Securing the Pipeline

Protecting Self-Hosted GitHub Runners

praetorian

April 23rd, 2023

About Me

- Adnan Khan - Lead Security Engineer at Praetorian
- **ShmooCon 2023 Speaker**
  - Phantom of the Pipeline: Abusing Self-Hosted CI/CD Runners

  github.com/praetorian-inc/gato

- Second conference talk!
- Based in Maryland now, but originally from the East Bay Area ☺

@adnanthekhan

Why this talk?

PEPE
SILVIA
DEVOPS
imgflip.com
FX

DevOps is Hard

- *Chaotic* is a word that could accurately describe many private GitHub organizations
  - Hundreds or thousands of repositories
  - Individual developers adding API tokens as repo secrets
  - Self-hosted runners frequently used for integration
  - Internal default repository permissions where every developer can create branches

Presentation Overview

- GitHub Actions CI/CD Primer
  - Self-Hosted Runners
  - API
  - Tokens
  - Secrets
- Risks and Attacks
- Detection techniques and mitigations for different risks and attack types
- Future Considerations
- Questions

GitHub Actions & CI/CD

GitHub Actions is unique in its integration with GitHub at large.

Evolution from a **focus on OSS** and small teams to **Enterprise SaaS platform** -> Conflict of features, Gaps in Security

Build agents can be **GitHub-Hosted** (in Azure) or **Self-Hosted**

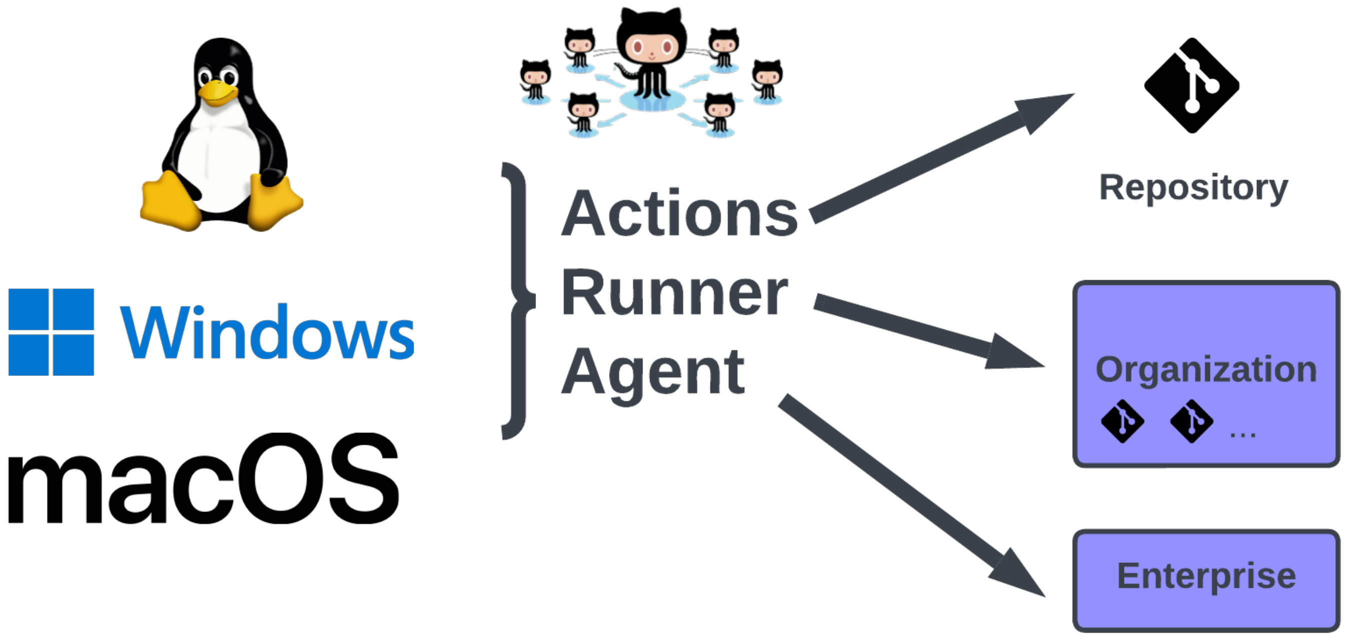
GitHub Actions also allow supplying secrets at a per-repository or per-organization level

Other CI/CD Solutions:

Bamboo

GitLab

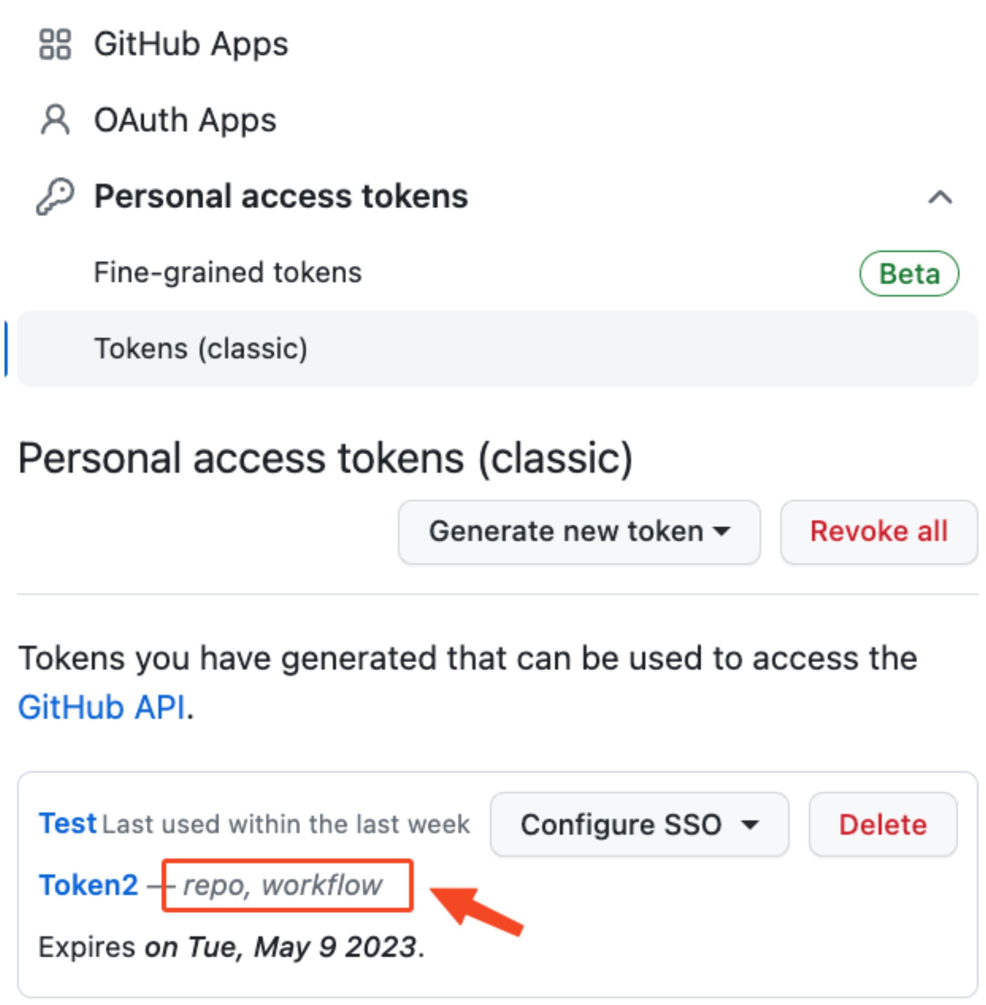
Self-Hosted Runners Primer



- Attached to a repository, organization, or enterprise
- The default configuration is insecure
    - **Non-ephemeral**
    - **Builds are not isolated**

GitHub Personal Access Tokens (PATs)

- Associated with **user accounts**
- Used for GitHub API access
- Can be used for git clone/push operations
- OAuth scopes control access



Personal access tokens (classic)

Tokens you have generated that can be used to access the GitHub API.

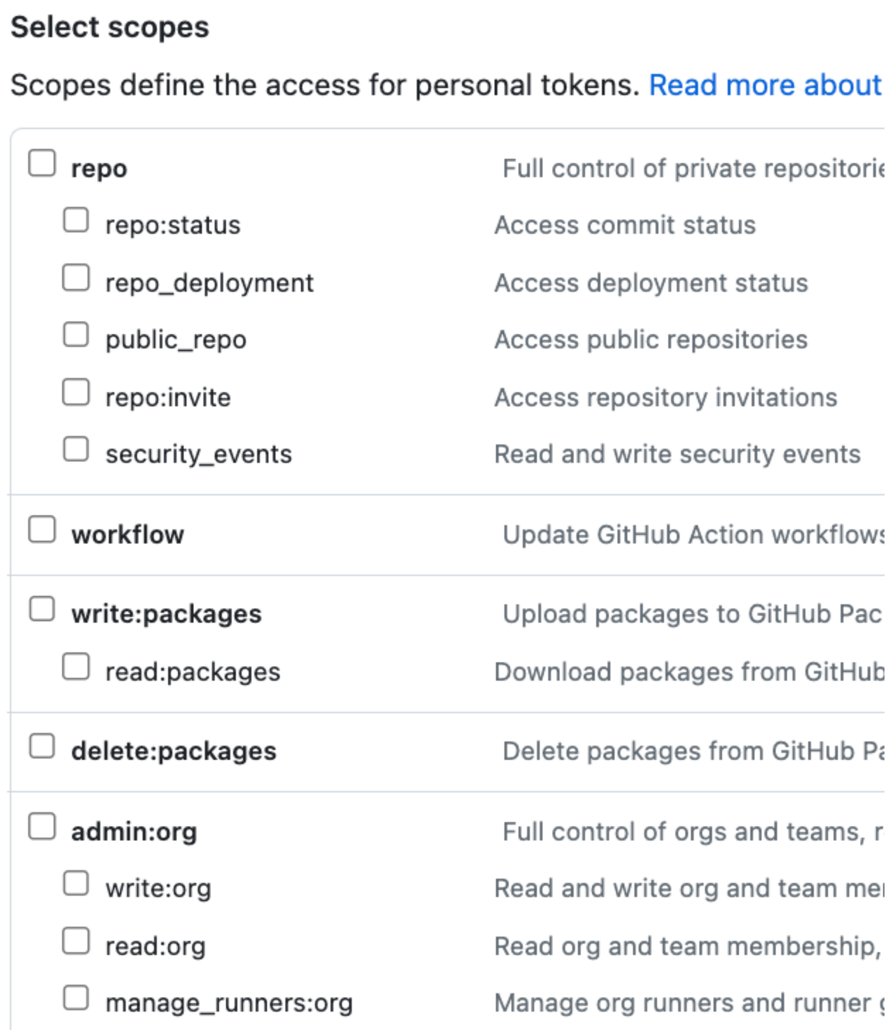
Test Last used within the last week

Token2 — *repo, workflow*

Expires *on Tue, May 9 2023*.

Token Scopes

- *"Scopes let you specify exactly what type of access you need. Scopes limit access for OAuth tokens. They* ***do not grant any additional permission beyond that which the user already has.****"*

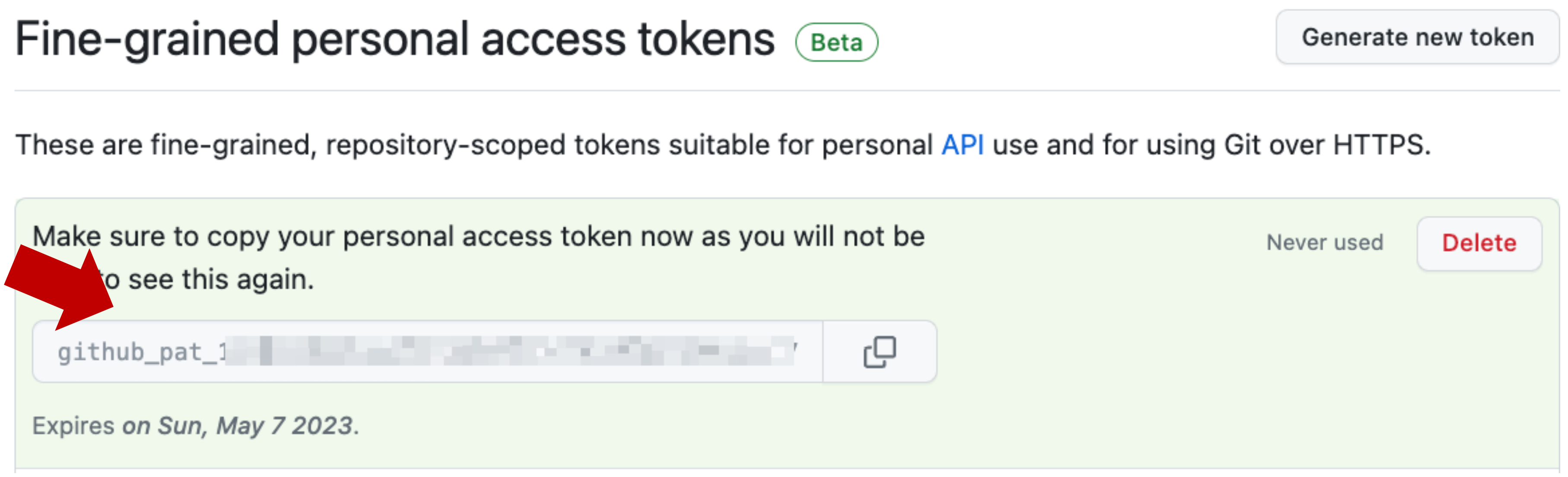


Select scopes

Scopes define the access for personal tokens. Read more about

☐ **repo**	Full control of private repositorie
☐ repo:status	Access commit status
☐ repo_deployment	Access deployment status
☐ public_repo	Access public repositories
☐ repo:invite	Access repository invitations
☐ security_events	Read and write security events
☐ **workflow**	Update GitHub Action workflows
☐ **write:packages**	Upload packages to GitHub Pac
☐ read:packages	Download packages from GitHub
☐ **delete:packages**	Delete packages from GitHub Pa
☐ **admin:org**	Full control of orgs and teams, r
☐ write:org	Read and write org and team me
☐ read:org	Read org and team membership,
☐ manage_runners:org	Manage org runners and runner

Token Types



Fine-grained personal access tokens

Beta

Generate new token

These are fine-grained, repository-scoped tokens suitable for personal API use and for using Git over HTTPS.

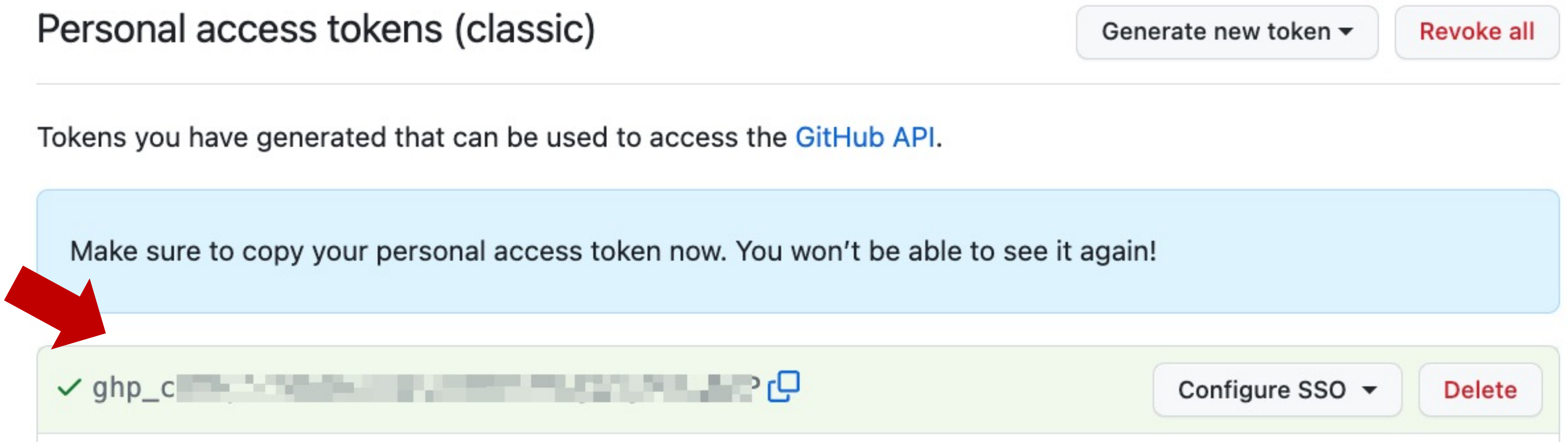
Make sure to copy your personal access token now as you will not be o see this again.

Never used

Delete

`github_pat_1`[redacted]

Expires *on Sun, May 7 2023*.



Personal access tokens (classic)

Generate new token ▾

Revoke all

Tokens you have generated that can be used to access the GitHub API.

Make sure to copy your personal access token now. You won't be able to see it again!

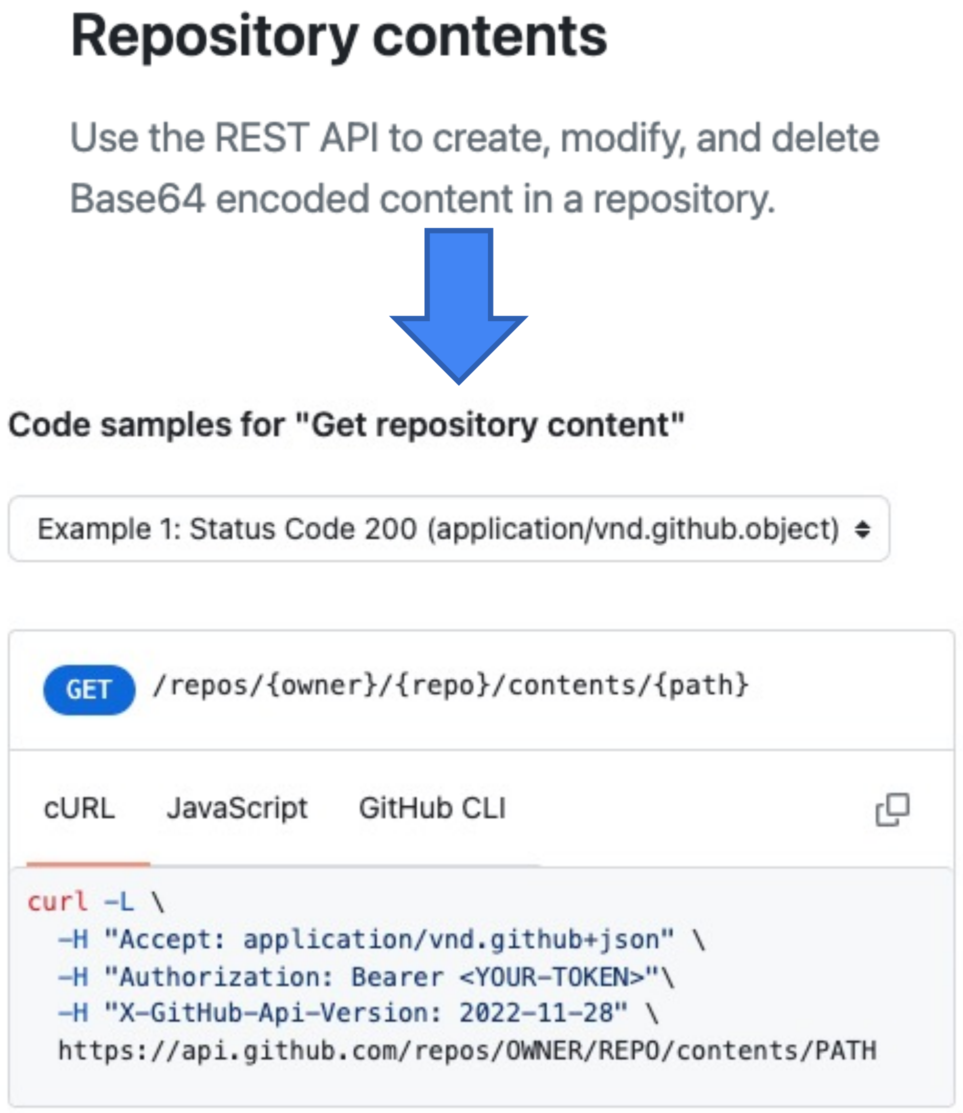
✓ ghp_c[redacted]

Configure SSO ▾

Delete

GitHub API Primer

- **Extensive** API
- Allows performing *nearly* any action
  - Low-level operations (read and commit single files using API)
- Can use API with PAT to bypass SSH CA restrictions on git operations



Repository contents

Use the REST API to create, modify, and delete Base64 encoded content in a repository.

Code samples for "Get repository content"

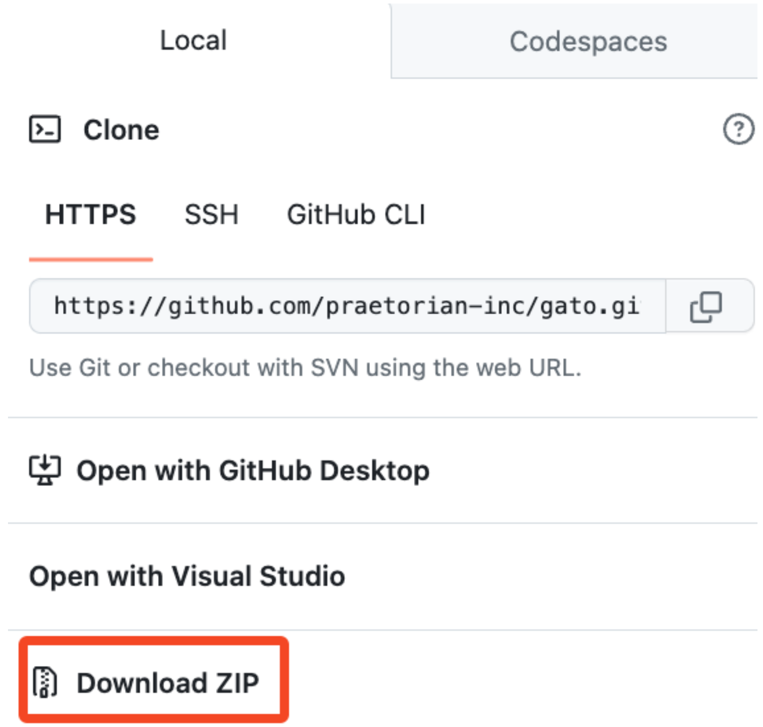
Example 1: Status Code 200 (application/vnd.github.object)

GET /repos/{owner}/{repo}/contents/{path}

```
curl -L \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/repos/OWNER/REPO/contents/PATH
```

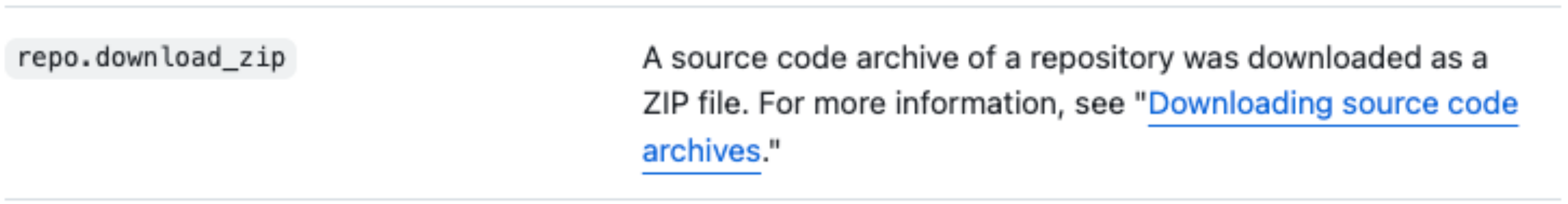
API Logging Gaps

- ***Vast majority*** of GET requests are **not** logged
    - Most notable exception is the `Repo download` event
    - Enterprise REST event logging is in **private beta** as of Feb 2023!



```
https://github.com/praetorian-inc/gato.gi
```

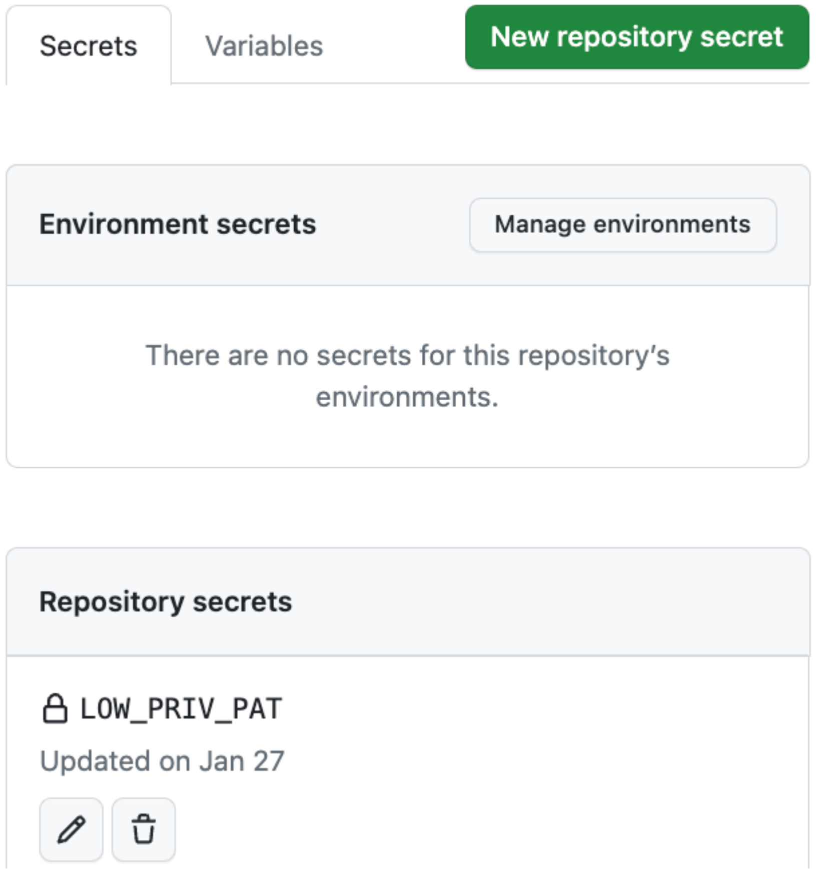
Use Git or checkout with SVN using the web URL.



`repo.download_zip`	A source code archive of a repository was downloaded as a ZIP file. For more information, see "Downloading source code archives."

Secrets

"Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork."



Environment secrets

There are no secrets for this repository's environments.

Repository secrets

LOW_PRIV_PAT

Updated on Jan 27

Why target Self-Hosted runners?

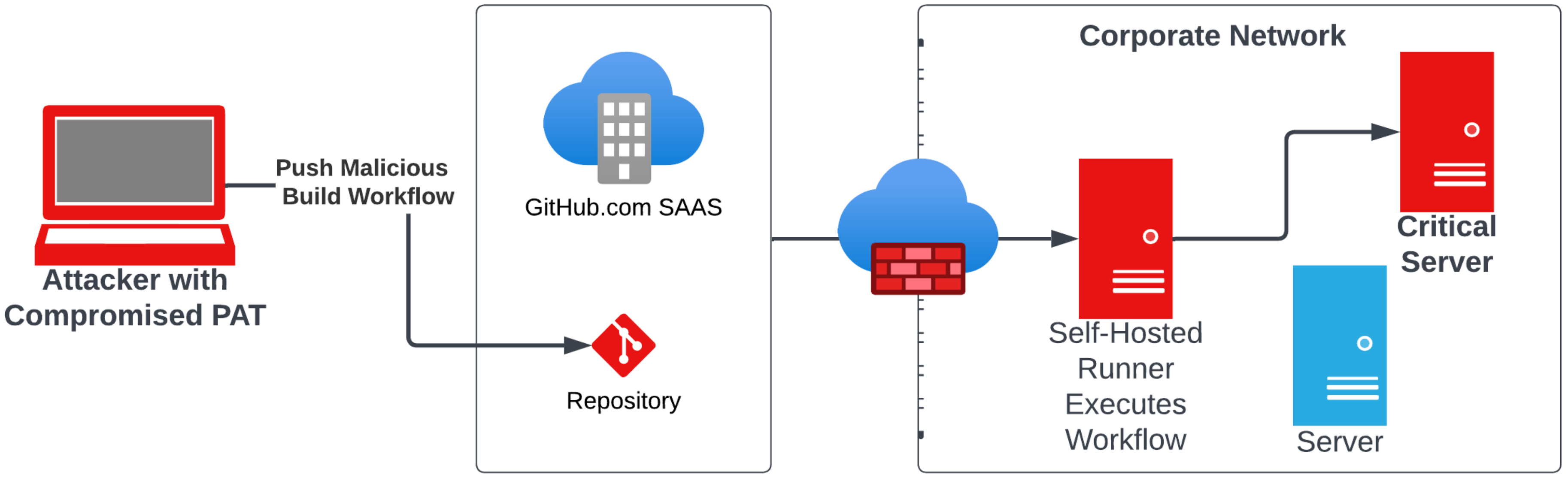
Implications of Compromise

- Source Code Theft
- Stolen secrets
  - Cloud Credentials → aws, Azure
  - Artifact repository tokens → Amazon ECR, docker hub, npm
- Supply chain attacks
- Broader network intrusion and lateral movement

Self-Hosted Runner Attacks

- Secrets extraction from ***other*** **builds** if runners are non-ephemeral
- Supply chain attacks
- Internal network access *from the Internet*

External -> Internal Attack


Attacker with Compromised PAT
Push Malicious Build Workflow
GitHub.com SAAS
Repository
Corporate Network
Self-Hosted Runner Executes Workflow
Critical Server
Server

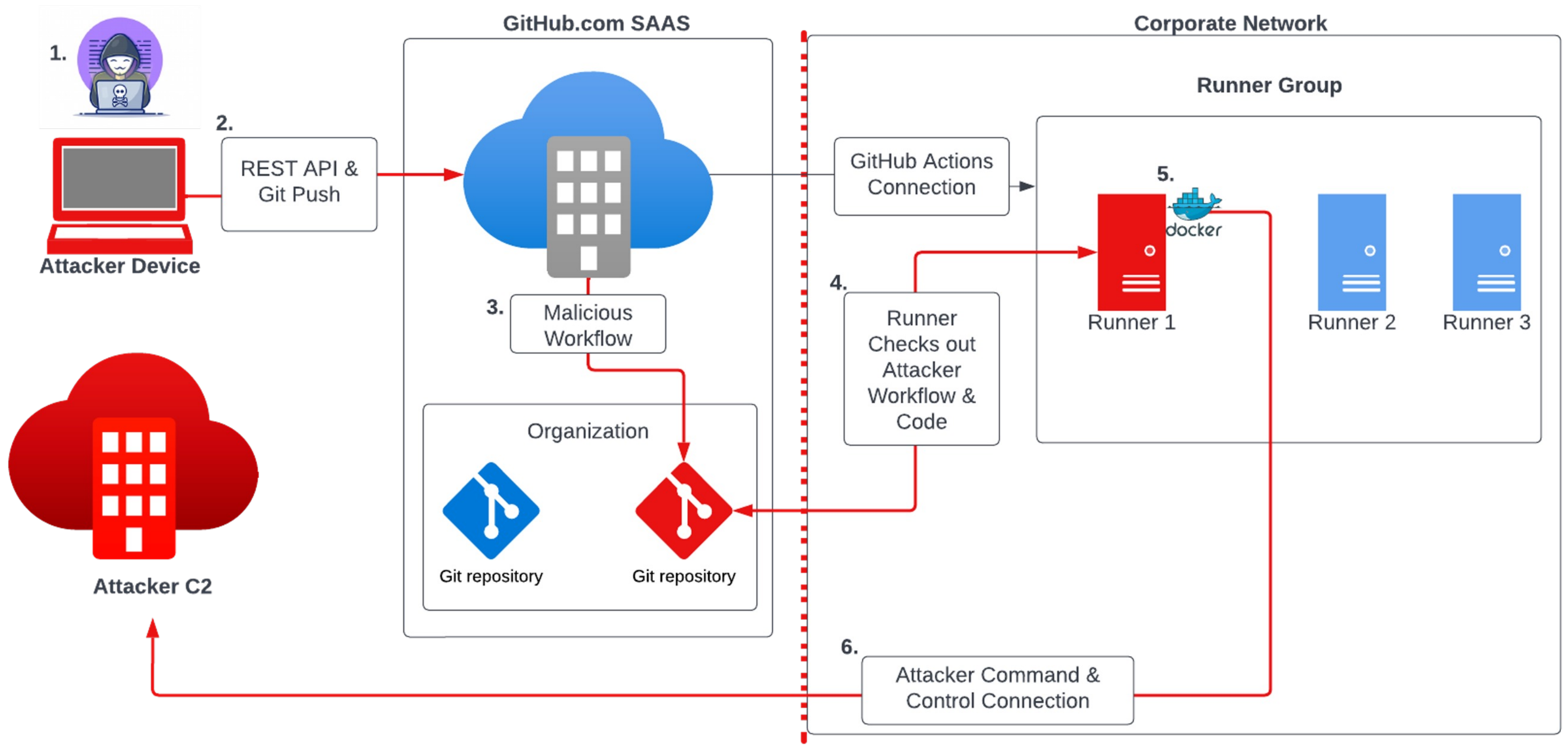
Deployment of EDR and AV is rare on self-hosted runners - unobfuscated malware will often go undetected

Is this happening?

- GitHub Self-Hosted Runners have yet to be publicly implicated in a major cyber-attack. However, this is likely due to the following:
  - (Until recently) Lack of open-source tooling to attack Self-Hosted runners
  - Lack of published offensive tradecraft until late 2022
- **We have yet to see** a client with a secure self-hosted runner implementation
- For most environments:

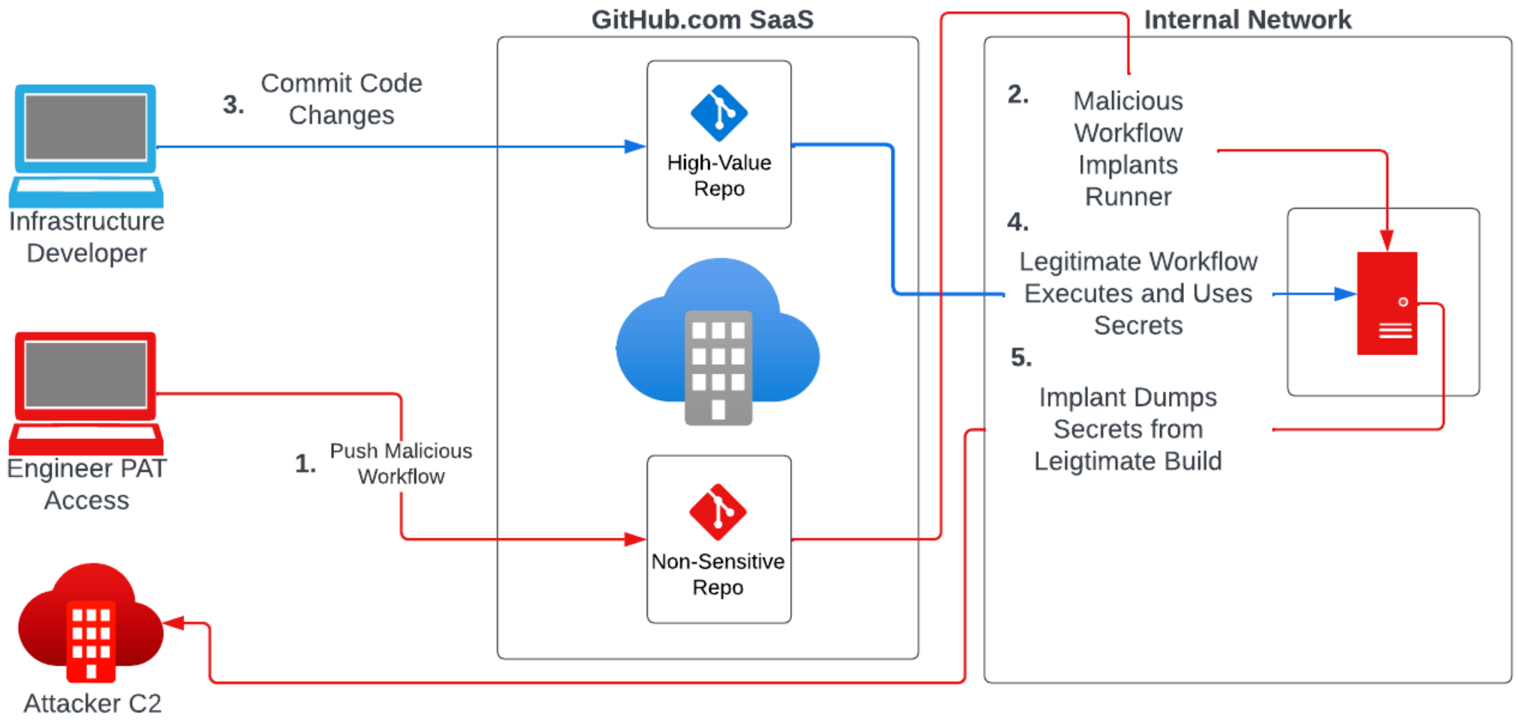
Compromised Dev = Game Over.

Example: Abuse of Compromised PAT



Real-world attack conducted on a Red Team

Example: Secret Compromise from Engineer Foothold


GitHub.com SaaS
Internal Network
3. Commit Code Changes
High-Value Repo
Infrastructure Developer
2. Malicious Workflow Implants Runner
4. Legitimate Workflow Executes and Uses Secrets
5. Implant Dumps Secrets from Leigtimate Build
Engineer PAT Access
1. Push Malicious Workflow
Non-Sensitive Repo
Attacker C2

Conducted by Praetorian during a Penetration Test

Risks and Mitigations

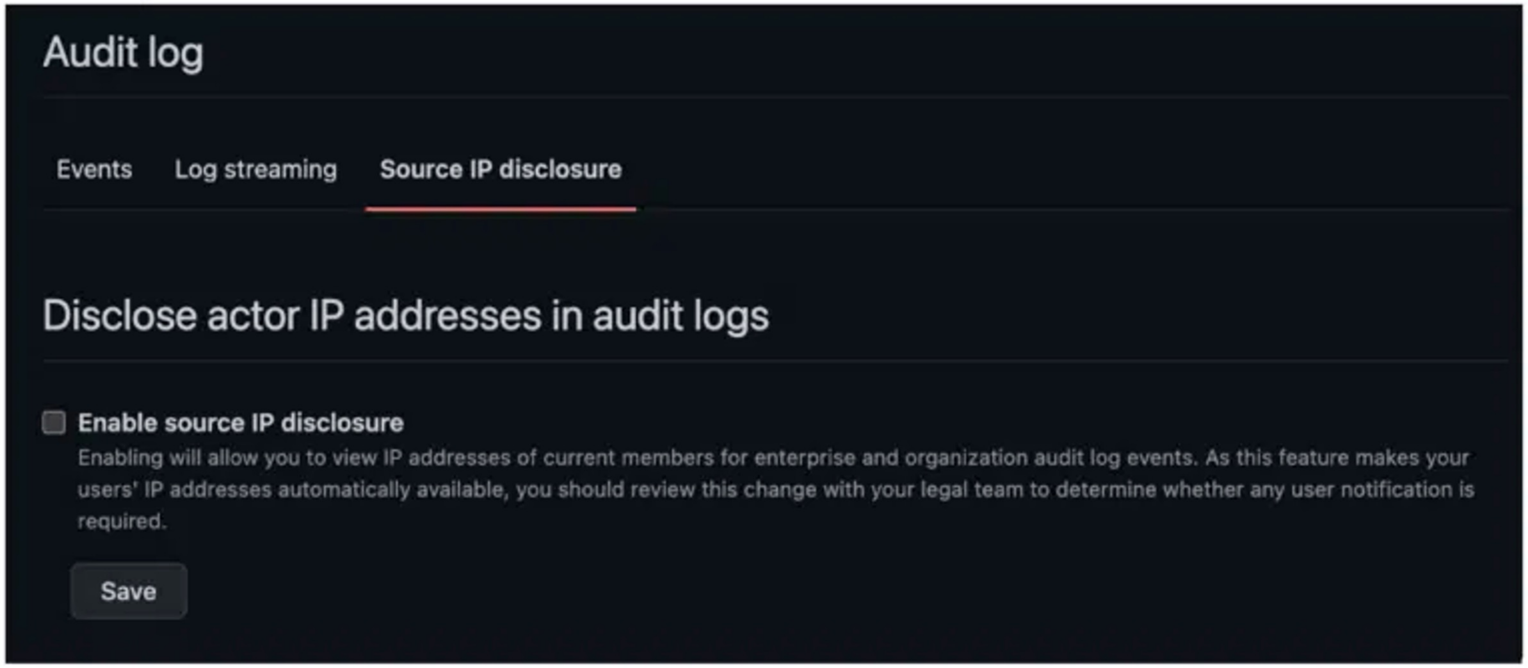
Network Foothold	Network segmentation, build monitoring
Secrets Stealing	Repo ACLs, secrets auditing/inventory
Compromised PAT	Fine-grained tokens, developer security awareness, detection engineering
Subsequent Build Modification	*Per-job* ephemeral runners
Repository Code Modification	*Per-job* ephemeral runners
Malware Deployment	Network segmentation, EDR

Properly securing a GitHub organization against sophisticated attackers requires a layered approach!

Shields Up: Perimeter Security

- Require SAML SSO for Org access!
- Require SSH Authentication for Git operations
  - Prevents `git clone` of repositories using a PAT
- Enable IP Logging
- ***Remember:*** Repo write = Repo secret access
  - If a repository has secrets, only those with a need-to-know should have write access
  - If a user has write access, they can **push a workflow that utilizes and then exfiltrates all secrets**.

Enable IP Logging


Audit log

Events | Log streaming | Source IP disclosure

Disclose actor IP addresses in audit logs

☐ **Enable source IP disclosure**
Enabling will allow you to view IP addresses of current members for enterprise and organization audit log events. As this feature makes your users' IP addresses automatically available, you should review this change with your legal team to determine whether any user notification is required.

Save

IP addresses are *not* logged by default, this should be enabled!

Risk: Classic Tokens

- Tied to an **individual user**, and granted OAuth scopes by that user at creation time.

- A single PAT, ***with no accompanying knowledge*** can be used to learn about and query everything that it has access to.

Recommendations:

- Organizations can restrict classic tokens, but this is all or nothing. If Classic PATs are not a part of your CI/CD workflow, consider restricting them *before* tooling and processes become dependent on them

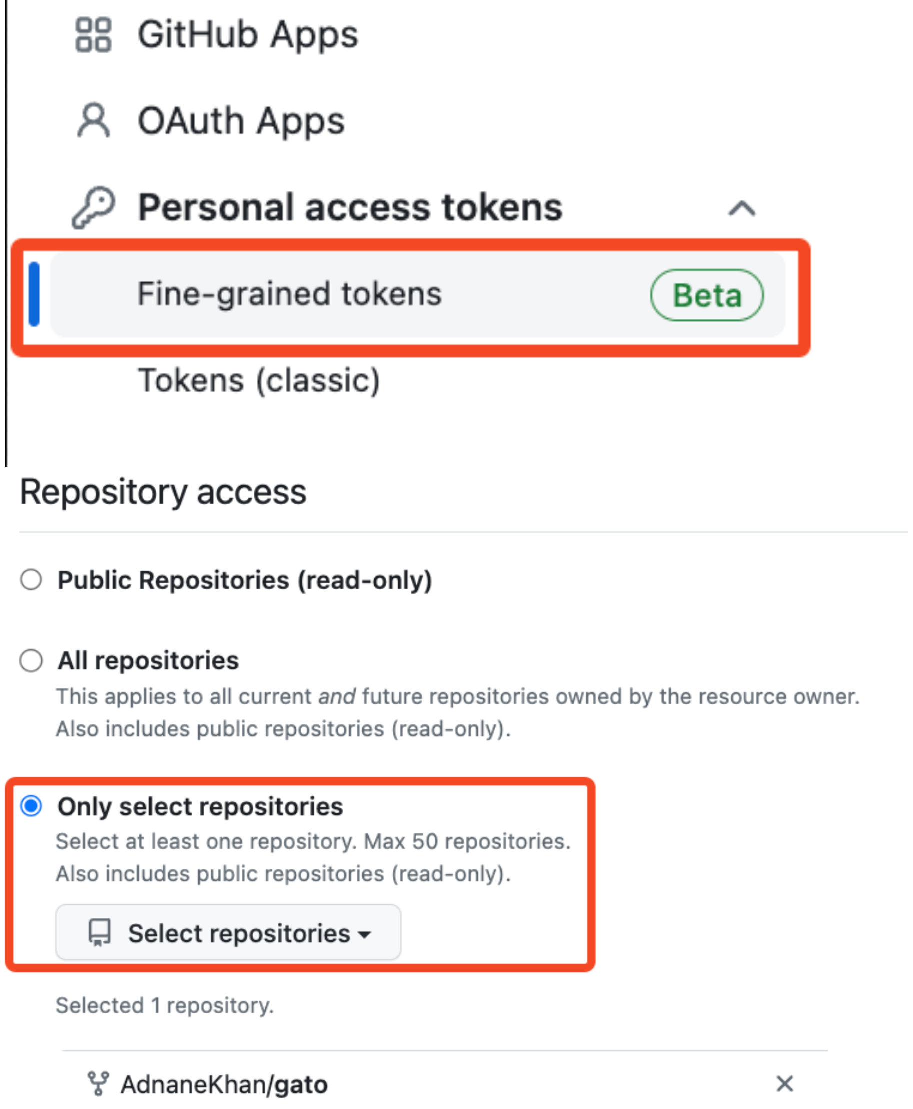
- Organizations can require that tokens be authorized to the organization with SSO. **All organizations should require this!**

Fine Grained PAT

Most important changes for organization security:

- **Can be authorized to specific repos**, and the permissions can be reduced to follow the principle of least privilege.
- Organizations can **approve & revoke**
- ***Much*** harder for an attacker to use for enumeration

Organizations should encourage developers to use fine-grained tokens over classic if they need to access company resources!



Repository access

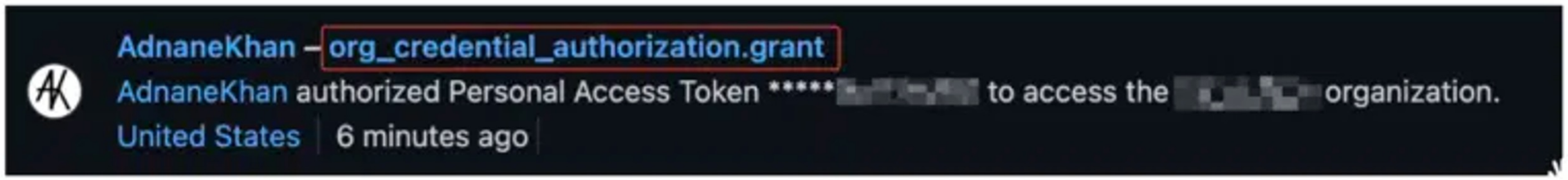
- ○ **Public Repositories (read-only)**
- ○ **All repositories**
  This applies to all current *and* future repositories owned by the resource owner. Also includes public repositories (read-only).
- ● **Only select repositories**
  Select at least one repository. Max 50 repositories. Also includes public repositories (read-only).

Selected 1 repository.

AdnaneKhan/**gato**

PAT SSO Alerting

Fine-grained personal access tokens are currently in Beta,
If your organization cannot fully disable Classic PATs, **monitoring PAT SSO authorization helps provide visibility**.



AdnaneKhan – org_credential_authorization.grant
AdnaneKhan authorized Personal Access Token *****[redacted] to access the [redacted] organization.
United States | 6 minutes ago

SSO grants can be **listed** & **revoked** via API
(https://api.github.com/orgs/ORG/credential-authorizations)

- **API response displays scopes**
- **Can detect over-provisioned token**

If there is suspicious activity associated with a user, then SSO grants can be revoked!

Lower the Blast Doors: Secondary Defenses

What can an organization do to protect themselves if an attacker *does* control a developer token or account?

Risk: Workflow Secrets Stealing

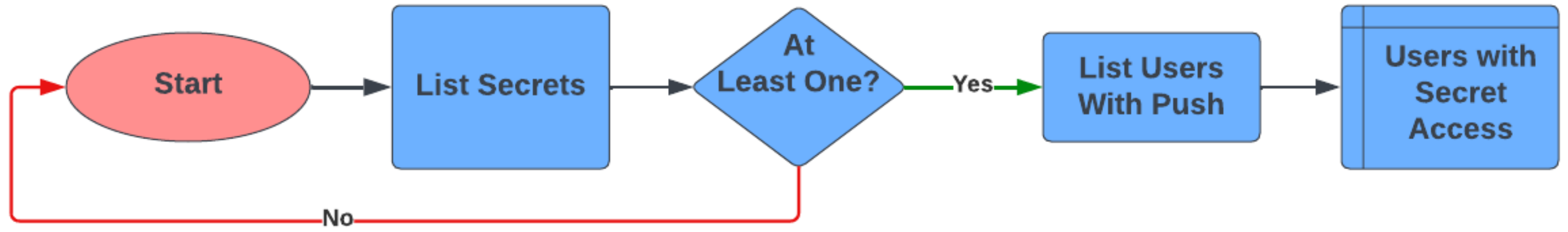
- **Any** user with write access to a repo can obtain secrets via GH Actions
  - List secrets (GitHub API)
  - Create malicious workflow that uses each secret
  - Run and extract secrets from workflow log
- Restricting branch access is not enough, **secrets can be read from any branch.**

Recommendation:

Inventory secrets associated with each repository & list of users with write permission. Ensure that you are comfortable with the resulting access mapping.

Inventory Secrets & Access

For each Repository:



Every user returned from the second call can obtain **all** plaintext secrets by using them in a workflow!

List Secrets API Call:
https://api.github.com/repos/OWNER/REPO/actions/secrets

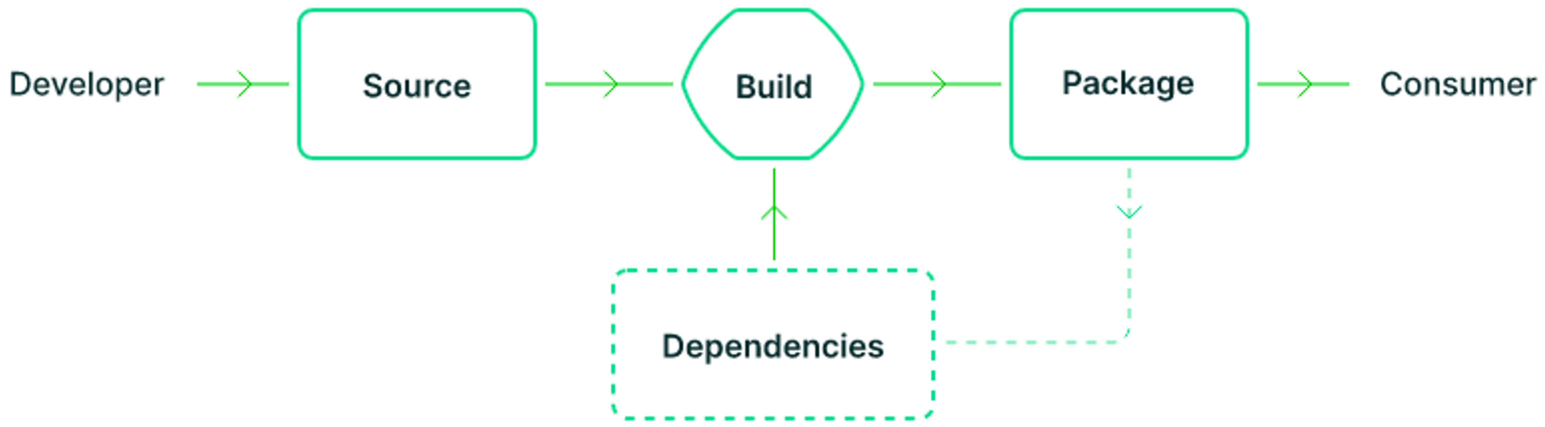
List Users with Push Access to a Repo:
https://api.github.com/repos/OWNER/REPO/collaborators?permission=push

Does something formalize all of this?

SLSA

Version 1.0 Released April 19th

SLSA Framework



The SLSA Framework is very new, however, it serves to formalize a set of standards and controls that can be used to qualify the maturity of an organization’s software supply chain security posture.

The SLSA framework offers **4 levels**, each with a different set of requirements that apply to the stages of the software supply chain lifecycle.

SLSA Levels and Tracks

Build L0	None
Build L1	Automated build process that generates provenance
Build L2	Signed provenance, hosted build service
Build L3	Build service hardening

SLSA Build L3 – Hardened Builds

The steps needed to secure Self-Hosted Runners fall within the SLSA level 3 requirements. Organizations that utilize Self-Hosted GitHub runners should aim for SLSA Build L3 compliance.

SLSA Level 3 requirements for the build step:

Isolated Builds

Protected signing secrets (user defined build should not be able to access signing keys)

This is on top of Level 1 and 2 requirements, which are scripted-build and utilization of a build service. Using GitHub actions meets these.

Risk: Malware Deployment

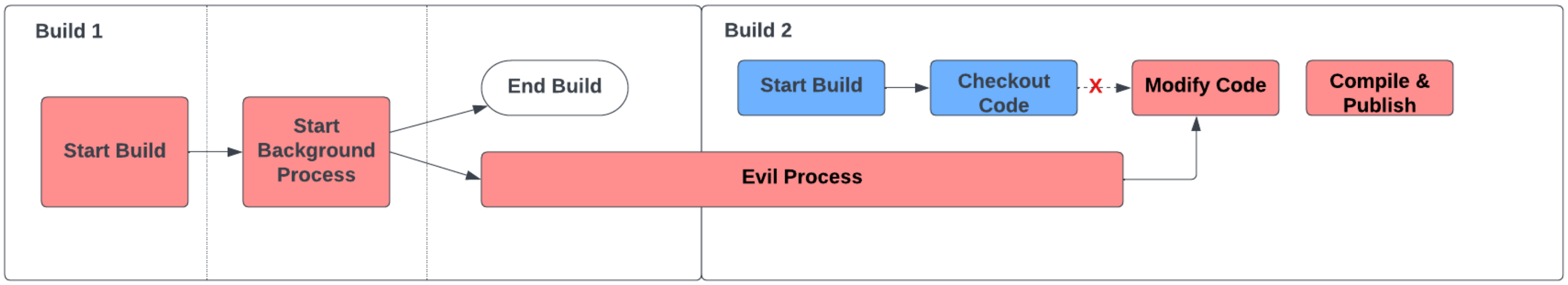
Anti-virus or EDR should be present on **all** Self-Hosted runners.

Attackers should **NOT** simply be able to run:

```
  export RUNNER_TRACKING_ID=0
nohup ./malware_elf &
```

Risk: Code Modification

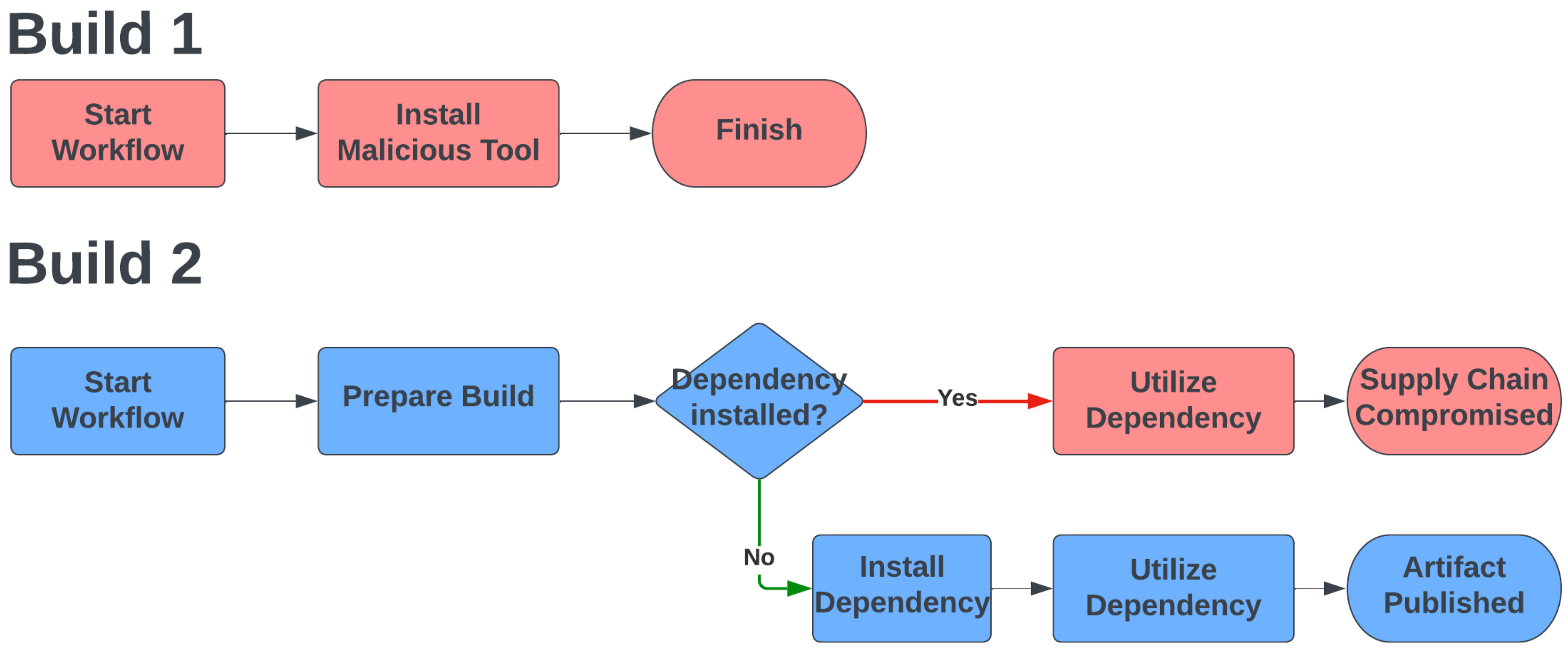
- Attackers can modify subsequent builds
- Malware can watch the filesystem for checkout, and then modify a specific file
- ***Very*** hard to detect



Risk: Tool Poisoning

On non-ephemeral runners, the aftereffects of one build should not impact how the next build functions.

If they do, then an attacker could utilize this to perform malicious behavior!



Risk: Subsequent Build Secrets Stealing

- If runners are containers that are isolated from the network, **but are not ephemeral**, then an attacker can steal secrets from subsequent builds
  - Dump memory from runner agent (if root)
  - View secrets used in shell scripts

Ephemeral is Best

1 Job - 1 Clean runner

SLSA L3 - Isolated & Ephemeral environment

Completely eliminates the following risks:

- ❎ Build time code modification of subsequent build
- ❎ Secret stealing via monitoring subsequent build
- ❎ Secret stealing via memory dumping of subsequent build

Ephemeral runners may not be realistic in all cases, so other mitigations and detective controls will be critical.

Setting Up Ephemeral Runners

GitHub supports ephemeral Self-Hosted Runners; **this only means that the agent de-registers itself** after completing the job.

End user is responsible for setting up autoscaling infrastructure to provision and de-provision runners in response to issued jobs.

Open-Source Frameworks:

https://github.com/actions/actions-runner-controller - Kubernetes Controller for GH Actions Self-Hosted Runners

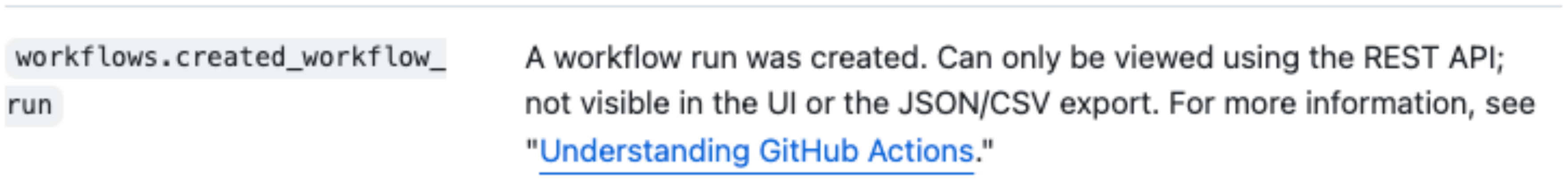
https://github.com/philips-labs/terraform-aws-github-runner - Terraform AWS Runner Controller

Internal Network Isolation

Even if an organization's Self-Hosted runners are ephemeral, internal access from the runner can be a risk!

Workflow runs can last up to **35 days** -> You **do not** want an attacker living in your network for 35 days!

- Ensure that self-hosted runners within the default group have minimal internal network access
- For runners that need more extensive access, limit repos that can use them and audit workflow executions via the **created_workflow_run** log event.

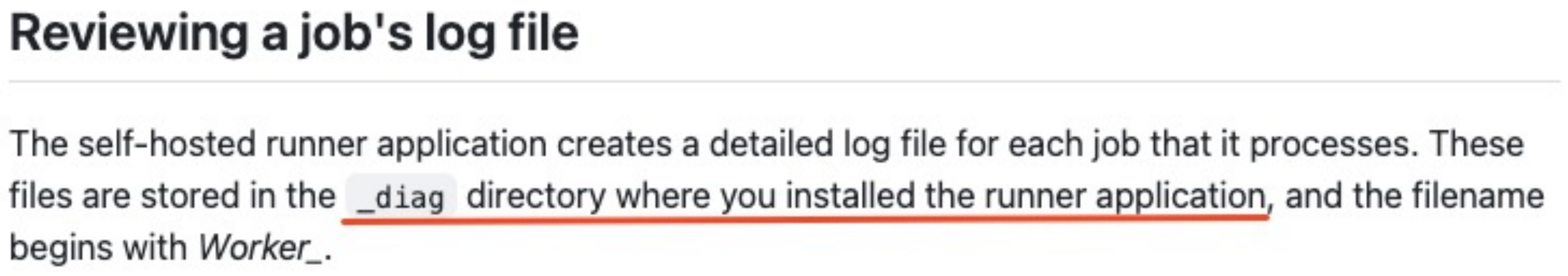


`workflows.created_workflow_run`	A workflow run was created. Can only be viewed using the REST API; not visible in the UI or the JSON/CSV export. For more information, see "Understanding GitHub Actions."

Catch Long-Running Builds

In most environments, a build lasting for days at minimum is indicative of a failure case.

- GitHub’s API does not offer an easy way to check builds on a single self-hosted runner.
- Self-Hosted runners themselves maintain log files on the filesystem, these can be ingested to track the workflow status on Self-Hosted runners.



Reviewing a job's log file

The self-hosted runner application creates a detailed log file for each job that it processes. These files are stored in the `_diag` directory where you installed the runner application, and the filename begins with *Worker_*.

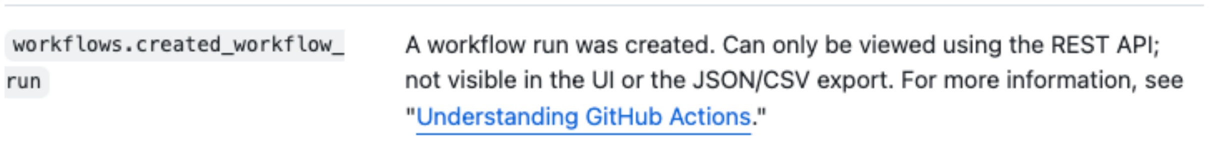
Detect Orphan Process

- On Linux/OS X runners, if a process is running with the **RUNNER_TRACKING_ID** environment variable set to 0, then it can persist beyond the job
  - Check all processes started by the GitHub actions user
  - Check /proc/<pid>/environ
  - If **RUNNER_TRACKING_ID** is set to empty or 0, raise an alert

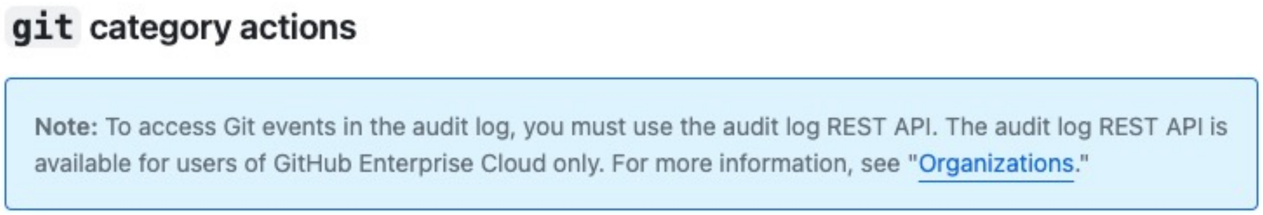
Audit Log Ingestion

Certain GitHub logs can only be ingested via the REST API

- Workflow runs
- git actions
  (push/clone/fetch)



`workflows.created_workflow_run`	A workflow run was created. Can only be viewed using the REST API; not visible in the UI or the JSON/CSV export. For more information, see "Understanding GitHub Actions."



`git` category actions

Note: To access Git events in the audit log, you must use the audit log REST API. The audit log REST API is available for users of GitHub Enterprise Cloud only. For more information, see "Organizations."

GitHub Enterprise can be configured to stream audit logs directly to a SIEM

This can be very noisy, but checking the source IP associated with these events can detect a potential compromise.

This is especially important for critical repositories within an organization.

Deploy a Honeypot Repository

- Create a repository accessible to **all** users in the GitHub Org
- Grant write access to **all** users
- Name it 'infrastructure-deployment-manager' (be creative!)
- Add fake secrets with names such as:
  - **AWS_ADMIN_ID**
  - **AWS_ADMIN_SECRET_KEY**
  - **GCP_PROD_INFRA_SA**
- Add a **repo-level** Self-Hosted runner (**run it as a container within an isolated cloud environment**)
- Query the GitHub Audit API every ~minute for log events related to that repo
  - https://docs.github.com/en/organizations/keeping-your-organization-secure/managing-security-settings-for-your-organization/reviewing-the-audit-log-for-your-organization#search-based-on-repository
- **Alert on and investigate any events!**

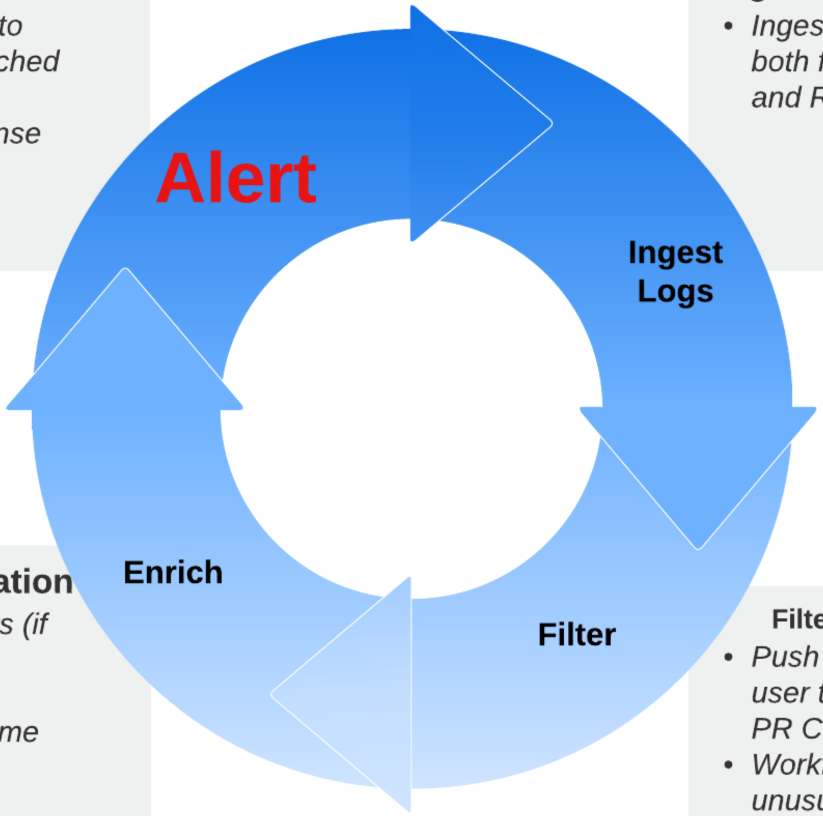
Detection Engineering

Alert

- *Notify Security team to manually review enriched results*
- *Begin incident response process if necessary*

Ingest Logs

- *Ingest GitHub audit logs both from log streaming and REST API*



Enrich with Information

- *Workflow file contents (if workflow updated)*
- *Event source IP*
- *Recent events for same actor (GitHub user)*

Filter Suspicious Events

- *Push to repository by bot user that typically updates PR Comments*
- *Workflow update from unusual IP*
- *Clone of sensitive repo by non-user account*

Detection Engineering (cont.)

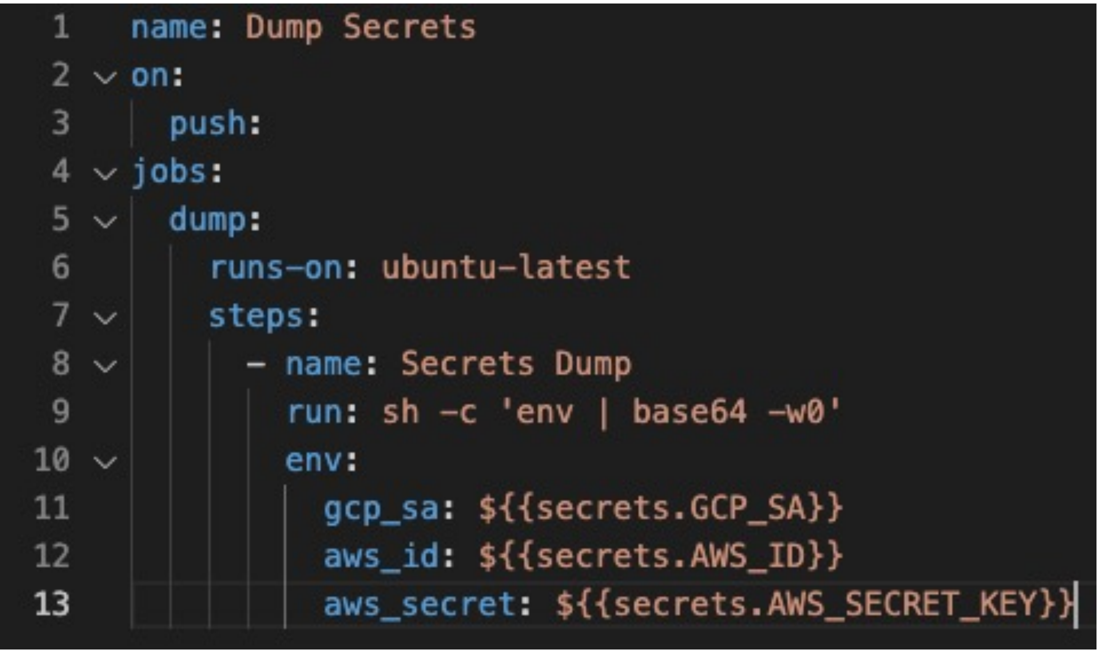
- Use GitHub API to enrich event with workflow contents:

  1. Use https://docs.github.com/en/rest/actions/workflow-runs?apiVersion=2022-11-28#get-a-workflow-run to get the workflow associated with the suspicious workflow created_workflow_run log event

     Capture the **head_sha** and **path** fields from the response

  2. Use https://docs.github.com/en/rest/repos/contents?apiVersion=2022-11-28#get-repository-content along with the sha and path to get the workflow file from the **.github/workflows/** directory within the repository.

This would be bad...



```yaml
name: Dump Secrets
on:
  push:
jobs:
  dump:
    runs-on: ubuntu-latest
    steps:
      - name: Secrets Dump
        run: sh -c 'env | base64 -w0'
        env:
          gcp_sa: ${{secrets.GCP_SA}}
          aws_id: ${{secrets.AWS_ID}}
          aws_secret: ${{secrets.AWS_SECRET_KEY}}
```

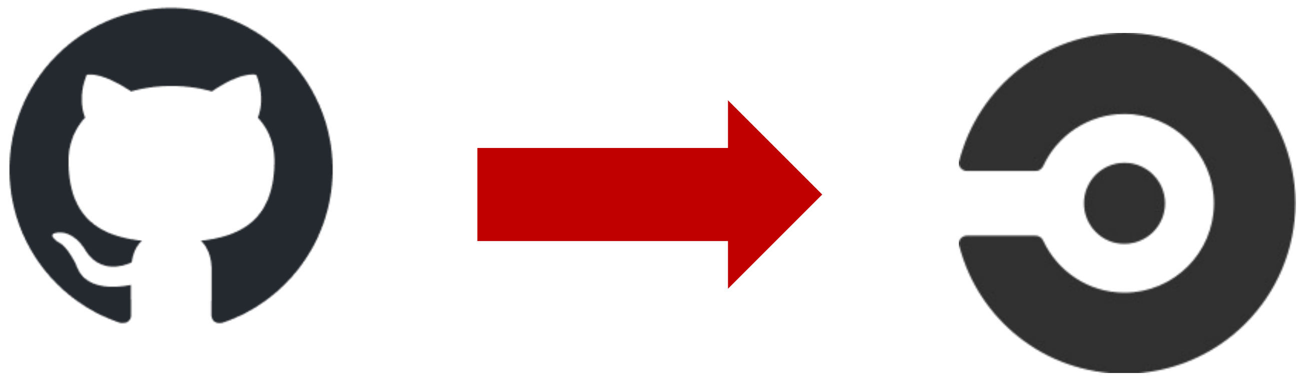
If an organization can detect malicious workflows amongst CI/CD noise, then they are more secure than most!

To Conclude...

- Developer compromises are increasingly becoming the source of impactful breaches
  - **CircleCI** – Dec 2022
  - **LastPass** – Aug 2022
- Protecting your organization's GitHub CI/CD implementation requires a layered approach
  - Least Privilege Principle
  - Secure Runner Implementation
  - Detection Engineering

Future Research

- GitHub to CircleCI Self-Hosted Runner Pivot
  - Scan for **.circleci/config.yaml files** to find evidence of self-hosted CircleCI runners
  - Push malicious workflow to GitHub, which a Self-Hosted CircleCI runner would then pick up.



Resources and References

GitHub API Documentation –

https://docs.github.com/en/rest?apiVersion=2022-11-28

Praetorian Self-Hosted Runners Blog Post -

https://www.praetorian.com/blog/self-hosted-github-runners-are-backdoors/

ShmooCon 2023 Talk –

https://www.youtube.com/watch?v=-JX7aC0AXEk&t=14222s

GitHub Actions Secrets RE – https://karimrahal.com/2023/01/05/github-actions-leaking-secrets/

Acknowledgements

- Shout out Mason Davis and Matt Jackoski, who were my co-presenters at ShmooCon and co-developers of the Gato tool.
- Also would like to thank John Stawinski and Jimmy Chang for utilizing Gato during an assessment and demonstrating a very cool attack path.

Thank you!!

Questions?