Detecting CI/CD Supply Chain Attacks with Canary Credentials | Tracebit

Archived: 2026-04-10 02:01:33 UTC

In recent weeks, a single threat actor - TeamPCP - compromised a chain of widely-used open source tools: Trivy, KICS, LiteLLM, and Telnyx. Each compromise fueled the next, with stolen credentials from one victim unlocking access to the next target. We've just shipped a GitHub Actions integration in our free Community Edition in response to this incident.

This post looks at the campaign and explores the question: once you've pinned your actions and hardened your runners, what actually detects credential exfiltration from a compromised CI/CD pipeline?

TeamPCP campaign timeline

Compromising a GitHub Action or PyPI package, even for just a few hours, can result in a large number of repositories executing malicious code, as demonstrated by this quantitative analysis of the TeamPCP incidents. Also Rami from Wiz has published a nice technical deep dive with further details about the campaign, including the number of downloads of the malicious PyPI packages.

- **March 19:** The threat actor compromised multiple GitHub actions of Aqua Security's Trivy scanner.
- **March 22:** TeamPCP used stolen credentials to compromise dozens of npm packages using a self-propagating worm, dubbed CanisterWorm. They also published malicious versions of Trivy on Docker Hub.
- **March 23:** GitHub actions and OpenVSX extensions of Checkmarx's KICS scanner were compromised with a similar credential-stealing malware used in Trivy's intrusion.
- **March 24:** LiteLLM PyPI package was compromised, resulting in two malicious versions published by the threat actor.
- **March 27:** Telnyx PyPI package was compromised, again with two malicious versions published by the threat actor.

What does the malicious code look for?

Secrets, of course.

All these attacks deploy malware designed for mass credential harvesting. Anything discovered in the environment - SSH keys, cloud credentials, git credentials, package manager tokens etc - are located and sent to the threat actor. Each incident deployed essentially the same self-described "TeamPCP Cloud Stealer", but with progressive refinements.

The compromised GitHub actions of Trivy and KICS (here's an example of the malicious commit affecting trivy-action) perform two types of credential harvesting based on the environment running the code:

- In GitHub-hosted Linux runners, the action runs the code contained in this memdump.py script, which dumps Runner.Worker process memory to steal GitHub secrets.
- In all other environments, it scrapes the filesystem searching for cloud credentials and other secrets across many sensitive file paths.

With the LiteLLM compromise, TeamPCP introduced a new lateral movement stage: after harvesting credentials, if a Kubernetes service account token is present, the malware reads all cluster secrets across all namespaces and attempts to create a privileged “alpine:latest” pod on every node in kube-system, mounting the host filesystem.

According to vxunderground, the LiteLLM attack alone resulted in approximately 300GB of data exfiltrated from 500,000 infected machines.

Deceptive detections with CI/CD canaries

Most guidance after these incidents focuses on preventive controls - pinning actions to specific commit SHAs, reviewing dependencies, hardening runner configurations. GitHub Actions themselves offer some quite comprehensive guidance.

Preventive controls are worth implementing in pipelines, though they can involve more effort than they first appear. To give one obvious example: a pinned action or dependency does not necessarily make for an idempotent one if it fetches anything dynamically at runtime.

What about detective controls? Runtime monitoring of CI/CD environments is desirable, but there’s a lot to consider. In both Trivy and KICS attacks, the malicious code starts before the legitimate scanner services, and eventually the workflows appear to complete normally. The LiteLLM case is more pernicious: the malware is implanted by installing a .pth file that executes on every Python process startup across the entire environment - not just when LiteLLM is explicitly used. Aside from the details of payload injection: the Trivy attack demonstrated fallback exfiltration to trusted hosts (by creating repos using the GitHub API) - so network monitoring alone is insufficient. Other major supply chain attacks using GitHub as exfiltration path include tj-actions, s1ngularity, Shai-Hulud.

Canary credentials take a different approach: fake credentials injected into your build environment. If a compromised action or package exfiltrates credentials from your runner, canaries give you a clear signal - either as an initial detection, or as additional context when you discover you’ve been affected by a wider supply chain incident. This isn't just a theoretical idea: Grafana wrote last year about their successes using this technique.

When we saw the TeamPCP campaign unfold, we built a GitHub Action for Tracebit’s free Community Edition that deploys canary AWS credentials into CI/CD workflows.

How it works

The idea is simple:

1. At the start of your workflow, the action calls the Tracebit API to receive a unique, short-lived set of canary AWS credentials

2. The credentials are written to `~/.aws/credentials`, exported as environment variables, and held in the runner process's memory - covering every common exfiltration surface: credential files, environment variable dumps, and process memory scraping.
3. Tracebit monitors for any use of those credentials. If they are used, you get an alert with full context.

Because canary credentials sit alongside real credentials in the runner's memory, environment variables, and credential files, they would be exfiltrated by the same code that steals legitimate secrets. In the TeamPCP example, this means any workflow using a compromised action or malicious package could trigger an alert with full context about the affected repository, workflow, and run.

Detection coverage against TeamPCP techniques

In cases where the threat actor compromised a GitHub action, the canaries cover both exfiltration surfaces:

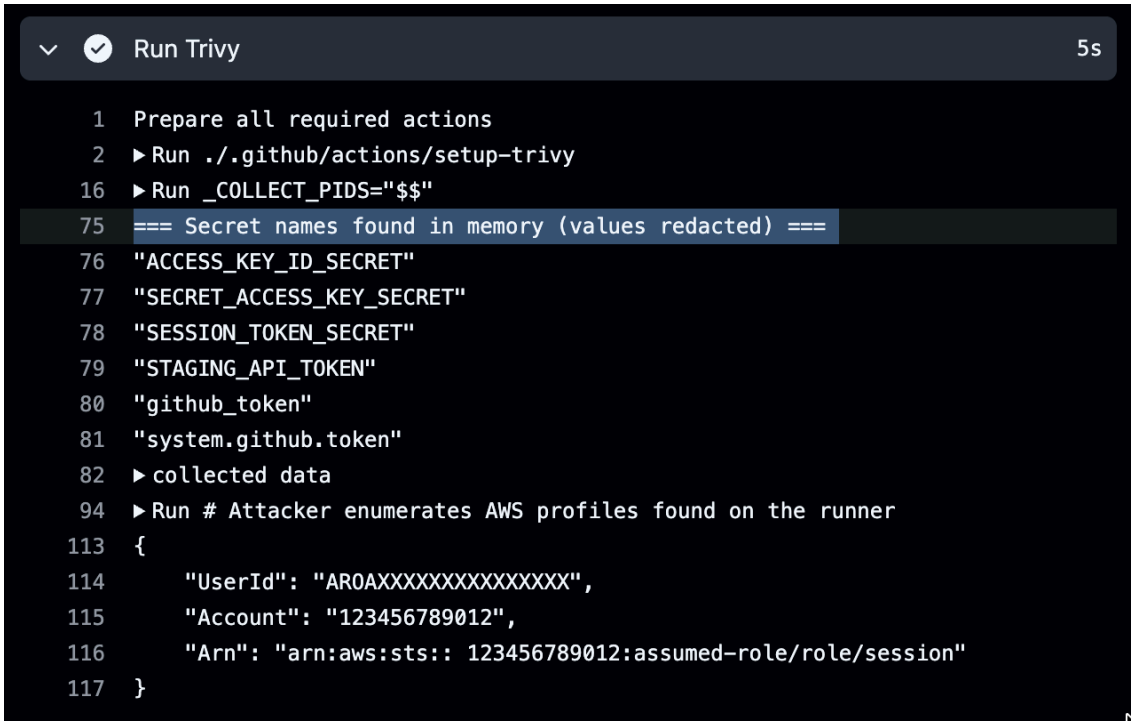
- On GitHub-hosted runners, the malicious code would have exfiltrated the canary AWS credentials from the Runner.Worker process memory.
- In other runner environments, the canaries would instead be exfiltrated from `~/.aws/credentials`.

We reproduced the Trivy attack by creating a workflow that includes the Tracebit Action and uses the malicious setup-trivy action. Instead of encrypting and exfiltrating stolen credentials, we modified it to print them in the action output and call STS:GetCallerIdentity. The workflow runs in a GitHub-hosted runner, like this:

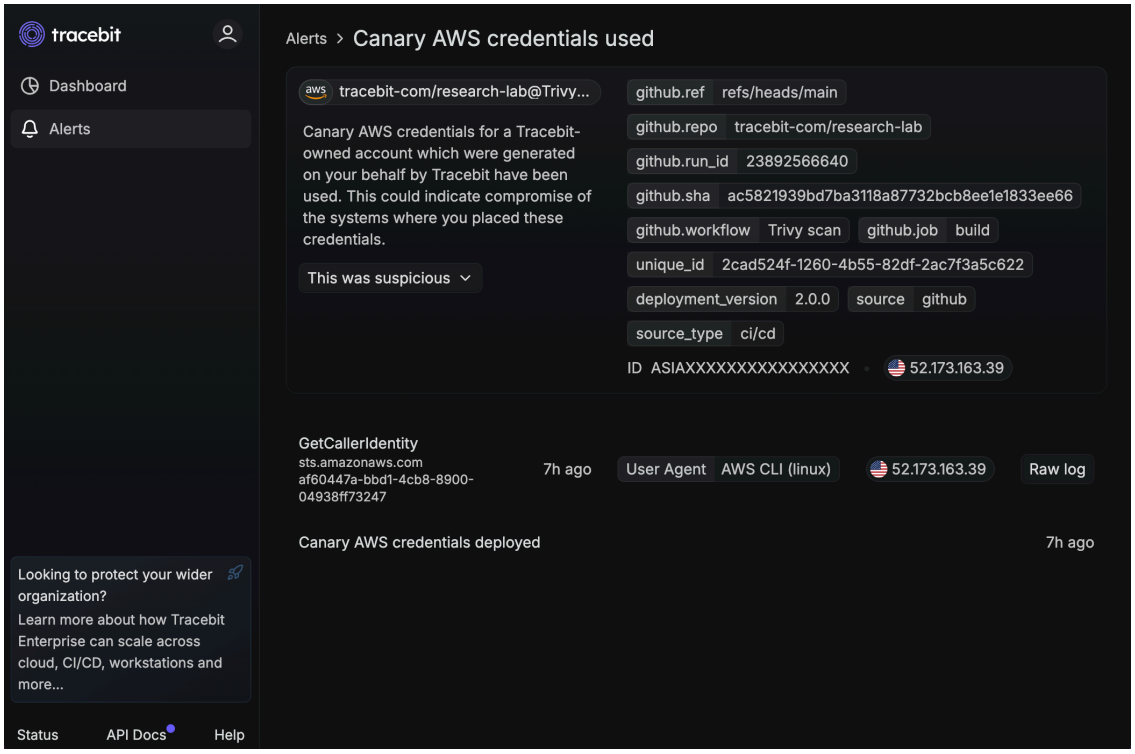
```
 name: "Trivy scan"

on:
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Configure credentials
        uses: tracebit-com/tracebit-community-action@517c410eae144100a1995cd720094c010995994d
        with:
          api-token: ${{ secrets.SECURITY_API_TOKEN }}
          profile: administrator
          profile-region: us-east-1
          async: true

      - name: Run Trivy
        uses: ./.github/actions/setup-trivy
```

After running the workflow, the malicious code inside the compromised setup-trivy action retrieved the GitHub

secrets from the memory of the Runner.Worker process, including the AWS canary credentials.



Using the exfiltrated credentials triggers an alert on the Community Edition dashboard, which can lead to further investigation.



The Tracebit alert has full context about the affected workflow: repository, workflow, job, commit SHA, and run ID. It also captures the attacker's IP address and user-agent, along with the CloudTrail logs generated by the API calls made with those credentials. Canary credentials have no permissions, so any API call except GetCallerIdentity fails - but the attempt itself is the signal.

Grabbing secrets from runner memory is a known technique, used in other impactful supply chain attacks, such as the incidents that affected tj-actions and reviewdog last year.

If the workflow instead uses a malicious version of a compromised package, such as LiteLLM, canary AWS credentials stored in `~/.aws/credentials` would have been exfiltrated. The LiteLLM malware not only harvested credentials, but actively called AWS APIs (ListSecrets, GetSecretValue, DescribeParameters), as documented in this article by StepSecurity. Tracebit canaries would have been used in these API calls, resulting in immediate detection of the attack.

Conclusion

Supply chain attacks can be self-sustaining. Trivy credentials unlocked CanisterWorm, which spread to 66+ packages. LiteLLM used Trivy in their CI/CD security scan workflow, handing over PyPI publishing tokens. Telnyx tokens were likely harvested from LiteLLM-compromised environments. Each victim's valid credentials become the attack surface for the next target.

Pinning and hardening reduce the attack surface, but what made TeamPCP's campaign so effective was the propagation - stolen credentials from one victim unlocking access to the next. Canary credentials can't prevent that chain from starting, but they can tell you that your credentials have left the building. Teams should treat their build infrastructure with the same detection rigor as production.

The Tracebit Community Action is free and takes a few minutes to add to a workflow. We hope it's useful - and we'd welcome feedback on where to take it next.
