajhesh Kannaa · April 2026

Volume I was about the front door. This is about the loading dock.

Most security programs spend ninety percent of their effort on runtime — who can log in, what can they do, what do we see — and almost nothing on the pipeline that builds and signs the artifacts running in production. Attackers figured this out about five years ago. We're still catching up.

The lessons in here came out of *Signed & Sealed*, which is a novel about a CI/CD compromise that spreads through a signing pipeline. If the book made you uncomfortable, this is where to put that feeling to work. If it didn't, I wrote it wrong.

ACT I

The pipeline.

You audit prod. The attacker owns the thing that ships to prod. Guess who wins.

01

Your CI has more power than your CTO.

GITHUB ACTIONS, OIDC, AND THE `CONTENTS:WRITE` PROBLEM

A GitHub Actions workflow runs with the permissions you give it. Almost nobody gives it the right ones. The default is too much. The common fix — federating into AWS via OIDC — is correct, and then half of teams attach a role that can do things the workflow has no business doing.

The pattern I see most: a deploy role with `s3:*` on the artifact bucket, `ecr:*` on every repo, and `iam:PassRole` on a wildcard. A compromised third-party action now has all of that, for as long as the OIDC token lives. Which is short, yes, but long enough.

WHAT THE WORKFLOW SHOULD SAY

```yaml
# .github/workflows/deploy.yml
permissions:
  id-token: write     # OIDC
  contents: read      # the default write is wrong

jobs:
  deploy:
    steps:
      - uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::...:role/deploy-specific-service
          aws-region: us-east-1
          role-session-name: gh-${{ github.run_id }}
```

WHAT THE ROLE SHOULD SAY

```json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "Federated": "arn:aws:iam::...:oidc-provider/token.actions.githubu
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
      "StringEquals": {
        "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
```

```
          "token.actions.githubusercontent.com:sub": "repo:your-org/your-repo:ref:refs/
        }
      }
    }]
  }
```

The `sub` claim is the piece everyone gets wrong. Without it, *any* workflow in *any* repo in your org can assume this role. Including the fork someone opened to demo a typo fix.

> **THE FORK PROBLEM**
>
> *Pull requests from forks can, with some workflows, run code under your repo's context. If that code can see OIDC tokens, it can assume your role. Scope `sub` to branches, not repos.*

REAL INCIDENTS

tj-actions/changed-files · **CVE-2025-30066** — a compromised GitHub Action leaked secrets from thousands of workflows. The fix was the same thing we should already have been doing: pin actions to a commit SHA, not a tag. · **Codecov 2021** — the bash uploader was modified in place; it extracted environment variables from CI and shipped them off.

02

Pin by SHA. Not by tag. Not by branch.

MUTABLE REFERENCES ARE A SUPPLY-CHAIN INVITATION

A tag is a label someone can move. A branch is obviously a moving target. A commit SHA is the one thing nobody can rewrite without producing a different SHA. This is the entire thing.

```
# wrong — the maintainer, or someone who compromised them, can point v4 anywhere.
- uses: actions/checkout@v4

# right — this cannot change under you.
- uses: actions/checkout@b4ffde65f46336ab88eb53be808477a3936bae11  # v4.1.1
```

The objection is always the same: "We'd have to update SHAs when new versions come out." Yes. Use Dependabot or Renovate with SHA pinning enabled. It opens a PR with the new SHA. You review it. You merge it. This takes less time than one incident.

> **APPLIES EVERYWHERE**
>
> *Same rule for container base images (pin digest, not tag), Terraform modules (pin commit), npm dependencies (lockfile committed and verified). Anywhere you're trusting code you didn't write, trust a specific version of it.*

03

Cache poisoning is a first-class attack now.

THE BUILD CACHE IS A WRITE SURFACE YOU PROBABLY FORGOT ABOUT

If your CI caches `node_modules`, the Go module cache, Docker layers, or a Bazel remote cache, that cache is writable by any job that can populate it. A malicious job writes a poisoned artifact under the cache key. Every subsequent job on that key reads it. Nothing got rebuilt. The diff in your PR is clean. Production got a different binary.

I don't have a clean detection for this. I'll say so. The best you can do is:

- Scope cache keys so jobs from different branches don't share them. The default in most CI systems is too wide.
- Treat the cache as untrusted input. If you're pulling a cached dep, verify its hash against a lockfile that's in the source tree.
- For anything that ships to production: do a fresh build from a clean cache at least weekly, compare the output, and raise hell if it differs.

> **THIS ONE KEEPS ME UP**
>
> *Cache poisoning is the quietest attack I know of against a modern pipeline. There's no log entry that says "a bad thing was cached." The only signal is that the built artifact differs from one you'd build from scratch. Most teams never compare.*

ACT II

Signing & verification.

You sign the artifact. Good. Now: who verifies, and where do they look?

04

An SBOM nobody verifies is a receipt for a crime.

SBOMS, SIGNATURES, AND THE STEP YOU PROBABLY SKIPPED

A software bill of materials lists what went into a build. Signing the SBOM and publishing it is table stakes now. The step most teams skip is *verifying the signature at deploy time*. If you don't verify, the SBOM is decorative. A determined attacker will produce a plausible one.

The pattern that works:

- Build produces artifact + SBOM + signature. Signing uses Sigstore, keyed to the workflow's OIDC identity.
- Registry (ECR, GHCR, whatever) stores all three.
- Deploy-time admission controller *verifies* the signature against an expected identity before letting the artifact run. In Kubernetes this is policy-controller or Kyverno.
- If verification fails, the pod doesn't start. This is the whole point.

ADMISSION POLICY EXAMPLE

```yaml
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: signed-by-our-ci
spec:
  images:
    - glob: "ghcr.io/your-org/*"
  authorities:
    - keyless:
        identities:
          - issuer: "https://token.actions.githubusercontent.com"
            subject: "https://github.com/your-org/your-repo/.github/workflows/release
```

> **THE QUIET FAILURE**
>
> *Most orgs roll out signing and forget verification. Six months later someone runs* `cosign verify` *on a random image and finds it's unsigned and nobody noticed. The signing side is the press release. The verifying side is the control.*

REAL INCIDENTS

SolarWinds 2020 — the build system itself was compromised. Signatures were valid; they were just signing malicious code. Signing without provenance is insufficient. · **3CX 2023** — a signed desktop app whose build was compromised via a prior supply chain compromise of a different vendor. Chain of trust, but the chain had a rotten link.

05

DNS is an exfil channel. Still. Forever.

ROUTE 53 RESOLVER LOGS & THE SHAPE OF A SLOW LEAK

If an attacker can resolve DNS from inside your environment, they can exfiltrate data over DNS. Encode the payload into subdomain labels, send queries to an authoritative server they control, reassemble on the other side. It's slow. It works. It bypasses most egress controls because DNS is the one thing that always has to work.

The detection isn't exotic. The signal is: very long subdomains, very high query volume to one domain, entropy in the labels.

DETECTION QUERY (ROUTE 53 RESOLVER LOGS)

```
SELECT query_name, COUNT(*) AS n, AVG(LENGTH(query_name)) AS avg_len
FROM resolver_logs
WHERE query_timestamp > current_timestamp - interval '1' hour
GROUP BY query_name
HAVING AVG(LENGTH(query_name)) > 40
    AND COUNT(*) > 100
ORDER BY n DESC;
```

Tune the thresholds to your environment. Legitimate CDNs and analytics vendors produce long subdomains. You'll need an allowlist. Building that allowlist is the actual work; the query is easy.

> **A BETTER CONTROL**
>
> *The real fix is a DNS firewall with an allowlist of domains your workloads are allowed to resolve. Everything else returns NXDOMAIN. This is uncomfortable to implement and very effective. Route 53 Resolver DNS Firewall and VPC DNS together cover most of the surface.*

06

IR under regulatory pause.

THE CLOCK STARTS WHEN LEGAL SAYS IT STARTS

This one isn't technical. It's the lesson the novel spends the most time on and the one practitioners know least about until they're in it.

When you have a confirmed breach with customer data involved, the incident-response clock you care about isn't the one in your runbook. It's the one your regulator cares about. GDPR gives you 72 hours from awareness. Some U.S. state laws are shorter. SEC rules for public companies are four business days after materiality determination. Your legal team will tell you when *awareness* or *materiality* is officially established — and that moment is when the clock starts, not before.

Practical consequences:

- Your IR documentation is now legal documentation. Write it like someone will read it in court. Timestamps, sources, decisions, who made them.
- There will be a period between "we think something happened" and "we officially know something happened" where your team needs to keep investigating but can't act in a way that constitutes confirmation. Your legal team will define the line. Ask them in advance.
- The attacker may still be in the environment during this period. You may be asked to watch and not evict. This is legal strategy and it is agonizing.

> **THE HARDEST PART OF THE JOB**
>
> *I've watched an IR lead, whose instinct was to pull the plug, hold off for nine hours at legal's request so that the forensics would meet evidence standards. He was right to do it. He didn't sleep for a week. The lesson is: the relationship with your GC matters before the incident, not during.*

REAL INCIDENTS

SEC v. SolarWinds 2023 — the SEC's case turned partly on internal communications about what was known and when. The paper trail mattered. · **Uber 2016 / 2022** — the 2016 breach and the way it was disclosed (or wasn't) led to a CSO criminal conviction in 2022. Disclosure isn't optional.

END OF VOLUME II

The pattern across all twelve lessons between the two volumes is: *the boring work is the work*. Pin your SHAs. Verify your signatures. Delete the old keys. Write the detection before you need it. Know your lawyer. Most of what we call "advanced" cloud security is just the basic work done with discipline over a long time.

The long-form is in *Signed & Sealed*. There is a person in that book whose job it is to hold the line during the regulatory pause. She is not having a good week. Neither is anyone around her. That part is accurate.

Set in IBM Plex Mono and Source Serif Pro. Same system as Volume I, one book apart.

Raajhesh Kannaa · Toronto · defensive.works
Cloud Security Cinematic Universe · Book 02 · Field Manual II · April 2026

— END OF FIELD MANUAL II —