SecurityRisk ADVISORS

An OffSec Adventure through CI/CD

Jonn Callahan // atticuss

- AppSec, red teams, and engineering
  - One of those freaks that actually enjoys reading code
  - DevRedOps: I *will* make this a thing

- Rock climber
- Metal head
- Dog lover
- Reformed cryptocoin miner

SRA

Overview

- More code being written -> more automated systems to handle them

- Complex build & deployment requirements demand complex systems
  - Internal build dependencies from libraries owned by other teams
  - External build dependencies from open-source libraries
  - App -> Docker -> registry -> helm chart -> EKS -> EC2 fleet
  - Microservice orchestration

- Many orgs have shifted away from Jenkins (we don’t talk about Jenkins X)
  - Goodbye easy script console pivot points ☹

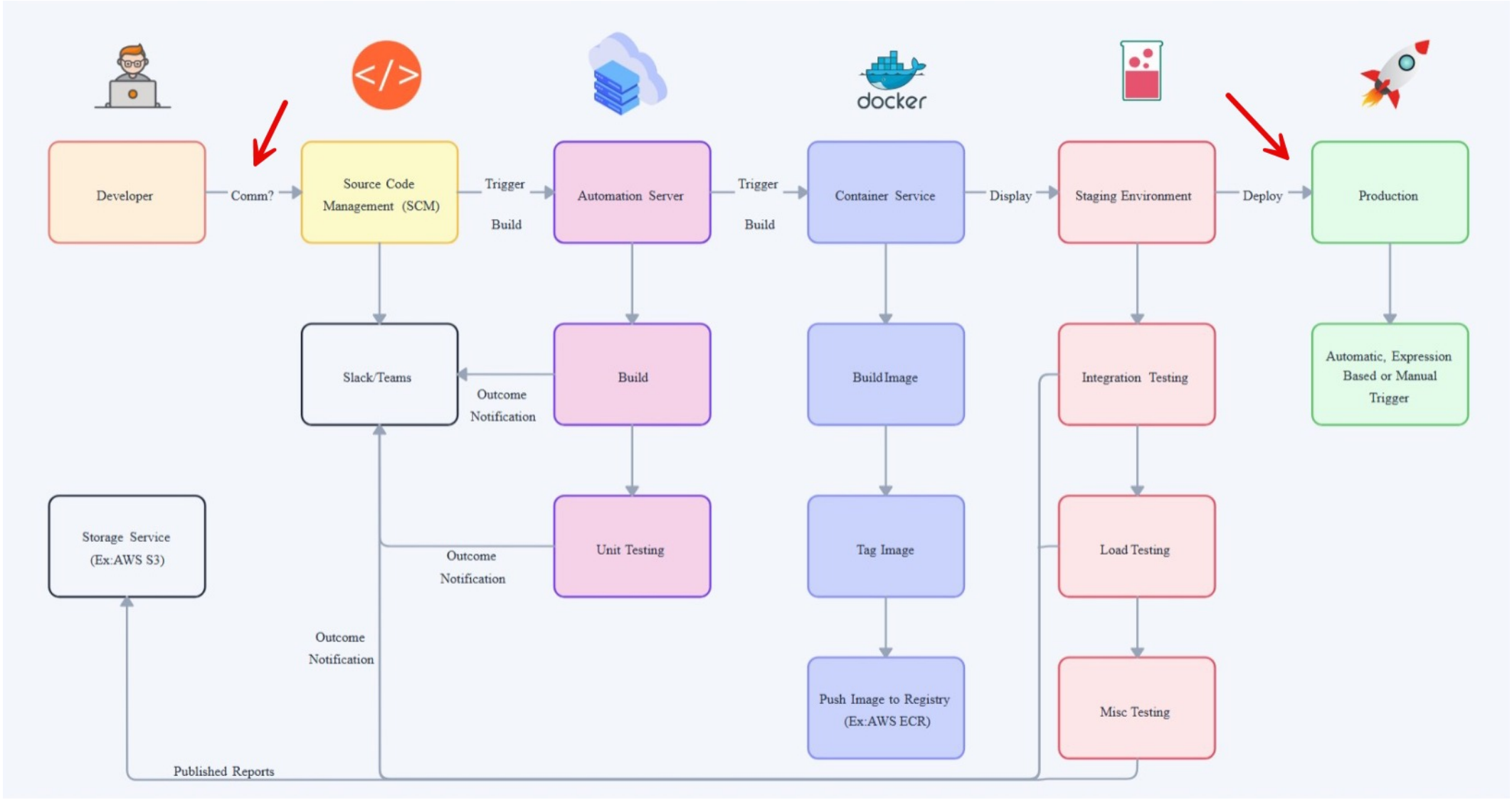
SRA

Public vs Private

- Hasn’t this been talked to death?
  - Yes, but primarily in the context of public repositories for open-source projects
- Many resources focus very heavily on the impact of malicious pull requests
  - “Pwn requests”
  - Dangerous CI/CD triggers
  - Default token scopes
- Far less focus on the impact of a compromised developer
  - Even in this context, much of the focus is on compromised public projects
  - (lol that npm ecosystem tho)

SRA

A Simple Workflow



Related Teams

- Ops/SRE do not have write-access to prod
- IT admins do not have access to Domain Admin
- Cloud admins do not have access to Global Administrator
- PIM, PAM, break-glass; pick your poison
  - Daily driver accounts are limited in access
  - Highly privileged accounts are gated behind strongly audited systems, and typically wired into alerting

SRA

Developers

- CI/CD systems have admin (or near-admin) privileges into target deployment environments
- Developers can interact with CI/CD systems
- Ergo developers have admin access in production
  - Hello modern environment take overs!

SRA

Attacker Context

- Compromised developer account
  - C2 running on developer machine
  - Stolen credentials with attacker-controlled MFA device
  - Personal access token or SSH key
  - Insider threat / assumed compromise scenario

- Honorable mention: malicious dependency
  - Targeted attacks via dependency confusion
  - Broad attacks via compromised third-party libraries

SRA

CI/CD Definitions

- Every CI/CD system (that I've seen) is configured via a repository-housed YAML
  - GitHub: .github/workflows/foo.yml
  - Gitlab: .gitlab-ci.yml
  - ADO: foo.yml (no name or path restrictions)

- Configuration files map events to workflows, e.g.:
  - On pull request open -> create Jira ticket and update description with link
  - On merge to main -> build Docker image and push into container repository

SRA

Example CI/CD Definition

```yaml
name: Build App

on:
  push:
    branches:
      - main

jobs:
  build-app:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: ${{ secrets.AWS_ROLE_ARN }}
          aws-region: us-east-1

      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v2

      - name: Build, tag, and push docker image to Amazon ECR
        env:
          REGISTRY: 001122334455.dkr.ecr.us-east-1.amazonaws.com
          REPOSITORY: my-ecr-repo
          IMAGE_TAG: ${{ github.sha }}
        run: |
          docker build -t $REGISTRY/$REPOSITORY:$IMAGE_TAG .
          docker push $REGISTRY/$REPOSITORY:$IMAGE_TAG
```

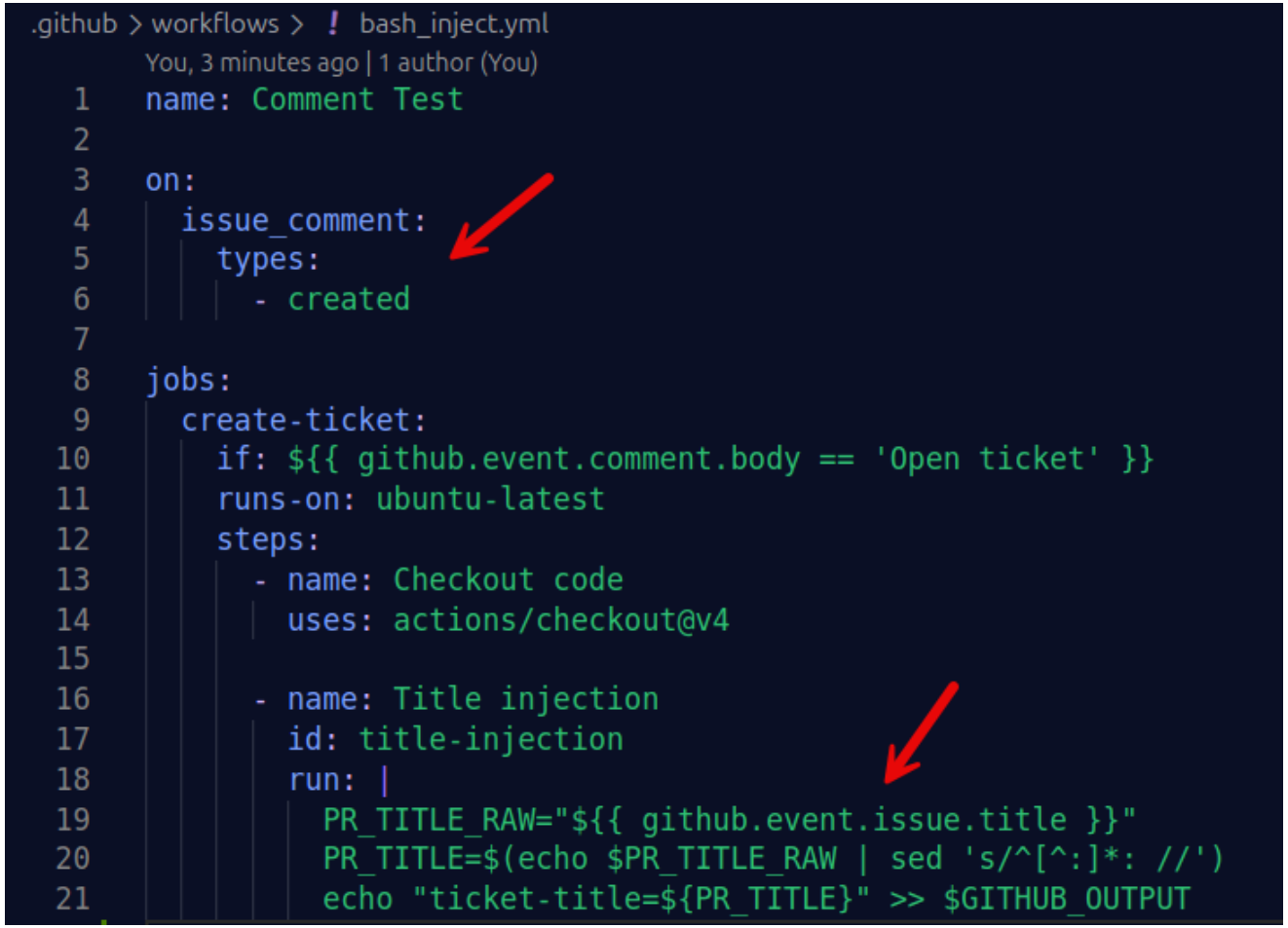
Pull Request Injection

- Talked about a lot within the context of public repositories
  - Usually under the term “pwn request”

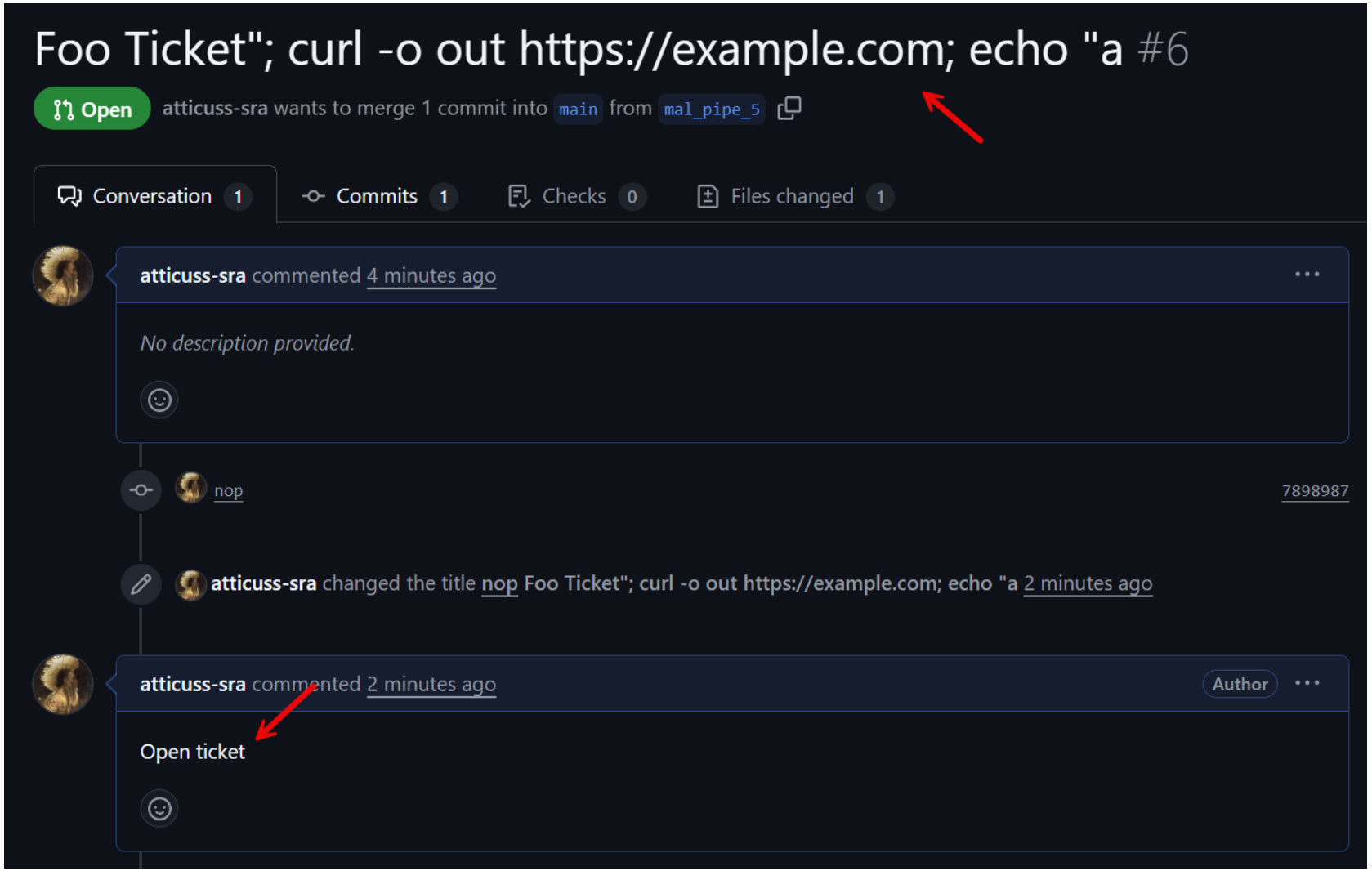
- Not quite as useful within internal organizations
  - But that doesn’t mean it’s useless!
  - Modify title on PR -> trigger pipeline -> revert PR title
  - if: ${{ steps.check-author.outputs.is-dependabot == 'true' }}

SRA

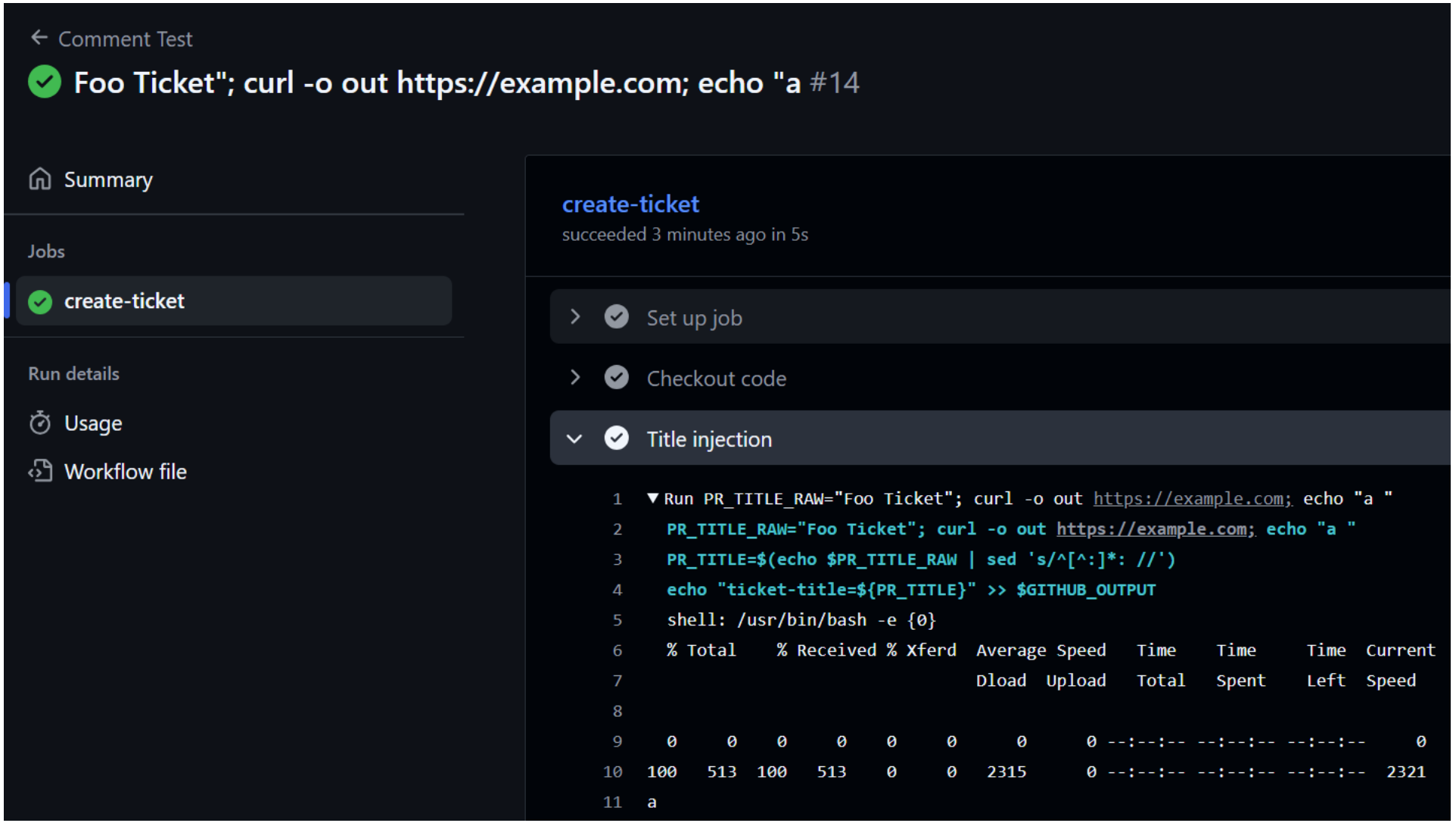
Pull Request Injection



Pull Request Injection



Pull Request Injection



Secrets Management

- CI/CD systems can house secrets, with various access control systems
  - GitHub can scope to specific repositories or an entire organization
  - GitLab and ADO can scope to specific repositories or projects
- Improperly scoped secrets -> violation of the principle of least privilege
- Secrets containing authentication data should be avoided when possible
- Alternatively, can use traditional third-party secret storage mechanisms
  - E.g., GitHub OIDC -> Hashicorp Vault

SRA

Secrets Management Exploitation

```
[14-Nov-25 17:09:28] [Helios] [ens33: 192.168.221.129]
 ~  > gh search code --owner=google "\"\${{ secrets.\" extension:yml"

Showing 30 of 346 results

google/go-cloud .codecov.yml
        #     run: codecov -t ${{ secrets.CODECOV_TOKEN }}

google/guava .github/workflows/ci.yml
        CI_DEPLOY_USERNAME: ${{ secrets.CI_DEPLOY_USERNAME }}
        CI_DEPLOY_PASSWORD: ${{ secrets.CI_DEPLOY_PASSWORD }}

google/flatbuffers .github/workflows/label.yml
        repo-token: "${{ secrets.GITHUB_TOKEN }}"

google/gvisor .github/workflows/go.yml
        if ! [[ -z "${{ secrets.GO_TOKEN }}" ]]; then
        -H "Authorization: token ${{ secrets.GITHUB_TOKEN }}" \

google/zx .github/workflows/publish.yml
        GOOGLE_NPM_TOKEN: ${{ secrets.AUTH_TOKEN }}
        GH_NPM_TOKEN: ${{ secrets.GITHUB_TOKEN }}

google/filament .github/workflows/postsubmit.yml
        GH_TOKEN: ${{ secrets.FILAMENTBOT_TOKEN }}
        GH_TOKEN: ${{ secrets.FILAMENTBOT_TOKEN }}

google/gson .github/workflows/scorecard.yml
        # repo_token: ${{ secrets.SCORECARD_TOKEN }}

google/oss-fuzz .github/workflows/cflite_pr.yml
        github-token: ${{ secrets.GITHUB_TOKEN }}
        # storage-repo: https://${{ secrets.PERSONAL_ACCESS_TOKEN }}@github.com/OWNER/STORAGE-REPO-NAME.git

google/tsunami-security-scanner .github/workflows/full-push.yml
        password: ${{ secrets.GITHUB_TOKEN }}
```


SRA

Secrets Management Exploitation

.github > workflows > ! secret_enum.yml

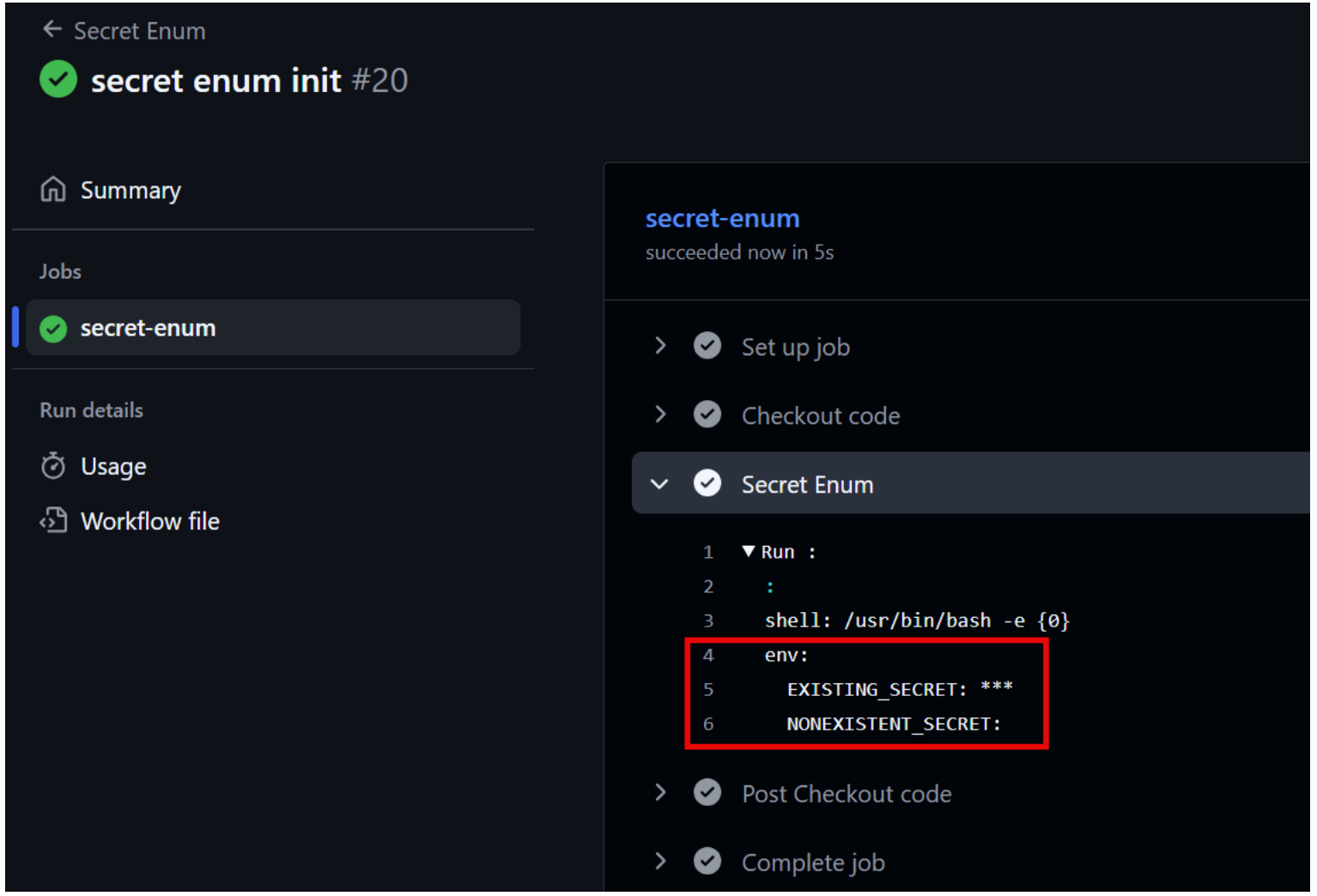
```yaml
name: Secret Enum

on:
  - push

jobs:
  secret-enum:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Secret Enum
        id: secret-enum
        env:
          EXISTING_SECRET: ${{ secrets.EXISTING_SECRET }}
          NONEXISTENT_SECRET: ${{ secrets.NONEXISTENT_SECRET }}
        run: |
          :
```

Secrets Management Exploitation



Self-Hosted Runners

- Allows workflows to execute on private infrastructure
  - Virtual machine, Kubernetes pod, ECS task, etc.
- Can be ephemeral or non-ephemeral
  - Non-ephemeral are impossible to secure
- Needs a means to authenticate into private resources
  - CI systems need to store artifacts (e.g., Docker images)
  - CD systems need to deploy to an environment (e.g., an AWS environment)
- Can be restricted in their use, such as limiting to a specific set of repositories

SRA

Self-Hosted Runner Authentication

- Pre-configured authentication
  - Authentication is tied to the runner (e.g., an EC2 with an IAM role)
  - Repository context is lost
  - Reused runners must then have access to all potential resources (e.g., multiple AWS accounts)

- Repository OIDC configuration
  - Authentication is tied to the repository
  - Reused runners can be restricted to the authentication required for a specific job
  - Far better adherence to the principle of least privilege
  - Limited to authentication providers that support it

SRA

Self-Hosted Runner Ephemerality

- Non-ephemeral are installed via bash script
  - Simple control loop: poll for work -> run job -> push results -> poll for work
  - Post-job cleanup scrubs the working directory
  - Trivial to obtain persistence

- Ephemeral workers run jobs in an isolated context
  - Kubernetes: schedule jobs as isolated pods
  - AWS ECS: schedule jobs as isolated tasks
  - Post-job cleanup scrubs the entire environment
  - Extremely difficult to obtain persistence

SRA

Non-Ephemeral Exploitation

.github > workflows > ! self_hosted.yml

You, 3 minutes ago | 1 author (You)

```yaml
name: Self Hosted

on:
  - push

jobs:
  self-hosted:
    runs-on: self-hosted
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Run Command
        id: run-command
        run: |
          touch /tmp/canary
```

```
ubuntu@ip-172-31-27-100:~/actions-runner$ ./run.sh

√ Connected to GitHub

Current runner version: '2.329.0'
2025-11-14 12:43:49Z: Listening for Jobs
2025-11-14 12:47:19Z: Running job: self-hosted
2025-11-14 12:47:23Z: Job self-hosted completed with result: Succeeded
^CExiting...
Runner listener exit with 0 return code, stop the service, no retry needed.
Exiting runner...
ubuntu@ip-172-31-27-100:~/actions-runner$ ls -al /tmp/canary
-rw-rw-r-- 1 ubuntu ubuntu 0 Nov 14 12:47 /tmp/canary
ubuntu@ip-172-31-27-100:~/actions-runner$
```


SRA

Non-Ephemeral Exploitation

.github > workflows > ! self_hosted.yml

You, 11 minutes ago | 1 author (You)

```yaml
name: Self Hosted

on:
  - push

jobs:
  self-hosted:
    runs-on: self-hosted
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Run Command
        id: run-command
        env:
          EXISTING_SECRET: ${{ secrets.EXISTING_SECRET }}
        run: |
          sleep 60
```


SRA

Non-Ephemeral Exploitation

```
ubuntu@ip-172-31-27-100:~/actions-runner$ ps aux | grep run
root         632  0.0  2.1  32420 20608 ?        Ss   03:24   0:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
root        1053  0.0  0.8  12020  7920 ?        Ss   03:25   0:00 sshd: /usr/sbin/sshd -D -o AuthorizedKeysCommand /usr/share/ec2-instanc
e-connect/eic_run_authorized_keys %u %f -o AuthorizedKeysCommandUser ec2-instance-connect [listener] 0 of 10-100 startups
ubuntu      2973  0.0  0.2   9000  2644 ?        Ss   12:53   0:00 SCREEN -d -R runner
ubuntu      2983  0.0  0.3   7740  3496 pts/1    S+   12:53   0:00 /bin/bash ./run.sh
ubuntu      2987  0.0  0.3   7740  3560 pts/1    S+   12:53   0:00 /bin/bash /home/ubuntu/actions-runner/run-helper.sh
ubuntu      2991  0.6 12.3 273702088 121244 pts/1 Sl+ 12:53   0:03 /home/ubuntu/actions-runner/bin/Runner.Listener run
ubuntu      3846 10.9 12.1 273711220 118744 pts/1 Sl+ 13:01   0:01 /home/ubuntu/actions-runner/bin/Runner.Worker spawnclient 154 160
ubuntu      3980  0.0  0.3   7744  3468 pts/1    S+   13:02   0:00 /usr/bin/bash -e /home/ubuntu/actions-runner/_work/_temp/882d772d-930a-
4d3d-bf1d-976cface4158.sh
ubuntu      3990  0.0  0.2   7076  2076 pts/0    S+   13:02   0:00 grep --color=auto run
ubuntu@ip-172-31-27-100:~/actions-runner$ sudo grep -z "EXISTING_SECRET=" /proc/3980/environ
EXISTING_SECRET=abcubuntu@ip-172-31-27-100:~/actions-runner$
```

Dependency Confusion

- Build systems will regularly rely on private packages
  - E.g., a private library hosted within Artifactory
- Attackers can enumerate private packages names and register them on public repositories
- An improperly configured build system may then opt for the public variant over the private one

SRA

Dependency Confusion Exploitation

- An organization has an “@acme/widget-lib” NodeJS library stored within Artifactory
  - Latest published version is 0.3
- This organization also has a private NodeJS application that relies on this library
  - “@acme/widget-lib”: “^0.3”
- Attacker registers the “@acme” namespace on npmjs.com
- Attacker publishes a malicious “@acme/widget-lib” package, version 0.4
- Artifactory is configured to serve upstream packages, sees that 0.4 is available on npmjs.com, and returns this version to the build system

SRA

Mutable Artifact Stores

- Mutable -> overwrite
- Especially pernicious for container image repositories
- Common pattern is to elevate code by copying container images from development into production environments
  - Development team opens PR on production infra repository
  - Attacker runs a malicious pipeline and overwrites deployment environment repository
  - Infra repository PR approved & merged
  - Infra pipeline copies image from development into production
  - Et voila

SRA

A Word on Covert Exploitation

- Let’s assume a perfectly hardened environment:
  - The self-hosted runner is ephemeral, tightly scoped to a specific repo, etc.
  - Secrets are tightly scoped to a specific repository
  - Branch protections are properly configured

- An attacker can still trivially execute code
  - Branch target repository
  - Introduce malicious CI/CD configuration file with the appropriate trigger
  - Create PR* or push branch to trigger workflow execution

- *when creating a PR, the CI/CD definitions within the **source** branch are respected

SRA

Repository Creation

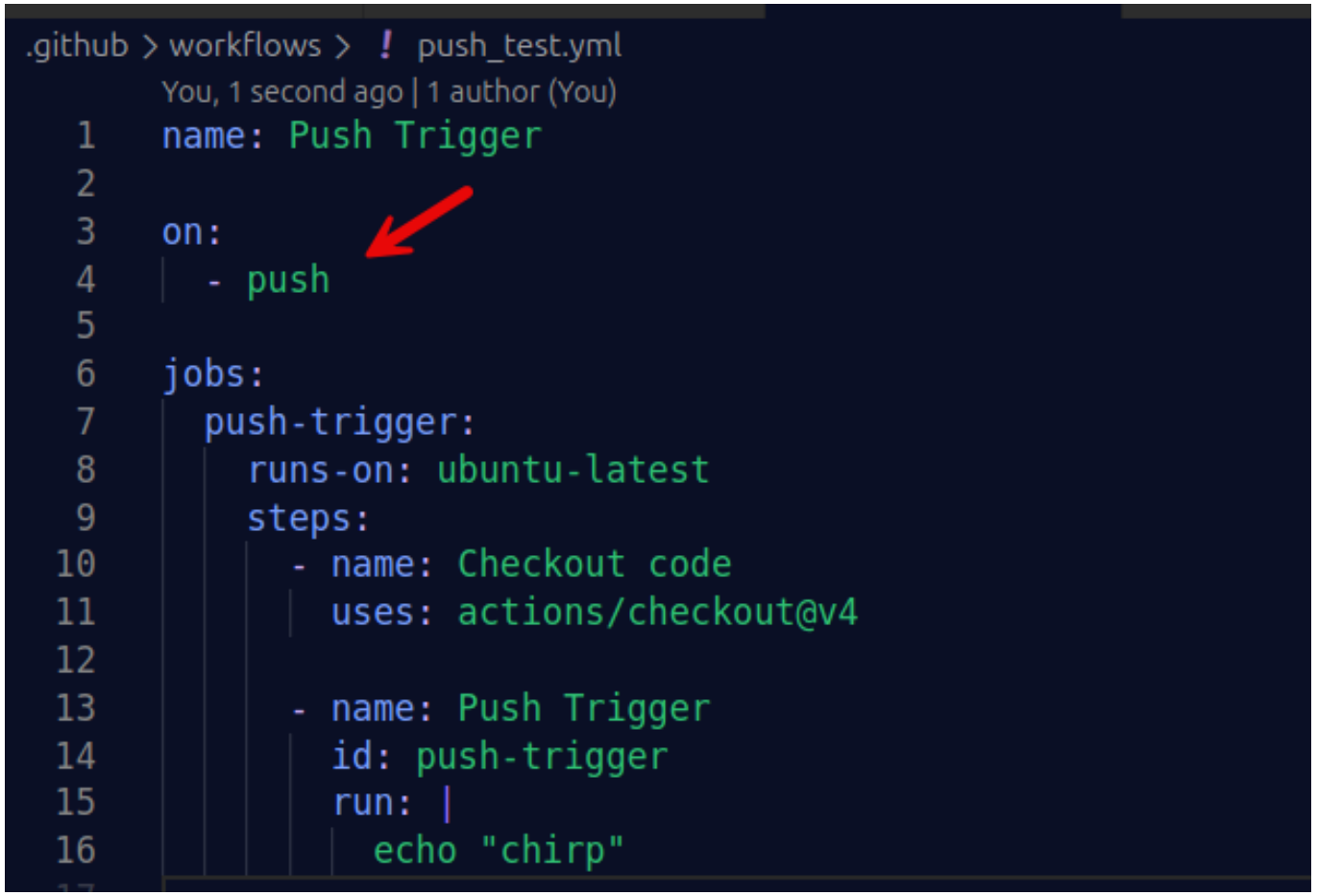
- PRs and pipeline execution create a lot of noise
  - Stakeholders notified (it's someone's job to review them)
  - PRs cannot be deleted, only closed or archived (GitLab is the exception)
  - Pipeline execution logs are high fidelity (ADO is the exception)

- Easy & covert exploitation of malicious pipelines
  - Unlikely to trigger automation or to be accidentally noticed

- Requires lax restrictions on privileges resource to fully leverage
  - Self-hosted runners (runners themselves & authentication setup)
  - Secrets

SRA

On Push Covert Execution



Attack Vectors Summary

- Pull request injection
  - Frequently referred to as “pwn requests”
  - Frequently discussed within the context of public repositories
- CI/CD secret exposure
  - Platforms allow for secrets to be scoped to limit their exposure
- Self-hosted runners
  - Typically runs with full/near-admin access to production environments
- Dependency confusion
  - Discover vulnerable instances with private repo read-access
  - Introduce vulnerable instances with write-access
- Unprotected reusable CI/CD configurations & resources, e.g.:
  - Prod Repository A references a script housed Utility Repository B
  - Utility Repository B does not have branch protections
- Repository creation & “on push” triggers
  - Covert malicious pipeline execution

A Call to Arms

- Developers are far from the only team with direct lines to production environments
- Unique in their lack of oversight or audit controls
- The industry is fundamentally behind in this aspect
  - Too focused on public repository exploitation and compromised open-source repositories
  - These are important (and embarrassingly common) problems
  - But far from the only problem!
- Wiz has an identity solution in this space
  - Never worked with it so can’t speak to its efficacy
  - https://www.wiz.io/blog/developer-infrastructure-security-posture-and-threat-detection

SRA

SecurityRisk ADVISORS

Questions?