

Securing Third-Party GitHub Actions at Scale

Onboard, adopt and monitor them without slowing teams down

Cloud Solution Architect · Microsoft

Personal views, not official Microsoft guidance

Every **uses:** line is someone else's code

- It runs **next to your source code, your token and your secrets**.
- "Third-party" means anything outside your enterprise: GitHub's own `actions/*`, Microsoft and Azure actions, everything on the Marketplace.
- ...and whatever those pull in at run time: **nested actions, container images, downloaded tools**.
- Most teams add them without a second thought.

Goal: give every action that second thought, without slowing anyone down.

March 2025: `tj-actions/changed-files`

- An attacker **moved the action's version tags** to a malicious commit, after an earlier compromise of `reviewdog/action-setup`.
- Every workflow using a tag ran the payload on its next run. **More than 23,000 repositories** used the action.
- The payload read the runner's memory and printed secrets, double base64-encoded, into the workflow logs. In public repositories, anyone could read them.

March 2026, Trivy: teams that had pinned `aquasecurity/trivy-action` to an older SHA were still exposed, because that commit loaded a nested action **by tag**.

Tags move. Commits don't. And a pin stops a reference from moving; it doesn't make the code safe.

What went wrong, and what was missing

What went wrong	What was missing
A moved tag delivered malicious code	An allowlist and SHA pinning
Nobody had ever looked at the action	A review before approval
The job held secrets it didn't need	Least privilege, and no secrets near third-party code
One token could write to every repository, one secret could reach Azure	Short-lived, scoped tokens and OIDC
Nothing limited where the payload could send data	Egress-controlled runners
Days to find every usage	An inventory with owners
The first alert came a day late	Continuous monitoring
Teams added any action they liked	A fast, approved route
A day to block, weeks to rotate	A rehearsed response plan

The principles (1/2)

1. **Make the secure path the fastest path.** Self-service requests, a catalog and published turnaround times.
2. **Only run what you've approved, at exact SHAs.** An enforced allowlist. Trusted publishers are the one deliberate exception.
3. **Pin everything to a full commit SHA.** `uses: owner/repo@<40-char-sha> # v2.4.1`, enforced by policy.
4. **Review the code that actually runs.** `action.yml`, `dist/`, `pre` and `post` hooks, nested actions, images and downloads. Not the README.
5. **Keep tokens and secrets on a short leash.** Read-only by default, `permissions: {}`, and no secrets near third-party code.

The principles (2/2)

6. **Lock down the runner, with the team that runs it.** Ephemeral, default-deny egress, and every new destination approved like a secret.
7. **Know what you run, and who owns it.** A catalog, an inventory and a sponsor team for every action.
8. **Keep watching**, trusted publishers included. Dependabot doesn't alert on SHA-pinned actions, so build your own daily check.
9. **Let every approval expire.** Old SHA: 3 months after a newer version, 30 days for High, the same day for Critical.
10. **Be ready for the day an action goes bad.** Block with `!owner/repo@*` in minutes, rotate everything, rehearse.

Platform runs the process. The requesting team sponsors it. Security watches what runs. Infra owns the runners and egress. None of them can do it alone.

The allowlist in practice

Actions policy: **Allow enterprise, and select non-enterprise, actions and reusable workflows**, enforced at enterprise level. **Verified creators: off**. A badge tells you who published an action, not that it's safe.

Entry	What it's for
<code>owner/repo@<full-sha></code>	One approved SHA from an untrusted publisher
<code>owner/repo@*</code>	Every SHA of one repository from a trusted publisher
<code>!owner/repo@*</code>	Blocking a compromised action, whatever else allows it
<code>!owner/repo@<full-sha></code>	Blocking one bad SHA from a trusted repository

- **Avoid** `owner/*`. It trusts every repository that owner has today or creates tomorrow.
- **Trusted publishers are a trade-off**. Fewer entries, but unreviewed future SHAs get in. Balance with pinning, daily monitoring and a fast block.
- **1,000 entries is a hard limit**. Generate the list from the catalog, keep at most two live SHAs per untrusted action, alert early.

When an action needs a new destination

A deployment to a third-party vendor, for example. The Infra team that runs the runners is part of the decision.

- **Treat a network destination like a secret.** Declare it in the request, approve it, and record it in the catalog's `network:` field.
- **Scope rules to a runner group, not the shared pool.** For example, a `deploy-northwind` group that only the repositories deploying to that vendor can use.
- **Allow exact hostnames.** No wildcards on shared hosting like `*.amazonaws.com` or `*.blob.core.windows.net`: attackers can host there too.
- **Find out what the action actually calls.** During onboarding, run it on a test runner that logs traffic without blocking it, and compare that with what the request declared.
- **Split the job.** The third-party action runs with no secrets and no extra network. A separate deploy job uses the vendor's CLI, with OIDC where the vendor supports it, and can reach only the vendor.

The catalog is the source of truth. The Infra team, or automation, turns it into egress rules.

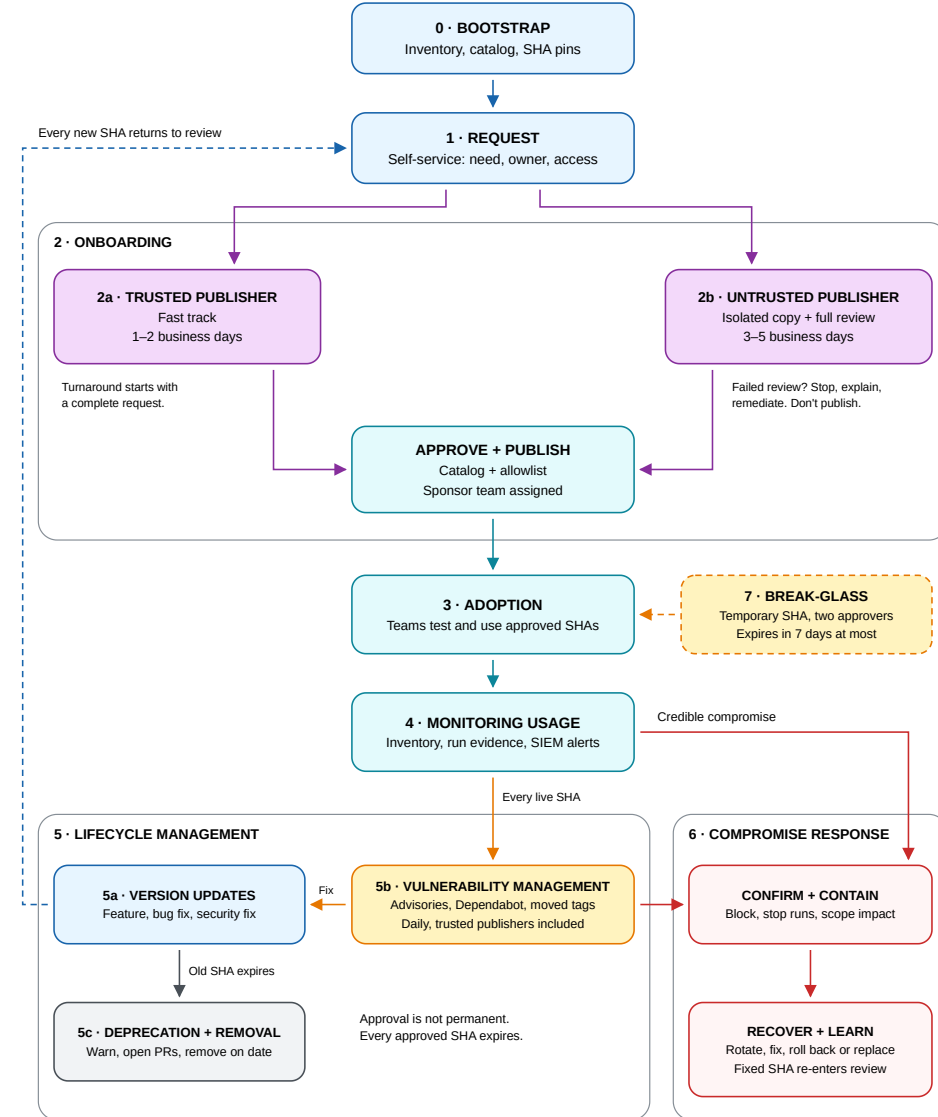
One lifecycle, clear owners

- 0 • **Bootstrap**: inventory, catalog, pin, enforce
- 1 • **Request**: a dev team asks, and sponsors it
- 2 • **Onboarding**: 2a trusted fast track · 2b full review
- 3 • **Adoption**: use the approved SHA, within its limits
- 4 • **Monitoring**: what workflows declare vs. what actually ran
- 5 • **Lifecycle**: updates, vulnerabilities, retirement
- 6 • **Compromise response**: block, scope, rotate, recover
- 7 • **Break-glass**: time-boxed exceptions, two approvals

Every new SHA goes back through review, whether it's a new action, an update or a security fix.

Approval is the start, not the finish.

Third-party GitHub Actions | Steps 0–7 match the guide | Two review lanes, one lifecycle



Platform team owns the process and tooling. Dev teams sponsor, adopt and remediate.

Four teams, one process

Step	Platform team	Dev teams	Security team	Infra team
0 · Bootstrap	Inventory, catalog, enforcement	Merge the pinning pull requests	–	Ephemeral runners, baseline egress allowlist
1 · Request	Triage	Ask, and explain why	Risky requests	–
2 · Onboarding	Run the checks, decide, publish	Review and test	Risky requests	New destinations, egress rules
3 · Adoption	Catalog and policy check	Use as approved	–	–
4 · Monitoring	Inventory, run evidence, dashboards	Fix flagged workflows	SIEM detections	Blocked-egress logs to the SIEM
5 · Lifecycle	Updates, daily checks, expiry dates	Request updates, merge pull requests	Critical findings	–
6 · Compromise	Confirm, contain, scope, track	Rotate their secrets	Investigate	Block the destination, isolate the runner group
7 · Break-glass	Approve and expire	Ask	Co-approve	–

If a step has no owner, it won't happen. If the owners don't talk, it won't work.

Detect and contain a leaked token

Detect

- Every `git.clone` in the audit log stream carries the actor, the IP address and a hash of the token.
- A Microsoft Sentinel rule runs every 5 minutes and flags any token that cloned **20 or more repositories in 15 minutes**.

```
let threshold = 20;
GitHubAuditLog_CL
| where TimeGenerated > ago(15m)
| where Action == "git.clone"
| extend HashedToken =
    tostring(parse_json(RawEvent).hashed_token)
| summarize Repos = dcount(Repository)
    by Actor, AccessType, HashedToken
| where Repos >= threshold
```

Contain, automatically

- **Cut the token off:** suspend the GitHub App installation, so none of its tokens work.
- **Gather the evidence:** repositories, IP addresses, and the jobs that received the bot's credentials.
- **Call a person**, who lifts the suspension or starts the compromise response.

In Contoso's drill, the rule fired **6 minutes** after the first clone. By then, 64 repositories were gone.

Automate only what's cheap to undo. Detection limits the damage; it doesn't prevent it.

Settings checklist (1/2)

Policy

- ☐ Actions policy: **Allow enterprise, and select non-enterprise, actions and reusable workflows**, enforced
- ☐ **Allow Marketplace actions by verified creators**: off
- ☐ Allowlist generated from the catalog, with alerts on manual edits
- ☐ **Require actions to be pinned to a full-length commit SHA**: on

Permissions

- ☐ **Workflow permissions**: read repository contents and packages
- ☐ **Allow GitHub Actions to create and approve pull requests**: off
- ☐ A required policy check on every pull request, through an organization ruleset

GitHub Advanced Security

- ☐ CodeQL (including the `actions` language), secret scanning with push protection, and Dependabot, wherever third-party code is reviewed or used

Settings checklist (2/2)

Runners

- ☐ Ephemeral runners by default
- ☐ Default-deny egress on self-hosted runners
- ☐ Self-hosted runner groups limited to selected organizations and repositories
- ☐ No sensitive secrets in third-party jobs on standard GitHub-hosted runners
- ☐ Every network destination in the catalog has an egress rule, scoped to a runner group
- ☐ Blocked-egress alerts go to the platform, security and Infra teams

Monitoring

- ☐ Audit log streamed to your SIEM, with detection rules, kept for your incident look-back window
- ☐ Daily checks: advisories, Dependabot on reviewed copies, moved tags
- ☐ A catalog with a sponsor and an expiry date for every SHA

Where to start

1. **Take an inventory.** What workflows declare, with tags resolved to SHAs and nested actions included, plus what ran, from job logs. Note the SHA and whether the job holds secrets.
2. **Sort it.** Trusted publisher: allow the repository. Untrusted and in use: review the SHA in use. Unused or duplicated: drop it.
3. **Seed the catalog and the allowlist.** One SHA per untrusted action, and a sponsor for each.
4. **Pin.** Automation opens pull requests that rewrite every `uses:` line to a full SHA.
5. **Enforce, one step at a time.** Policy check in warning mode, then the allowlist, then SHA pinning, then blocking. Announce each step a month ahead.

Approval is the start, not the finish.

Thank you. Questions?

The full guide: rmoreirao.github.io/securing-third-party-github-actions

The principles and settings, plus the Contoso story: who does what, request forms, an onboarding checklist with criticality levels, catalog schema, expiry rules and the incident runbook.

Contoso and Fabrikam are fictional. Licensed under CC BY 4.0.