



Insecure GitHub Action Workflows

Finding and exploiting them for fun and profit

Simon Gerst | 26. October 2023

```
sc[] = "\x6a\x0b" // push byte +0xb
// pop eax
// cdq
// push edx
"\x2f\x73\x68" // push dword 0x68
"\x62\x69\x6e" // push dword 0x6e
// mov ebx, esp
// xor ecx, ecx
// int 0x80
```

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 (e.g. 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<https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell>

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<https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell>

Example workflow

```
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<https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell>

Example workflow

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

 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<https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell>

Example workflow

```
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<https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell>

Example workflow

```
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<https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell>

Example workflow

```
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<https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell>

Example workflow

```
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<https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell>

Example workflow

```
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<https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell>

Example workflow

```
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

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<https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell>

Example workflow

```
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

`echo "Hello World_${{github.actor}}"` : Command to run



Example triggers

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

Example triggers

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

```
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

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

Example triggers

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

```
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 }}: ${{ github.
            event.comment.body }}"
```



Example triggers

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



Example triggers

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

```
name: Stale Issues
on:
  schedule:
    - cron: '20_16_*_*_*' # Run 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 }}
          stale-issue-message: 'This issue is stale
            because it has been open 30 days with
            no activity.'
          days-before-stale: 30
```

Example triggers

- 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`

Example triggers

- 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`

```
on:
  workflow_dispatch:
    inputs:
      V8_VERSION:
        description: "V8_VERSION"
        required: false
        type: string
        default: ""
      V8_BUILD_MODE:
        description: "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 for the token: can be set to (permissive) or (restricted)

Permission model for workflows

- Every workflow gets an API token: `GITHUB_TOKEN`
- Base permissions for the token: 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 for the token: can be set to (permissive) or (restricted)
- (permissive): practically read/write access to everything
- (restricted): only read access or no access at all
- If pull request is from a fork: reduce all permissions to a maximum of read access
- *No secrets are available*

Permission model for workflows

- Every workflow gets an API token: `GITHUB_TOKEN`
- Base permissions for the token: can be set to (permissive) or (restricted)
- (permissive): practically read/write access to everything
- (restricted): only read access or no access at all
- If pull request is from a fork: reduce all permissions to a maximum of read access
- No secrets are available
- E.g. `issue_comment`, `issues`, `push`, `workflow_run` give base permissions and secret access.

Permission model for workflows

- Every workflow gets an API token: `GITHUB_TOKEN`
- Base permissions for the token: can be set to (permissive) or (restricted)
- (permissive): practically read/write access to everything
- (restricted): only read access or no access at all
- If pull request is from a fork: reduce all permissions to a maximum of read access
- *No secrets are available*
- E.g. `issue_comment`, `issues`, `push`, `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 workflow file before execution
- Predefined variables: `github`, `env`, `secrets`, ...
- Predefined functions: `fromJson`, `startsWith`, `endsWith`, `contains`, ...

Example workflow — unsafe templating

```
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

```
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

```
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

```
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
- Experienced developers CTF players will notice something:

Example workflow — unsafe templating

```
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
- Experienced developers CTF players will notice something:
- We are mixing *code* and *data*



Example workflow — unsafe templating

```
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`
- Experienced developers CTF players will notice something:
- We are mixing *code* and *data*
- Comment body is `$(touch /tmp/pwned)`

Example workflow — unsafe templating

```
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
- Experienced developers CTF players will notice something:
- We are mixing *code* and *data*
- Comment body is \$(touch /tmp/pwned)

Example workflow — unsafe templating

```
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
- Experienced developers CTF players will notice something:
- 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

```
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

```
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 }}
```

- The shell will *not* interpret the environment variable as code

How to fix this?

- Don't mix code and data
- Instead: Store data in environment variables

```
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}
```

- The shell will *not* interpret the environment variable as code
→ No code injection



Cloning and executing untrusted code

- `on: pull_request_target` (and some other triggers!) gives access to *secrets* and a token with *write* access



Cloning and executing untrusted code

- `on: pull_request_target` (and some other triggers!) gives 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!) gives 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!) gives 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!) gives 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

```
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 *unprivileged* environment XOR clone and don't execute untrusted code in *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 *unprivileged* environment XOR clone and don't execute untrusted code in *privileged* environment
- Sometimes this is not possible (e.g. execute tests and post results as 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 *unprivileged* environment XOR clone and don't execute untrusted code in *privileged* environment
- Sometimes this is not possible (e.g. execute tests and post results as comment)
- *Split* the workflow into two parts:
- *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 *unprivileged* environment XOR clone and don't execute untrusted code in *privileged* environment
- Sometimes this is not possible (e.g. execute tests and post results as comment)
- *Split* the workflow into two parts:
- *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

```
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
```

How to fix this? — Workflow splitting

`pull_request` unprivileged, executes untrusted code

```
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

```
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*

```
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"
```

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

```
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

```

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`!

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` which allows us to 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, that the data is sanitized, 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
- We can have different workflows for different branches

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
- We can have different workflows for different branches
- 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
- We can have different workflows for different branches
- Bug might only be fixed on the default branch!
- Just exploit another, still vulnerable branch 🕶️



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

Resources

- <https://securitylab.github.com/research/github-actions-preventing-pwn-requests/>
- <https://securitylab.github.com/research/github-actions-untrusted-input/>
- <https://securitylab.github.com/research/github-actions-building-blocks/>
- <https://github.blog/2023-08-09-four-tips-to-keep-your-github-actions-workflows-secure/>
- <https://docs.github.com/de/actions/security-guides/security-hardening-for-github-actions>

Questions?

Ping intrigus on Slack

or

DM @intrigus_ on twitter (note the underscore)

or

shoot me a mail at `insecure-gh-actions-23@intrigus.org`