Attacking and Defending GitHub Actions

Is it safe now?

By Simon Gerst

02.07.2026

GitHub Actions

What?

- CI/CD
- Automate workflows
- Run on GitHub servers
- Free for public repos

GitHub Actions

What?

- CI/CD
- Automate workflows
- Run on GitHub servers
- Free for public repos

Why?

- Easy to use
- Easy to integrate
- Easy to extend
- Free for public repos

GitHub Actions

What?

- CI/CD
- Automate workflows
- Run on GitHub servers
- Free for public repos

Why?

- Easy to use
- Easy to integrate
- Easy to extend
- Free for public repos

Why attack?

- Can modify source code: supply chain attacks
- Give access to secrets, such as API keys: escalate privileges

Workflow files

- YAML files [a]
- Must be stored in `.github/workflows`
- Can be triggered by events
- Can be triggered manually
- Can be triggered by other workflows

[a] ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell

Example workflow

```yaml
name: hello-world
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    steps:
      - name: Hello World
        run: echo "Hello World ${{ github.actor }}"
```

Workflow files

- YAML files [a]
- Must be stored in `.github/workflows`
- Can be triggered by events
- Can be triggered manually
- Can be triggered by other workflows

[a] ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell

Example workflow

```yaml
name: hello-world
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    steps:
      - name: Hello World
        run: echo "Hello World ${{ github.actor }}"
```

> `name`: name of the workflow

Workflow files

- YAML files [a]
- Must be stored in `.github/workflows`
- Can be triggered by events
- Can be triggered manually
- Can be triggered by other workflows

[a] ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell

Example workflow

```yaml
name: hello-world
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    steps:
      - name: Hello World
        run: echo "Hello World ${{ github.actor }}"
```

> `on`: trigger for the workflow

Workflow files

- YAML files [a]
- Must be stored in `.github/workflows`
- Can be triggered by events
- Can be triggered manually
- Can be triggered by other workflows

[a] ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell

Example workflow

```yaml
name: hello-world
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    steps:
      - name: Hello World
        run: echo "Hello World ${{ github.actor }}"
```

> `jobs`: jobs to run

Workflow files

- YAML files [a]
- Must be stored in `.github/workflows`
- Can be triggered by events
- Can be triggered manually
- Can be triggered by other workflows

[a] ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell

Example workflow

```yaml
name: hello-world
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    steps:
      - name: Hello World
        run: echo "Hello World ${{ github.actor }}"
```

`hello_world_job`: name of the job

Workflow files

- YAML files [a]
- Must be stored in `.github/workflows`
- Can be triggered by events
- Can be triggered manually
- Can be triggered by other workflows

[a] ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell

Example workflow

```yaml
name: hello-world
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    steps:
      - name: Hello World
        run: echo "Hello World ${{ github.actor }}"
```

> `runs-on`: OS to run the job on

Workflow files

- YAML files [a]
- Must be stored in `.github/workflows`
- Can be triggered by events
- Can be triggered manually
- Can be triggered by other workflows

[a] ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell

Example workflow

```yaml
name: hello-world
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    steps:
      - name: Hello World
        run: echo "Hello World ${{ github.actor }}"
```

> `steps`: steps to run

Workflow files

- YAML files [a]
- Must be stored in `.github/workflows`
- Can be triggered by events
- Can be triggered manually
- Can be triggered by other workflows

[a] ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell

Example workflow

```yaml
name: hello-world
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    steps:
      - name: Hello World
        run: echo "Hello World ${{ github.actor }}"
```

`name: Hello World`: name of the step

Workflow files

- YAML files [a]
- Must be stored in `.github/workflows`
- Can be triggered by events
- Can be triggered manually
- Can be triggered by other workflows

[a] ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell

Example workflow

```yaml
name: hello-world
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    steps:
      - name: Hello World
        run: echo "Hello World ${{ github.actor }}"
```

> `run`: a shell command step

Example triggers

Events

- Push to a branch: `on: push`
- Pull request: `on: pull_request`

```yaml
name: Build and Test
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Install dependencies
        run: npm install
      - name: Build project
        run: npm run build
      - name: Run tests
        run: npm test
```

Example triggers

Events

- Push to a branch: `on: push`
- Pull request: `on: pull_request`
- Issue comment: `on: issue_comment`

```yaml
name: Comment Notification
on:
  issue_comment:
    types: [created]
jobs:
  notify:
    runs-on: ubuntu-latest
    steps:
      - name: Send notification
        uses: peter-evans/slack-action@v3
        with:
          slack_secret: ${{ secrets.SLACK_WEBHOOK_URL }}
          text: "New comment by ${{
github.event.comment.user.login }}"
```

Example triggers

Events

- Push to a branch: `on: push`
- Pull request: `on: pull_request`
- Issue comment: `on: issue_comment`
- Cron: `on: schedule`

```yaml
name: Stale Issues
on:
  schedule:
    - cron: '20 16 * * *' # every day at 16:20 UTC
jobs:
  stale:
    runs-on: ubuntu-latest
    steps:
      - name: Close Stale Issues
        uses: actions/stale@v3.0.5
        with:
          repo-token: ${{ secrets.GITHUB_TOKEN }}
          days-before-stale: 30
```

Example triggers

Events

- Push to a branch: `on: push`
- Pull request: `on: pull_request`
- Issue comment: `on: issue_comment`
- Cron: `on: schedule`
- Manual (workflow dispatch): `on: workflow_dispatch`

```yaml
on:
  workflow_dispatch:
    inputs:
      V8_VERSION:
        description: "V8_VERSION"
        required: false
        type: string
        default: ""
      V8_BUILD_MODE:
        default: "release-fuzz"
        required: true
        type: choice
        options:
          - release-fuzz
          - debug-no-fuzz
```

Permission model for workflows

- Every workflow gets an API token: GITHUB_TOKEN

Permission model for workflows

- Every workflow gets an API token: GITHUB_TOKEN
- Base permissions can be set to permissive or restricted

Permission model for workflows

- Every workflow gets an API token: GITHUB_TOKEN
- Base permissions can be set to permissive or restricted
- Permissive: practically read/write access to everything

Permission model for workflows

- Every workflow gets an API token: GITHUB_TOKEN
- Base permissions can be set to permissive or restricted
- Permissive: practically read/write access to everything
- Restricted: only read access, or no access at all

Permission model for workflows

- Every workflow gets an API token: GITHUB_TOKEN
- Base permissions can be set to permissive or restricted
- Permissive: practically read/write access to everything
- Restricted: only read access, or no access at all
- For pull requests from forks: reduce all permissions to a maximum of read access, and no secrets are available

Permission model for workflows

- Every workflow gets an API token: GITHUB_TOKEN
- Base permissions can be set to permissive or restricted
- Permissive: practically read/write access to everything
- Restricted: only read access, or no access at all
- For pull requests from forks: reduce all permissions to a maximum of read access, and no secrets are available
- `issue_comment`, `issues`, `push`, and `workflow_run` give base permissions and secret access

Permission model for workflows

- Every workflow gets an API token: GITHUB_TOKEN
- Base permissions can be set to permissive or restricted
- Permissive: practically read/write access to everything
- Restricted: only read access, or no access at all
- For pull requests from forks: reduce all permissions to a maximum of read access, and no secrets are available
- `issue_comment`, `issues`, `push`, and `workflow_run` give base permissions and secret access
- `pull_request_target` gives base permissions and secret access even for forks

Permission model for workflows

- Every workflow gets an API token: GITHUB_TOKEN
- Base permissions can be set to permissive or restricted
- Permissive: practically read/write access to everything
- Restricted: only read access, or no access at all
- For pull requests from forks: reduce all permissions to a maximum of read access, and no secrets are available
- `issue_comment`, `issues`, `push`, and `workflow_run` give base permissions and secret access
- `pull_request_target` gives base permissions and secret access even for forks
- Only `pull_request` is restricted for forks

User-controlled input

User-controlled input

the root of all evil

Templating in workflows

```
${{ ... }}
```

- Inserted as-is into the workflow file before execution
- Predefined variables: `github`, `env`, `secrets`, ...
- Predefined functions: `fromJson`, `startsWith`, `endsWith`, `contains`, ...

Example workflow: unsafe templating

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is ${{ github.event.comment.body }}"
```

Example workflow: unsafe templating

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is ${{ github.event.comment.body }}"
```

- Comment body is `This is a test`

Example workflow: unsafe templating

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is This is a test"
```

- Comment body is `This is a test`

Example workflow: unsafe templating

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is This is a test"
```

- Comment body is `This is a test`
- This is similar to XSS or SQL injection

Example workflow: unsafe templating

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is This is a test"
```

- Comment body is `This is a test`
- This is similar to XSS or SQL injection
- We are mixing *code* and *data*

Example workflow: unsafe templating

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is This is a test"
```

- Comment body is `This is a test`
- This is similar to XSS or SQL injection
- We are mixing *code* and *data*
- Comment body is `$(touch /tmp/pwned)`

Example workflow: unsafe templating

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is $(touch /tmp/pwned)"
```

- Comment body is `This is a test`
- This is similar to XSS or SQL injection
- We are mixing *code* and *data*
- Comment body is `$(touch /tmp/pwned)`

Example workflow: unsafe templating

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is $(touch /tmp/pwned)"
```

- Comment body is `This is a test`
- This is similar to XSS or SQL injection
- We are mixing *code* and *data*
- Comment body is `$(touch /tmp/pwned)`
- Oops, we just let someone execute arbitrary code

How to fix this?

- Don't mix code and data
- Instead, store data in environment variables

How to fix this?

- Don't mix code and data
- Instead, store data in environment variables

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is $COMMENT"
        env:
          COMMENT: ${{ github.event.comment.body }}
```

How to fix this?

- Don't mix code and data
- Instead, store data in environment variables
- The shell will not interpret the environment variable as code

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is $COMMENT"
        env:
          COMMENT: ${{ github.event.comment.body }}
```

How to fix this?

- Don't mix code and data
- Instead, store data in environment variables
- The shell will not interpret the environment variable as code
- No code injection

```yaml
name: Example Workflow
on:
  issue_comment:
    types: [created]
jobs:
  example_job:
    runs-on: ubuntu-latest
    steps:
      - name: Print comment
        run: echo "The comment is $COMMENT"
        env:
          COMMENT: ${{ github.event.comment.body }}
```

Cloning and executing untrusted code

- `on: pull_request_target` and some other triggers give access to secrets and a token with write access

Cloning and executing untrusted code

- `on: pull_request_target` and some other triggers give access to secrets and a token with write access
- Attacker opens PR against repository

Cloning and executing untrusted code

- `on: pull_request_target` and some other triggers give access to secrets and a token with write access
- Attacker opens PR against repository
- Code is cloned and executed

Cloning and executing untrusted code

- `on: pull_request_target` and some other triggers give access to secrets and a token with write access
- Attacker opens PR against repository
- Code is cloned and executed
- Oops, we just let someone execute arbitrary code

Cloning and executing untrusted code

- `on: pull_request_target` and some other triggers give access to secrets and a token with write access
- Attacker opens PR against repository
- Code is cloned and executed
- Oops, we just let someone execute arbitrary code
- `on: pull_request` is safe

Cloning and executing untrusted code

- `on: pull_request_target` and some other triggers give access to secrets and a token with write access
- Attacker opens PR against repository
- Code is cloned and executed
- Oops, we just let someone execute arbitrary code
- `on: pull_request` is safe
- Doesn't give access to secrets and write access

Example workflow: unsafe cloning

```yaml
on:
  pull_request_target
jobs:
  build:
    name: Build and test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
        with:
          ref: ${{ github.event.pull_request.head.sha }}
      - uses: actions/setup-node@v1
      - run: |
          npm install
          npm build
      - run: gh pr comment ${{ github.event.pull_request.number }} -b "Build successful"
```

- `${{ github.event.pull_request.head.sha }}` is user-controlled and cloned
- `npm install` and `npm build` can execute user-controlled code

How to fix this?

- Don't clone and execute untrusted code

How to fix this?

- Don't clone and execute untrusted code
- Just as with binaries: code should be executable XOR writable
- Clone and execute untrusted code in an unprivileged environment XOR clone (don't execute) in a privileged environment

How to fix this?

- Don't clone and execute untrusted code
- Just as with binaries: code should be executable XOR writable
- Clone and execute untrusted code in an unprivileged environment XOR clone (don't execute) in a privileged environment
- Sometimes this is not possible, e.g. when tests need to run and post results as a comment

How to fix this?

- Don't clone and execute untrusted code
- Just as with binaries: code should be executable XOR writable
- Clone and execute untrusted code in an unprivileged environment XOR clone (don't execute) in a privileged environment
- Sometimes this is not possible, e.g. when tests need to run and post results as a comment

Split the workflow

- Unprivileged workflow: clone and execute untrusted code
- Save results as artifacts

How to fix this?

- Don't clone and execute untrusted code
- Just as with binaries: code should be executable XOR writable
- Clone and execute untrusted code in an unprivileged environment XOR clone (don't execute) in a privileged environment
- Sometimes this is not possible, e.g. when tests need to run and post results as a comment

Split the workflow

- Unprivileged workflow: clone and execute untrusted code
- Save results as artifacts
- Privileged workflow: fetch artifacts and work with them

How to fix this? Workflow splitting

How to fix this? Workflow splitting

`pull_request`: unprivileged, executes untrusted code

```yaml
on:
  pull_request
jobs:
  build:
    name: Build and test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
        with:
          ref: ${{ github.event.pull_request.head.sha
}}
      - uses: actions/setup-node@v1
      - run: |
          npm install
          npm build
        # upload artifact if needed
```

How to fix this? Workflow splitting

`pull_request`: unprivileged, executes untrusted code

```yaml
on:
  pull_request
jobs:
  build:
    name: Build and test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
        with:
          ref: ${{ github.event.pull_request.head.sha
}}
      - uses: actions/setup-node@v1
      - run: |
          npm install
          npm build
        # upload artifact if needed
```

`workflow_run`: privileged, doesn't execute untrusted code

```yaml
on:
  workflow_run:
    workflows:
      - name-of-previous-workflow
    types:
      - completed
    status:
      - success
jobs:
  comment:
    name: Build and test
    runs-on: ubuntu-latest
    steps:
      - run: gh pr comment ${{
github.event.workflow_run.pull_requests[0].number }} -b
"Build successful"
```

How to (mostly) fix this? Safer checkout defaults

What changed?

- `actions/checkout@v7` refuses common pwn requests by default

What is not fixed?

[a] github.blog/changelog/2026-06-18-safer-pull_request_target-defaults-for-github-actions-checkout/

How to (mostly) fix this? Safer checkout defaults

What changed?

- `actions/checkout@v7` refuses common pwn requests by default
- Fork PR code is blocked in `pull_request_target`

What is not fixed?

[a] github.blog/changelog/2026-06-18-safer-pull_request_target-defaults-for-github-actions-checkout/

How to (mostly) fix this? Safer checkout defaults

What changed?

- `actions/checkout@v7` refuses common pwn requests by default
- Fork PR code is blocked in `pull_request_target`
- Also blocked in PR-based `workflow_run`

What is not fixed?

[a] github.blog/changelog/2026-06-18-safer-pull_request_target-defaults-for-github-actions-checkout/

How to (mostly) fix this? Safer checkout defaults

What changed?

- `actions/checkout@v7` refuses common pwn requests by default
- Fork PR code is blocked in `pull_request_target`
- Also blocked in PR-based `workflow_run`
- Floating major tags get the backport on July 16, 2026 [a]

What is not fixed?

[a] github.blog/changelog/2026-06-18-safer-pull_request_target-defaults-for-github-actions-checkout/

How to (mostly) fix this? Safer checkout defaults

What changed?

- `actions/checkout@v7` refuses common pwn requests by default
- Fork PR code is blocked in `pull_request_target`
- Also blocked in PR-based `workflow_run`
- Floating major tags get the backport on July 16, 2026 [a]

What is not fixed?

- Manual `git` or `gh` checkout is not blocked

[a] github.blog/changelog/2026-06-18-safer-pull_request_target-defaults-for-github-actions-checkout/

How to (mostly) fix this? Safer checkout defaults

What changed?

- `actions/checkout@v7` refuses common pwn requests by default
- Fork PR code is blocked in `pull_request_target`
- Also blocked in PR-based `workflow_run`
- Floating major tags get the backport on July 16, 2026 [a]

What is not fixed?

- Manual `git` or `gh` checkout is not blocked
- `issue_comment` pwn requests are not blocked

[a] github.blog/changelog/2026-06-18-safer-pull_request_target-defaults-for-github-actions-checkout/

How to (mostly) fix this? Safer checkout defaults

What changed?

- `actions/checkout@v7` refuses common pwn requests by default
- Fork PR code is blocked in `pull_request_target`
- Also blocked in PR-based `workflow_run`
- Floating major tags get the backport on July 16, 2026 [a]

What is not fixed?

- Manual `git` or `gh` checkout is not blocked
- `issue_comment` pwn requests are not blocked
- Pinned SHA, minor, and patch versions must upgrade

[a] github.blog/changelog/2026-06-18-safer-pull_request_target-defaults-for-github-actions-checkout/

How to (mostly) fix this? Safer checkout defaults

What changed?

- `actions/checkout@v7` refuses common pwn requests by default
- Fork PR code is blocked in `pull_request_target`
- Also blocked in PR-based `workflow_run`
- Floating major tags get the backport on July 16, 2026 [a]

What is not fixed?

- Manual `git` or `gh` checkout is not blocked
- `issue_comment` pwn requests are not blocked
- Pinned SHA, minor, and patch versions must upgrade
- Opt-out exists: `allow-unsafe-pr-checkout`

[a] github.blog/changelog/2026-06-18-safer-pull_request_target-defaults-for-github-actions-checkout/

How to (mostly) fix this? Safer checkout defaults

What changed?

- `actions/checkout@v7` refuses common pwn requests by default
- Fork PR code is blocked in `pull_request_target`
- Also blocked in PR-based `workflow_run`
- Floating major tags get the backport on July 16, 2026 [a]

What is not fixed?

- Manual `git` or `gh` checkout is not blocked
- `issue_comment` pwn requests are not blocked
- Pinned SHA, minor, and patch versions must upgrade
- Opt-out exists: `allow-unsafe-pr-checkout`

The common checkout-based pwn request is heavily mitigated now 🎉

[a] github.blog/changelog/2026-06-18-safer-pull_request_target-defaults-for-github-actions-checkout/

Workflow splitting and new problems

```
on:
  pull_request
jobs:
  build:
    # ...
    # upload PR number as artifact
    - run: echo ${{ github.event.pull_request.number }}
> pr_number
    - uses: actions/upload-artifact@v2
      with:
        name: pr_number
        path: pr_number
```

Workflow splitting and new problems

```
on:
  pull_request
jobs:
  build:
    # ...
    # upload PR number as artifact
    - run: echo ${{ github.event.pull_request.number }}
> pr_number
    - uses: actions/upload-artifact@v2
      with:
        name: pr_number
        path: pr_number
```

```
on:
  workflow_run:
jobs:
  comment:
    # ...
    - uses: actions/download-artifact@v2
      with:
        name: pr_number
    - run: |
        echo "PR_NUMBER=$(cat pr_number)" >> $GITHUB_ENV
    - run: gh pr comment $PR_NUMBER -b "Build
successful"
```

Workflow splitting and new problems

```yaml
on:
  pull_request
jobs:
  build:
    # ...
    # upload PR number as artifact
    - run: echo ${{ github.event.pull_request.number }}
> pr_number
    - uses: actions/upload-artifact@v2
      with:
        name: pr_number
        path: pr_number
```

```yaml
on:
  workflow_run:
jobs:
  comment:
    # ...
    - uses: actions/download-artifact@v2
      with:
        name: pr_number
    - run: |
        echo "PR_NUMBER=$(cat pr_number)" >> $GITHUB_ENV
    - run: gh pr comment $PR_NUMBER -b "Build
successful"
```

- We completely control the value of `pr_number`!

Workflow splitting and new problems

```
on:
  pull_request
jobs:
  build:
    # ...
    # upload PR number as artifact
    - run: echo ${{ github.event.pull_request.number }}
> pr_number
    - uses: actions/upload-artifact@v2
      with:
        name: pr_number
        path: pr_number
```

```
on:
  workflow_run:
jobs:
  comment:
    # ...
    - uses: actions/download-artifact@v2
      with:
        name: pr_number
    - run: |
        echo "PR_NUMBER=$(cat pr_number)" >> $GITHUB_ENV
    - run: gh pr comment $PR_NUMBER -b "Build
successful"
```

- We completely control the value of `pr_number`!
- `$GITHUB_ENV` is a special file used for setting environment variables

Workflow splitting and new problems

```
on:
  pull_request
jobs:
  build:
    # ...
    # upload PR number as artifact
    - run: echo ${{ github.event.pull_request.number }}
> pr_number
    - uses: actions/upload-artifact@v2
      with:
        name: pr_number
        path: pr_number
```

```
on:
  workflow_run:
jobs:
  comment:
    # ...
    - uses: actions/download-artifact@v2
      with:
        name: pr_number
    - run: |
        echo "PR_NUMBER=$(cat pr_number)" >> $GITHUB_ENV
    - run: gh pr comment $PR_NUMBER -b "Build
successful"
```

- We completely control the value of `pr_number`!
- `$GITHUB_ENV` is a special file used for setting environment variables
- We can inject a newline in `pr_number`, letting us set arbitrary environment variables

How to fix this?

Only really one solution:

How to fix this?

Only really one solution:

Be very careful when working with untrusted data.

How to fix this?

Only really one solution:

Be very careful when working with untrusted data.

- Verify that the types are correct
- Verify that the data is sanitized
- Verify that the data is escaped, ...

Bonus problems with `pull_request_target`

- `pull_request_target` uses the workflow file of the PR target branch

Bonus problems with `pull_request_target`

- `pull_request_target` uses the workflow file of the PR target branch
- All other triggers use the workflow file of the default branch

Bonus problems with `pull_request_target`

- `pull_request_target` uses the workflow file of the PR target branch
- All other triggers use the workflow file of the default branch
- Different branches can have different workflows

Bonus problems with `pull_request_target`

- `pull_request_target` uses the workflow file of the PR target branch
- All other triggers use the workflow file of the default branch
- Different branches can have different workflows
- A bug might only be fixed on the default branch

Bonus problems with `pull_request_target`

- `pull_request_target` uses the workflow file of the PR target branch
- All other triggers use the workflow file of the default branch
- Different branches can have different workflows
- A bug might only be fixed on the default branch
- Just exploit another, still vulnerable branch

Bonus problems with `pull_request_target`

- ~~`pull_request_target` uses the workflow file of the PR target branch~~
- ~~All other triggers use the workflow file of the default branch~~
- ~~Different branches can have different workflows~~
- ~~A bug might only be fixed on the default branch~~
- ~~Just exploit another, still vulnerable branch~~
- **Workflows now always use the default branch of the repository**

[a] github.blog/changelog/2025-11-07-actions-pull_request_target-and-environment-branch-protections-changes/

My workflow has (almost) no permissions and no secrets

- Scenario: `pull_request_target` workflow, only permissions for comments

My workflow has (almost) no permissions and no secrets

- Scenario: `pull_request_target` workflow, only permissions for comments
- This is safe, right?

My workflow has (almost) no permissions and no secrets

- Scenario: `pull_request_target` workflow, only permissions for comments
- This is safe, right?
- GitHub Actions provide caches

My workflow has (almost) no permissions and no secrets

- Scenario: `pull_request_target` workflow, only permissions for comments
- This is safe, right?
- GitHub Actions provide caches
- Normally they are not shared between pull requests, but for `pull_request_target` they are!

My workflow has (almost) no permissions and no secrets

- Scenario: `pull_request_target` workflow, only permissions for comments
- This is safe, right?
- GitHub Actions provide caches
- Normally they are not shared between pull requests, but for `pull_request_target` they are!
- Poison the cache, hope that a more privileged workflow uses it

My workflow has (almost) no permissions and no secrets

- Scenario: `pull_request_target` workflow, only permissions for comments
- This is safe, right?
- GitHub Actions provide caches
- Normally they are not shared between pull requests, but for `pull_request_target` they are!
- Poison the cache, hope that a more privileged workflow uses it
- Profit

Cache poisoning: Actions cache

- Poison the cache

[a] github.com/AdnaneKhan/Cacheract

Cache poisoning: Actions cache

- Poison the cache
- Privileged workflow restores it

[a] github.com/AdnaneKhan/Cacheract

Cache poisoning: Actions cache

- Poison the cache
- Privileged workflow restores it
- `actions/cache/restore` extracts attacker-controlled files

[a] github.com/AdnaneKhan/Cacheract

Cache poisoning: Actions cache

- Poison the cache
- Privileged workflow restores it
- `actions/cache/restore` extracts attacker-controlled files
- Restore overwrites arbitrary files

[a] github.com/AdnaneKhan/Cacheract

Cache poisoning: Actions cache

- Poison the cache
- Privileged workflow restores it
- `actions/cache/restore` extracts attacker-controlled files
- Restore overwrites arbitrary files
- `setup-node` uses Actions cache internally

[a] github.com/AdnaneKhan/Cacheract

Cache poisoning: Actions cache

- Poison the cache
- Privileged workflow restores it
- `actions/cache/restore` extracts attacker-controlled files
- Restore overwrites arbitrary files
- `setup-node` uses Actions cache internally
- Need to predict the key

[a] github.com/AdnaneKhan/Cacheract

Cache poisoning: Actions cache

- Poison the cache
- Privileged workflow restores it
- `actions/cache/restore` extracts attacker-controlled files
- Restore overwrites arbitrary files
- `setup-node` uses Actions cache internally
- Need to predict the key

[a] github.com/AdnaneKhan/Cacheract

- Ready-made tool:
- Cacheract can find, predict, and poison cache entries [a]
- Self-replicating worm

Cache poisoning: `tempo/tempo-xyz`

Why?

- Tempo uses `Depot` runners for CI and releases

How?

Cache poisoning: `tempo/tempo-xyz`

Why?

- Tempo uses `Depot` runners for CI and releases
- Depot cache is not branch-isolated

How?

Cache poisoning: `tempo/tempo-xyz`

Why?

- Tempo uses `Depot` runners for CI and releases
- Depot cache is not branch-isolated
- Cache namespace is shared across unrelated jobs and branches

How?

Cache poisoning: `tempo/tempo-xyz`

Why?

- Tempo uses `Depot` runners for CI and releases
- Depot cache is not branch-isolated
- Cache namespace is shared across unrelated jobs and branches
- Normally, PRs cannot poison default-branch caches

How?

Cache poisoning: `tempo/tempo-xyz`

Why?

- Tempo uses `Depot` runners for CI and releases
- Depot cache is not branch-isolated
- Cache namespace is shared across unrelated jobs and branches
- Normally, PRs cannot poison default-branch caches

How?

- First external PR needs maintainer approval

Cache poisoning: `tempo/tempo-xyz`

Why?

- Tempo uses `Depot` runners for CI and releases
- Depot cache is not branch-isolated
- Cache namespace is shared across unrelated jobs and branches
- Normally, PRs cannot poison default-branch caches

How?

- First external PR needs maintainer approval
- After one benign merge, later PR workflows run immediately

Cache poisoning: `tempo/tempo-xyz`

Why?

- Tempo uses `Depot` runners for CI and releases
- Depot cache is not branch-isolated
- Cache namespace is shared across unrelated jobs and branches
- Normally, PRs cannot poison default-branch caches

How?

- First external PR needs maintainer approval
- After one benign merge, later PR workflows run immediately
- PR tests run on Depot runners

Cache poisoning: `tempo/tempo-xyz`

Why?

- Tempo uses `Depot` runners for CI and releases
- Depot cache is not branch-isolated
- Cache namespace is shared across unrelated jobs and branches
- Normally, PRs cannot poison default-branch caches

How?

- First external PR needs maintainer approval
- After one benign merge, later PR workflows run immediately
- PR tests run on Depot runners
- Unprivileged PR code can write cache entries

Cache poisoning: `tempo/tempo-xyz`

Why?

- Tempo uses `Depot` runners for CI and releases
- Depot cache is not branch-isolated
- Cache namespace is shared across unrelated jobs and branches
- Normally, PRs cannot poison default-branch caches

How?

- First external PR needs maintainer approval
- After one benign merge, later PR workflows run immediately
- PR tests run on Depot runners
- Unprivileged PR code can write cache entries

Those entries are visible to the release workflow.

`tempo/tempo-xyz`: predicting sccache keys

What?

- Release builds use `Mozilla-Actions/sccache-action`

How?

`tempo/tempo-xyz`: predicting sccache keys

What?

- Release builds use `Mozilla-Actions/sccache-action`
- No direct `cache-restore` step is used

How?

`tempo/tempo-xyz`: predicting sccache keys

What?

- Release builds use `Mozilla-Actions/sccache-action`
- No direct `cache-restore` step is used
- `sccache` talks directly to the cache service

How?

`tempo/tempo-xyz`: predicting sccache keys

What?

- Release builds use `Mozilla-Actions/sccache-action`
- No direct `cache-restore` step is used
- `sccache` talks directly to the cache service
- We cannot easily evict existing cache entries

How?

`tempo/tempo-xyz`: predicting sccache keys

What?

- Release builds use `Mozilla-Actions/sccache-action`
- No direct `cache-restore` step is used
- `sccache` talks directly to the cache service
- We cannot easily evict existing cache entries
- Target: a `reth-mdbx-sys` object built from `mdbx.c`

How?

`tempo/tempo-xyz`: predicting sccache keys

What?

- Release builds use `Mozilla-Actions/sccache-action`
- No direct `cache-restore` step is used
- `sccache` talks directly to the cache service
- We cannot easily evict existing cache entries
- Target: a `reth-mdbx-sys` object built from `mdbx.c`

How?

- Cache key material is deterministic

`tempo/tempo-xyz`: predicting sccache keys

What?

- Release builds use `Mozilla-Actions/sccache-action`
- No direct `cache-restore` step is used
- `sccache` talks directly to the cache service
- We cannot easily evict existing cache entries
- Target: a `reth-mdbx-sys` object built from `mdbx.c`

How?

- Cache key material is deterministic
- BUT: `mdbx.c` uses `__TIME__` and `__DATE__`

`tempo/tempo-xyz`: predicting sccache keys

What?

- Release builds use `Mozilla-Actions/sccache-action`
- No direct `cache-restore` step is used
- `sccache` talks directly to the cache service
- We cannot easily evict existing cache entries
- Target: a `reth-mdbx-sys` object built from `mdbx.c`

How?

- Cache key material is deterministic
- BUT: `mdbx.c` uses `__TIME__` and `__DATE__`
- That makes the key depend on build time

`tempo/tempo-xyz`: predicting sccache keys

What?

- Release builds use `Mozilla-Actions/sccache-action`
- No direct `cache-restore` step is used
- `sccache` talks directly to the cache service
- We cannot easily evict existing cache entries
- Target: a `reth-mdbx-sys` object built from `mdbx.c`

How?

- Cache key material is deterministic
- BUT: `mdbx.c` uses `__TIME__` and `__DATE__`
- That makes the key depend on build time
- Future keys can be computed and poisoned

`tempo/tempo-xyz`: release impact

Impact

- Poisoned object code will be linked into the final build output

How to fix this?

`tempo/tempo-xyz`: release impact

Impact

- Poisoned object code will be linked into the final build output
- `__attribute__((constructor))` runs before `main`

How to fix this?

`tempo/tempo-xyz`: release impact

Impact

- Poisoned object code will be linked into the final build output
- `__attribute__((constructor))` runs before `main`
- Official GitHub release artifacts can still be PGP-signed

How to fix this?

`tempo/tempo-xyz`: release impact

Impact

- Poisoned object code will be linked into the final build output
- `__attribute__((constructor))` runs before `main`
- Official GitHub release artifacts can still be PGP-signed
- Users can download a signed, malicious release binary

How to fix this?

`tempo/tempo-xyz`: release impact

Impact

- Poisoned object code will be linked into the final build output
- `__attribute__((constructor))` runs before `main`
- Official GitHub release artifacts can still be PGP-signed
- Users can download a signed, malicious release binary

How to fix this?

- Remove caching from release workflows

`tempo/tempo-xyz`: release impact

Impact

- Poisoned object code will be linked into the final build output
- `__attribute__((constructor))` runs before `main`
- Official GitHub release artifacts can still be PGP-signed
- Users can download a signed, malicious release binary

How to fix this?

- Remove caching from release workflows
- Require approval for all PR workflows

`tempo/tempo-xyz`: release impact

Impact

- Poisoned object code will be linked into the final build output
- `__attribute__((constructor))` runs before `main`
- Official GitHub release artifacts can still be PGP-signed
- Users can download a signed, malicious release binary

How to fix this?

- Remove caching from release workflows
- Require approval for all PR workflows
- Not just for first-time contributors


Reality is often disappointing

`tempo/tempo-xyz`: reality check

What we expected

- Depot Cache is shared between branches

What happened

`tempo/tempo-xyz`: reality check

What we expected

- Depot Cache is shared between branches
- PR workflow writes poisoned entries

What happened

tempo/tempo-xyz: reality check

What we expected

- Depot Cache is shared between branches
- PR workflow writes poisoned entries
- Release workflow restores the same entries

What happened

tempo/tempo-xyz: reality check

What we expected

- Depot Cache is shared between branches
- PR workflow writes poisoned entries
- Release workflow restores the same entries

What happened

- Depot Cache behaves differently in public and private repositories

`tempo/tempo-xyz`: reality check

What we expected

- Depot Cache is shared between branches
- PR workflow writes poisoned entries
- Release workflow restores the same entries

What happened

- Depot Cache behaves differently in public and private repositories
- Private repositories do not use the same cache scope

tempo/tempo-xyz: reality check

What we expected

- Depot Cache is shared between branches
- PR workflow writes poisoned entries
- Release workflow restores the same entries

What happened

- Depot Cache behaves differently in public and private repositories
- Private repositories do not use the same cache scope
- The attack does not work in practice

tempo/tempo-xyz: reality check

What we expected

- Depot Cache is shared between branches
- PR workflow writes poisoned entries
- Release workflow restores the same entries

What happened

- Depot Cache behaves differently in public and private repositories
- Private repositories do not use the same cache scope
- The attack does not work in practice

Sometimes you run into undocumented behavior.

Keeping you safe(r) — I

- Old organizations have insecure defaults.

Keeping you safe(r) — I

- Old organizations have insecure defaults.
- Check your organization settings to look like this:

Keeping you safe(r) — I

- Old organizations have insecure defaults.
- Check your organization settings to look like this:

Default token permission: read repository contents and packages.

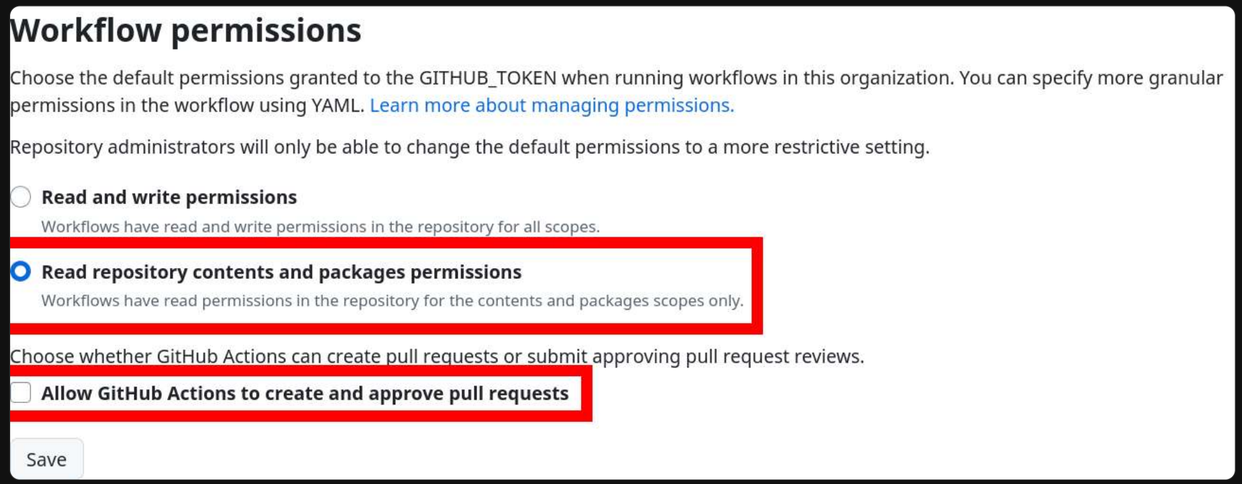
GitHub Actions should not be allowed to create or approve pull requests.

Keeping you safe(r) — I

- Old organizations have insecure defaults.
- Check your organization settings to look like this:

Default token permission: read repository contents and packages.

GitHub Actions should not be allowed to create or approve pull requests.



Keeping you safe(r) — II

- Pin actions to a specific version (commit hash)

Keeping you safe(r) — II

- Pin actions to a specific version (commit hash)

Keeping you safe(r) — II

- Pin actions to a specific version (commit hash)
- Enable CodeQL analysis for GitHub Actions:

github.blog/security/application-security/how-to-secure-your-github-actions-workflows-with-codeql/

Keeping you safe(r) — III

Planned cache hardening

- GitHub is exploring `cache-mode` for workflows and jobs [a]

Why it matters

[a] github.com/orgs/community/discussions/194493

Keeping you safe(r) — III

Planned cache hardening

- GitHub is exploring `cache-mode` for workflows and jobs [a]
- `write`: restore and save (current default)

Why it matters

[a] github.com/orgs/community/discussions/194493

Keeping you safe(r) — III

Planned cache hardening

- GitHub is exploring `cache-mode` for workflows and jobs [a]
- `write`: restore and save (current default)
- `read`: restore only

Why it matters

[a] github.com/orgs/community/discussions/194493

Keeping you safe(r) — III

Planned cache hardening

- GitHub is exploring `cache-mode` for workflows and jobs [a]
- `write`: restore and save (current default)
- `read`: restore only
- `none`: no cache access

Why it matters

[a] github.com/orgs/community/discussions/194493

Keeping you safe(r) — III

Planned cache hardening

- GitHub is exploring `cache-mode` for workflows and jobs [a]
- `write`: restore and save (current default)
- `read`: restore only
- `none`: no cache access

Why it matters

- The cache service enforces the mode

[a] github.com/orgs/community/discussions/194493

Keeping you safe(r) — III

Planned cache hardening

- GitHub is exploring `cache-mode` for workflows and jobs [a]
- `write`: restore and save (current default)
- `read`: restore only
- `none`: no cache access

Why it matters

- The cache service enforces the mode
- `permissions:` does not currently control cache tokens

[a] github.com/orgs/community/discussions/194493

Keeping you safe(r) — III

Planned cache hardening

- GitHub is exploring `cache-mode` for workflows and jobs [a]
- `write`: restore and save (current default)
- `read`: restore only
- `none`: no cache access

Why it matters

- The cache service enforces the mode
- `permissions:` does not currently control cache tokens
- Untrusted workflows can opt out of cache writes

[a] github.com/orgs/community/discussions/194493

Keeping you safe(r) — III

Planned cache hardening

- GitHub is exploring `cache-mode` for workflows and jobs [a]
- `write`: restore and save (current default)
- `read`: restore only
- `none`: no cache access

Why it matters

- The cache service enforces the mode
- `permissions:` does not currently control cache tokens
- Untrusted workflows can opt out of cache writes

Reduce cache-poisoning risk without changing existing workflows by default.

[a] github.com/orgs/community/discussions/194493

Keeping you safe(r) — IV

Workflow execution protections

- Public preview for enterprises, organizations, and repositories [a]

What can you block?

[a] github.blog/changelog/2026-06-18-control-who-and-what-triggers-github-actions-workflows/

Keeping you safe(r) — IV

Workflow execution protections

- Public preview for enterprises, organizations, and repositories [a]
- Admins define who can trigger workflows

What can you block?

[a] github.blog/changelog/2026-06-18-control-who-and-what-triggers-github-actions-workflows/

Keeping you safe(r) — IV

Workflow execution protections

- Public preview for enterprises, organizations, and repositories [a]
- Admins define who can trigger workflows
- Admins define which events are allowed

What can you block?

[a] github.blog/changelog/2026-06-18-control-who-and-what-triggers-github-actions-workflows/

Keeping you safe(r) — IV

Workflow execution protections

- Public preview for enterprises, organizations, and repositories [a]
- Admins define who can trigger workflows
- Admins define which events are allowed
- Rules are evaluated before a run starts

What can you block?

[a] github.blog/changelog/2026-06-18-control-who-and-what-triggers-github-actions-workflows/

Keeping you safe(r) — IV

Workflow execution protections

- Public preview for enterprises, organizations, and repositories [a]
- Admins define who can trigger workflows
- Admins define which events are allowed
- Rules are evaluated before a run starts

What can you block?

- Actors: users, roles, apps, Copilot, Dependabot

[a] github.blog/changelog/2026-06-18-control-who-and-what-triggers-github-actions-workflows/

Keeping you safe(r) — IV

Workflow execution protections

- Public preview for enterprises, organizations, and repositories [a]
- Admins define who can trigger workflows
- Admins define which events are allowed
- Rules are evaluated before a run starts

What can you block?

- Actors: users, roles, apps, Copilot, Dependabot
- Events: `push`, `pull_request_target`, `workflow_dispatch`, ...

[a] github.blog/changelog/2026-06-18-control-who-and-what-triggers-github-actions-workflows/

Keeping you safe(r) — IV

Workflow execution protections

- Public preview for enterprises, organizations, and repositories [a]
- Admins define who can trigger workflows
- Admins define which events are allowed
- Rules are evaluated before a run starts

What can you block?

- Actors: users, roles, apps, Copilot, Dependabot
- Events: `push`, `pull_request_target`, `workflow_dispatch`, …
- Manual-trigger abuse by low-trust users

[a] github.blog/changelog/2026-06-18-control-who-and-what-triggers-github-actions-workflows/

Keeping you safe(r) — IV

Workflow execution protections

- Public preview for enterprises, organizations, and repositories [a]
- Admins define who can trigger workflows
- Admins define which events are allowed
- Rules are evaluated before a run starts

What can you block?

- Actors: users, roles, apps, Copilot, Dependabot
- Events: `push`, `pull_request_target`, `workflow_dispatch`, …
- Manual-trigger abuse by low-trust users
- Misconfigured workflow files across many repos

[a] github.blog/changelog/2026-06-18-control-who-and-what-triggers-github-actions-workflows/

Keeping you safe(r) — IV

Workflow execution protections

- Public preview for enterprises, organizations, and repositories [a]
- Admins define who can trigger workflows
- Admins define which events are allowed
- Rules are evaluated before a run starts

What can you block?

- Actors: users, roles, apps, Copilot, Dependabot
- Events: `push`, `pull_request_target`, `workflow_dispatch`, …
- Manual-trigger abuse by low-trust users
- Misconfigured workflow files across many repos

Central policy beats reasoning about security one YAML file at a time.

[a] github.blog/changelog/2026-06-18-control-who-and-what-triggers-github-actions-workflows/

Resources

- docs.github.com/en/actions/reference/workflows-and-actions/workflow-syntax
- docs.github.com/en/actions/reference/workflows-and-actions/events-that-trigger-workflows
- docs.github.com/en/actions/reference/security/secure-use
- docs.github.com/en/actions/reference/workflows-and-actions/dependency-caching
- ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell
- securitylab.github.com/research/github-actions-preventing-pwn-requests/
- securitylab.github.com/research/github-actions-untrusted-input/
- securitylab.github.com/research/github-actions-building-blocks/
- securitylab.github.com/resources/github-actions-new-patterns-and-mitigations/
- github.blog/2023-08-09-four-tips-to-keep-your-github-actions-workflows-secure/
- github.blog/security/application-security/how-to-secure-your-github-actions-workflows-with-codeql/
- github.blog/changelog/2025-11-07-actions-pull_request_target-and-environment-branch-protections-changes/
- github.blog/changelog/2026-06-18-safer-pull_request_target-defaults-for-github-actions-checkout/
- github.blog/changelog/2026-06-18-control-who-and-what-triggers-github-actions-workflows/
- github.com/orgs/community/discussions/194493
- github.com/AdnaneKhan/Cacheract
- github.com/actions/cache

Thank you!

Questions? Concerns? Comments?