Hardening npm Publishing

サプライチェーン侵害から公開フローを守る

azu (@azu_re)

2026-06-23 / OSS開発者は今何をするべきか？

自己紹介

- Name : azu
- X/Twitter : @azu_re
- Website : Web scratch, JSer.info
- Open Source : textlint / secretlint / HonKit など
- npmに公開しているパッケージは数百個

なぜ今、公開フローを守るのか

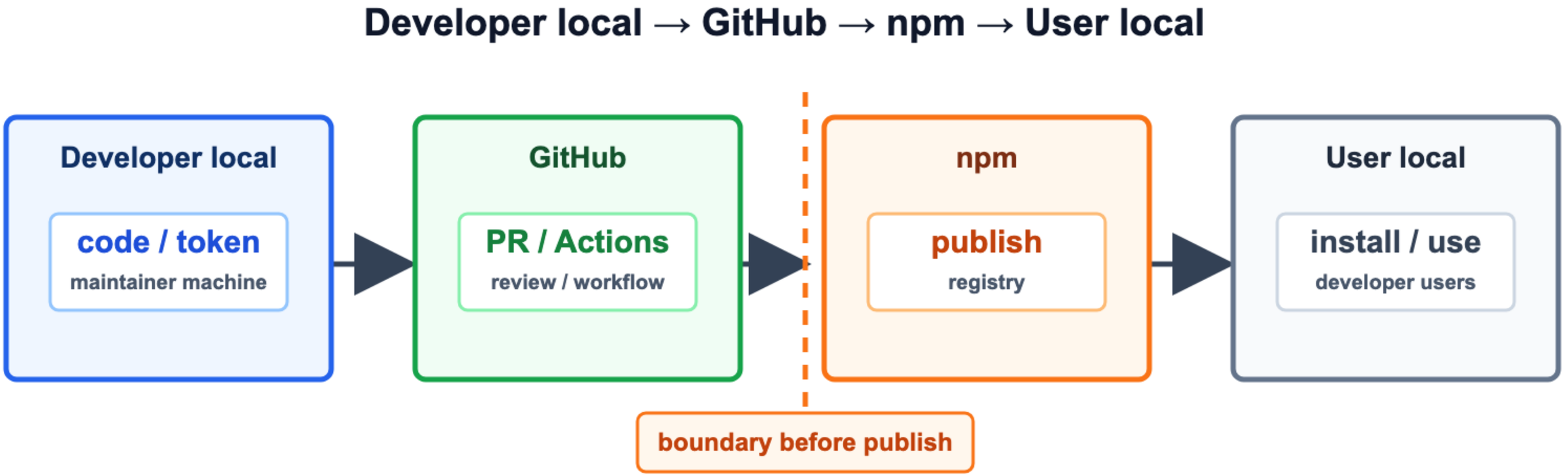
- npmは依存が深く、1つのパッケージ侵害が広い範囲に波及する
- 攻撃の起点がメンテナのトークンやCI/CDに移ってきている
- 脆弱性のあるGitHub ActionsのWorkflowなどが起点となるケースが増えている
- 自分のパッケージが踏み台になる前提で守る

この発表の目的

- 改ざんをゼロにする話ではない
- 侵害されても、悪いpackageがregistryへ出る前に止める
- PRとかの中身が安全かなどを保証する話は主題にしない

問題: 公開フローのどこを守るか

- 攻撃者は、1箇所のインジェクトで全部抜けるなら一番弱い部分を狙う
- 1つの対策だけで公開フロー全体は守れない
- ローカルから公開までの経路を段階ごとに守る必要がある
- どこかが破られても、最終的な公開までには止めることが目的


Developer local → GitHub → npm → User local
Developer local
code / token
maintainer machine
GitHub
PR / Actions
review / workflow
npm
publish
registry
User local
install / use
developer users
boundary before publish

公開までのレイヤー

1. ローカル: Token管理

2. GitHub: Actions（PR / Environments / Deployment protection）

3. npm: staged publishing

1. ローカルのToken管理

生のcredentialをローカルに置かない

- 第一前提は、ローカルの情報を抜かれても公開権限できない
- ローカルファイルに生のcredentialを保存しない
- 1Password/Bitwardenなど多要素認証をしないと取り出せないところへ保存する
- 使う時も、強いトークンをローカルに常駐させない

GitHub

Personal Access Token

GitHub: classic PATを常用しない

- classic PAT(Personal Access Token)は強すぎる権限
- 使うとしてもローカル限定。CIでは使わない
- 常用はfine-grained PATにして、リポジトリと権限を絞る
- fine-grainedはリソースオーナーに紐づくので多少手間だが許容する

fine-grained PATは用途ごとに分ける

- 1つの強いトークンを使い回さず、用途ごとに発行する
- アクセスレベルを最小にする（read-only / 必要な操作だけ）
- 対象を絞る（特定リポジトリ / public のみ）
- 漏れても影響範囲がそのトークンの用途に限定される

 自分の場合は、read-onlyのトークンしか発行していない

課題: Checks APIがfine-grained未対応

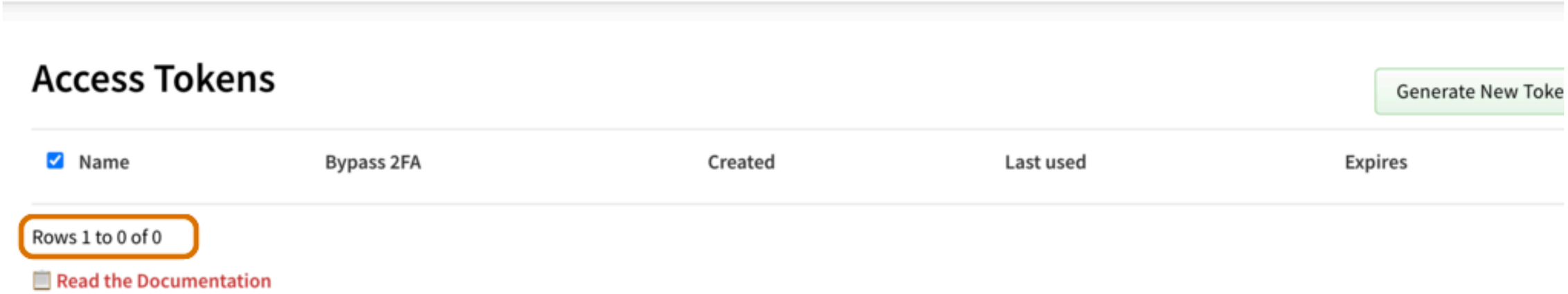
- Checks APIがfine-grained PATに対応していない
- これが直れば、常用トークンから強いclassicをほぼ消せる

参考: github.com/orgs/community/discussions/129512

npm

npmのアクセストークン管理

npm: アクセストークンを0個にする



2. トークンレス npm

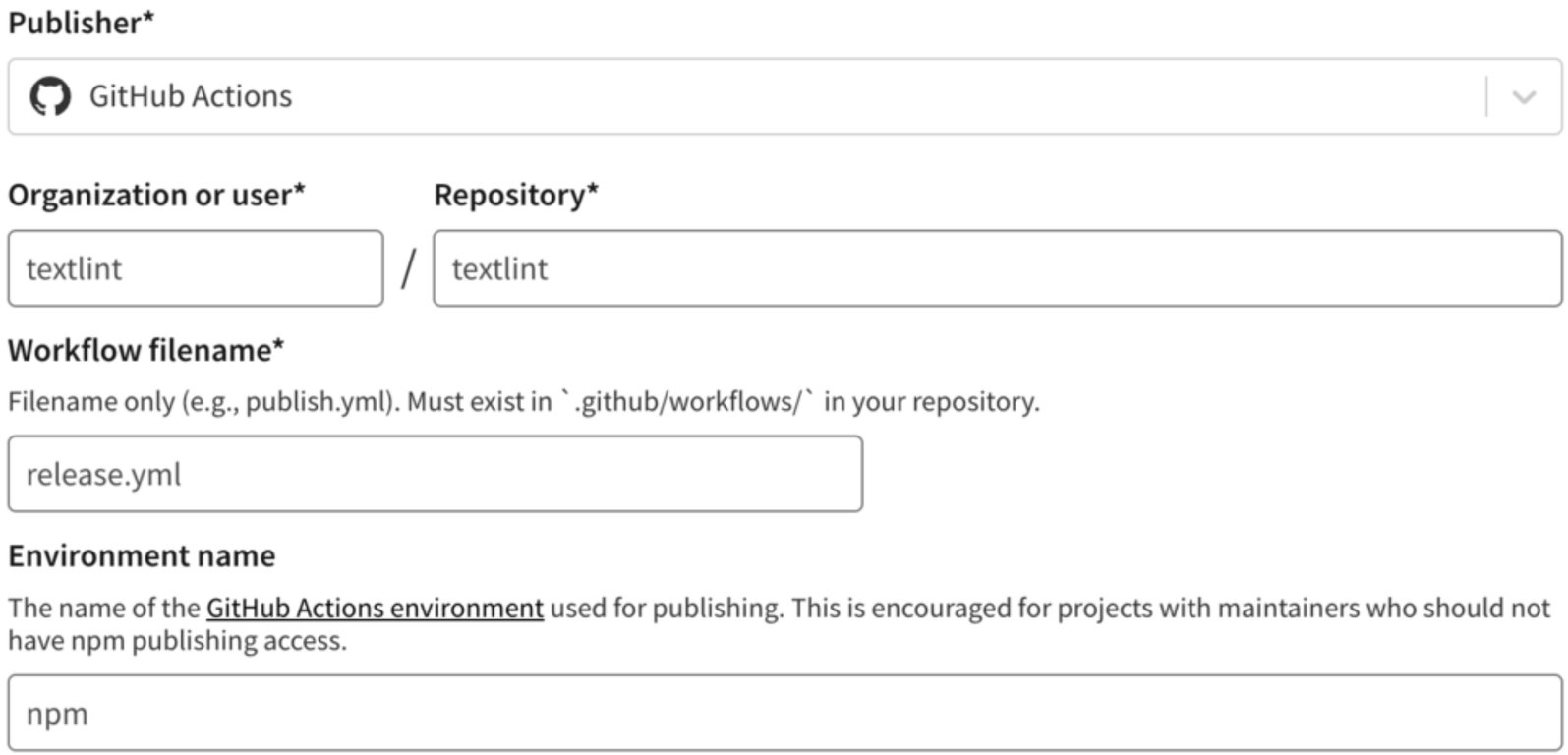
OIDC Trusted Publishing

npm/OIDC Trusted Publishingとは

- 個人に紐づく長期的なトークンをやめ、short-livedでworkflow固有のトークンでパッケージを公開する仕組み
- npmとGitHub ActionsがOIDCでToken Exchangeする
- 「特定リポジトリの特定workflowからの実行」をnpmが確認できる
- npm 11.5.1以上が必要

参考: efcl.info/2025/09/07/npm-oidc/

npmjs.comでTrusted Publisherを登録



npmjs.comでTrusted Publisherの動き

1. GitHub Actionsがnpmに対してOIDC tokenを要求する
2. npmjs.comがOIDC tokenのclaimを確認し、設定されたTrusted Publisher登録と照合する
3. 登録と一致すれば、tokenを発行してGitHub Actionsに返す
4. GitHub Actionsがそのtokenを使って `npm publish` する

workflow側の最小構成

```yaml
permissions:
  contents: write
  id-token: write   # OIDC

steps:
  - uses: actions/setup-node@v5
    with:
      registry-url: 'https://registry.npmjs.org'
  - run: npm publish --access public
```

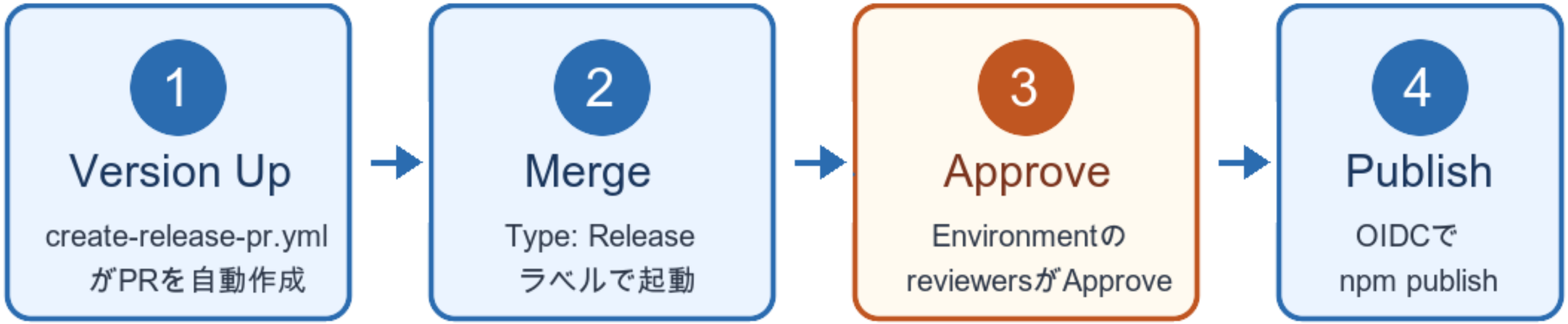
provenanceで来歴を残す

- OIDCでの公開ではprovenance（来歴の署名）が自動付与される
- どのリポジトリのどのworkflowでビルドされたかを証明できる
- npm provenanceはpackageとworkflowを結びつける証拠になる
- ただしbuild中に何が起きたかまでは保証しない

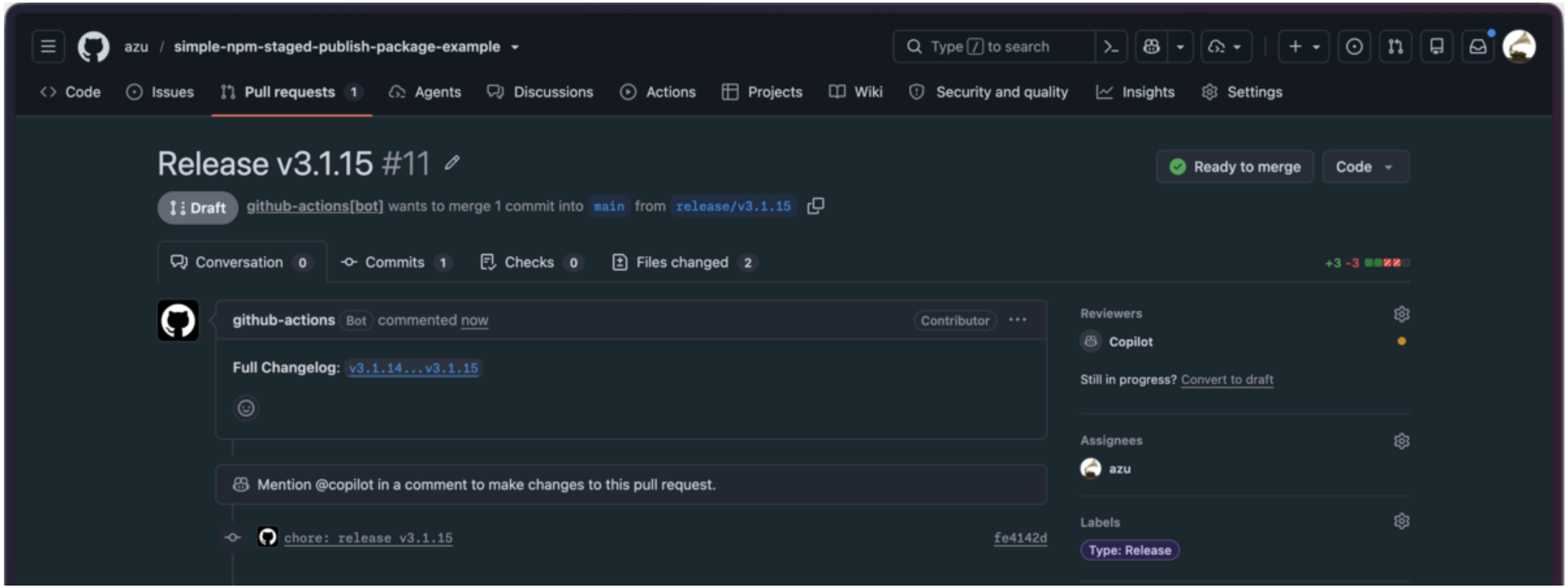
3. GitHub Actions の構成

Rulesets / Environments / Deployment protection

リリースフローの流れ

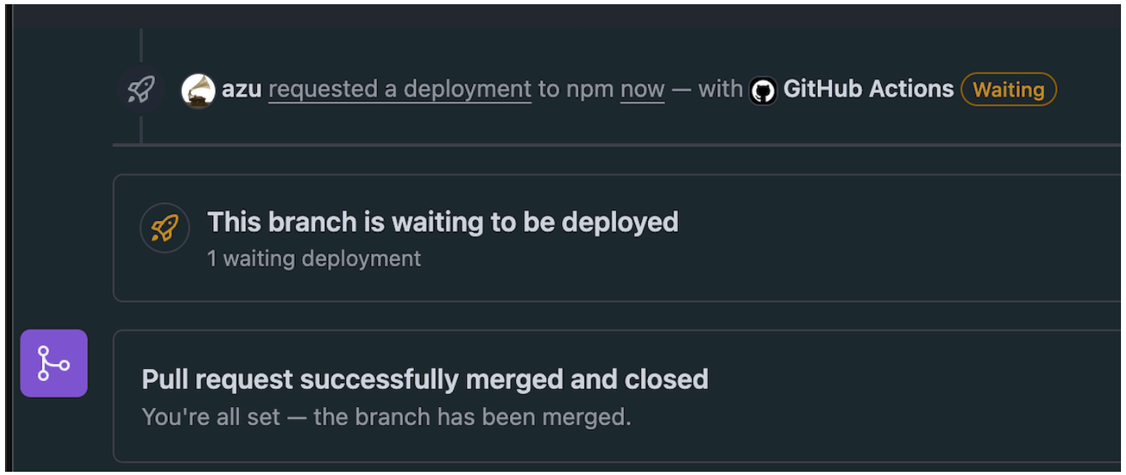

1
Version Up
create-release-pr.yml
がPRを自動作成
2
Merge
Type: Release
ラベルで起動
3
Approve
Environmentの
reviewersがApprove
4
Publish
OIDCで
npm publish

Step 1: Version Up PR

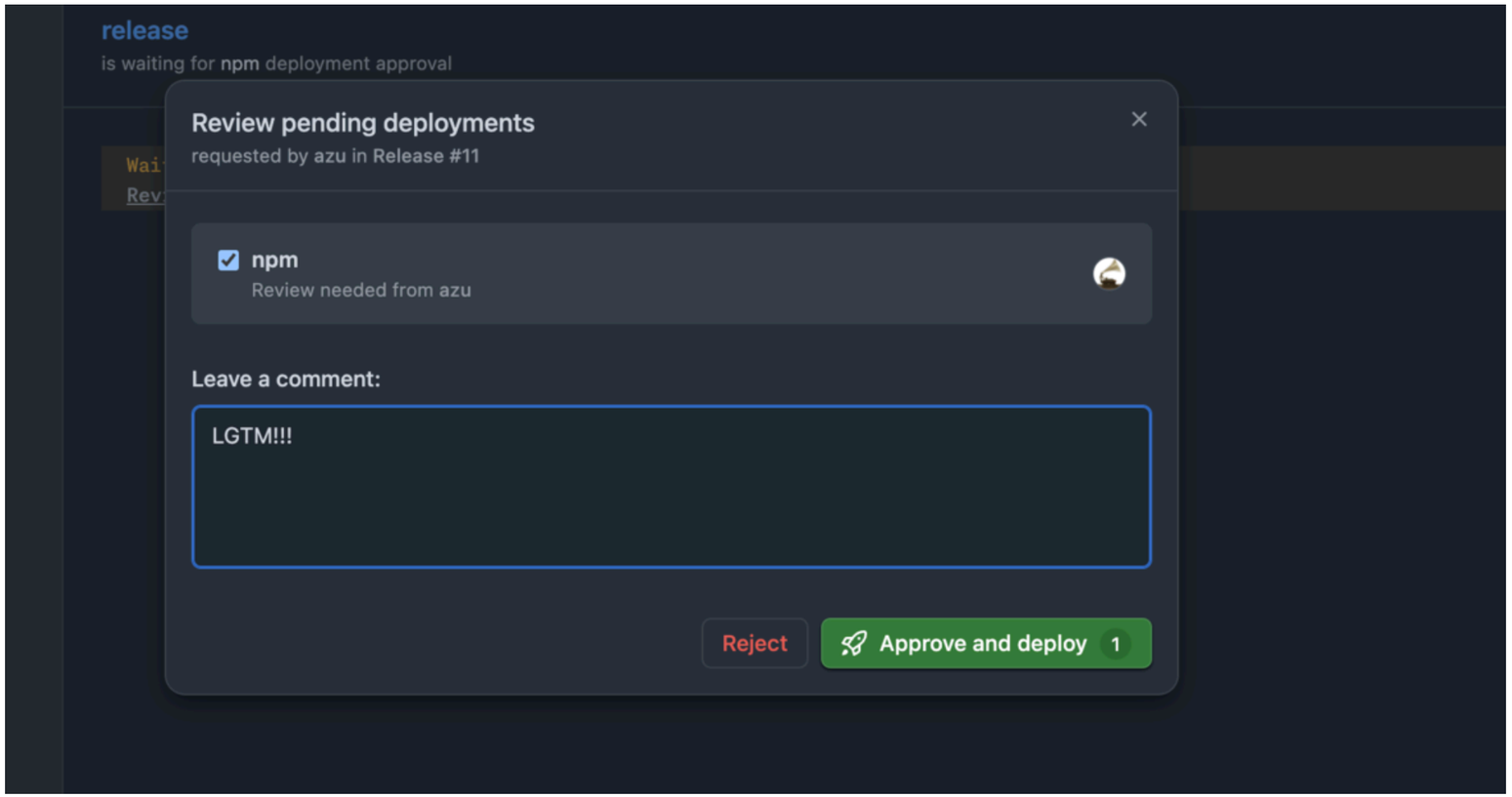


Step 2: Merge Release PR

- Release PRをレビューしてマージする
- `environment: npm`を参照させて、jobの実行を待機させる
  - GitHub Environmentsのフロー(Step 3)
- Approveして初めて`release` jobが動作する



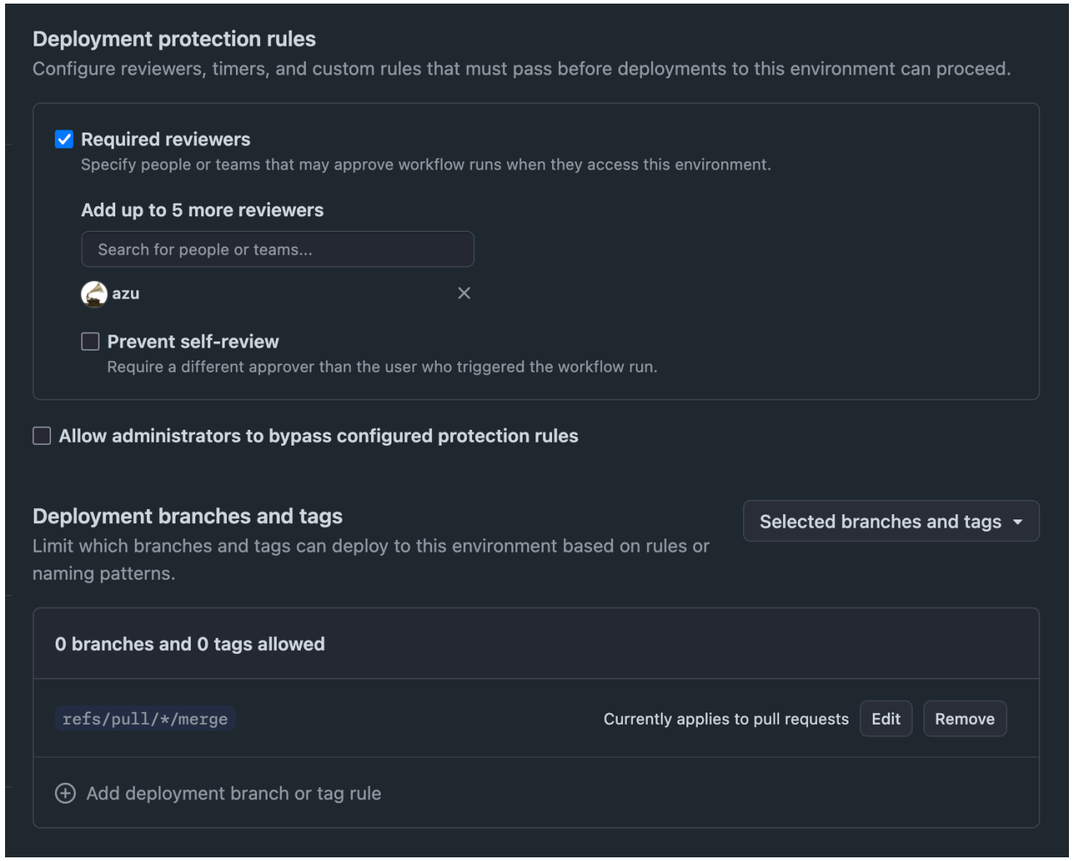
Step 3: Approve Deployment



Step 4: Publish to npm

- Approve後に`release` jobが開始
- OIDCでnpmとtoken exchangeする
- npm publishを実行してパッケージを公開
- provenance付きでregistryへ公開される

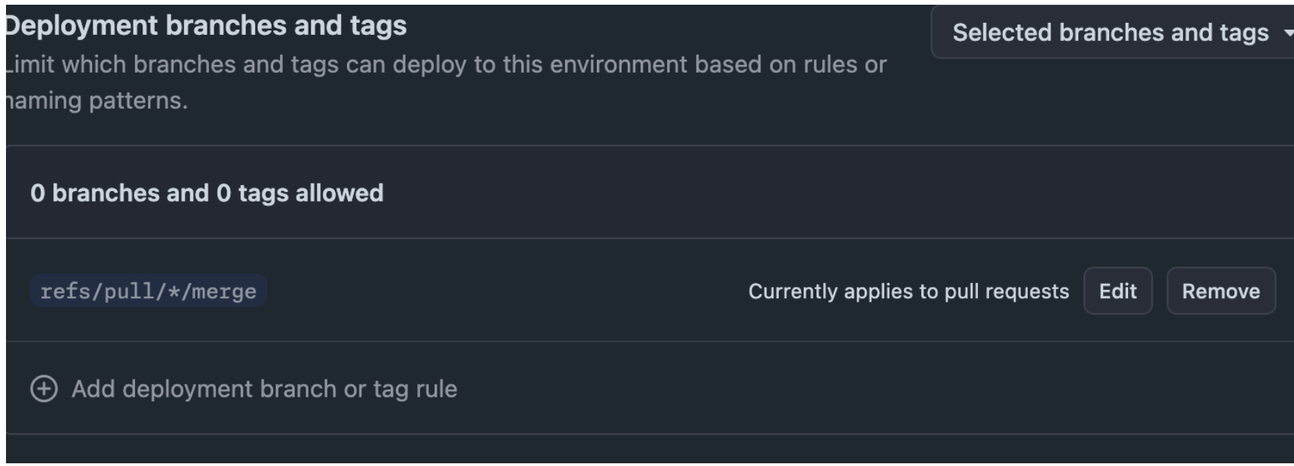
EnvironmentsのDeployment protection rules



Jobの実行をPRをマージしたタイミングのみ許可

- GitHubがPR用のmerge refを作る
- 許可するrefは `refs/pull/*/merge` のみ
  - → 必ずPRを経由しないとWorkflowが実行できない = 目立たせる
- Approveするまで`release` jobは開始しないようにできる

参考: GitHub Docs / GitHub Changelog



GitHub Environments

なぜDeployment protection rulesを使うか

provenanceだけでは足りない

- provenanceで「どこから出たか」は分かる
- でも「作る途中で何が混ざったか」は分からない
- 悪いpackageにも正しい署名が付くことがある
- だからpublishに進む前に別のApproveを求める

参考: Mini Shai-Hulud: Where SLSA's Boundaries Fall

Workflow改変だけではpublishへ進ませない

- npmのTrusted Publisherはworkflowファイル名とEnvironment名を確認する
- Environmentで `refs/pull/*/merge` だけJobの実行を許可
- Environmentで、required reviewersのApproveがJobの実行に必要
- → workflowファイル名一致だけではOIDC交換まで進めない

Environment名をnpm側と一致させる

environment: npm

GitHub Actions / release.yml

```
    if: |
      github.event.pull_request.merged == true &&
      contains(github.event.pull_request.labels.*.name, 'Type: Release')
    environment:
      name: npm
    permissions:
      contents: write
      id-token: write  # OIDC
    outputs:
      released: ${{ steps.tag-check.outputs.exists == 'false' }}
      version: ${{ steps.package.outputs.version }}
      package-name: ${{ steps.package.outputs.name }}
      release-url: ${{ steps.create-release.outputs.url }}
    steps:
      - name: Checkout
        uses: actions/checkout@08c6903cd8c0fde910a37f88322edcfb5dd907a8 # v5.0.0
        with:
          persist-credentials: false

      - name: Get package info
```

npm

Workflow filename*

Filename only (e.g., publish.yml). Must exist in `.github/workflows/` in your repository.

release.yml

Environment name

The name of the GitHub Actions environment used for publishing. This is encouraged for projects with maintainers who should not

npm

事例: Workflow改変 + OIDCの問題

- Bitwarden CLIのサプライチェーン攻撃の事例
- workflow改変 + 直接pushでOIDC credentialを取得
- credentialを持ち出して、悪意あるパッケージを公開
- GitHub write権限がnpm publish権限へ広がる
- 境界にApproveを置く

参考: Bitwarden ソフトウェアサプライチェーン攻撃の概要と対応指針 - GMO Flatt Security Blog

4. npm staged publishing

公開前にApprove stepを追加する

- npm publish は直接公開、npm stage publish はApprove待ちにする
- パッケージ設定でpublishとstage publishを許可するか選べる
- stage publishのみ許可にすると、直接公開は拒否される
- npm 11.15.0以上 / Node 22.14.0以上、OIDC時のみ利用可

参考: docs.npmjs.com/staged-publishing

staged publishingのフロー

Staged PackagesのApprove画面

staged publishingの課題

- Approveが1パッケージずつしかできない
- monorepo（数十個一括）だと現実的でない
- CLIからapproveするにはnpmトークンが要る
- npm / GitHubのどちらか一方はtokenlessに寄せる


公開経路
Local
maintainer
PR
review
Actions
workflow
npm
publish
Registry
package
Users
install
確認点
PR Review
mainへ入る前
Environment
Approve
publish権限へ進む前
staged
Approve
registry公開前
対策
token 0
長期tokenを置かない
OIDC
短期credential
Environment
publish前にApprove
staged
公開前にApprove

npm publishを難しくする

- GitHub → npm 境界 + npm publishにもMFAを求める
- 攻撃が成功するには、GitHubとnpmのアカウントを同時に侵害する必要がある
- npmにはセキュリティのみがMFAとして登録している
-  GitHubはTOTPが削除できないバグがある
- How to remove authenticator app · community · Discussion #54699

まとめ

公開フローを段階ごとに制御する

1. ローカル: 強い権限を常駐させない

2. Actions: PR + Environment + Approve

3. staged publishing: registry公開前にApprove

AIエージェント時代も同じ

- 全権限を1つの主体/環境に集めない
- AIが全部の権限を持っているなら、攻撃者はAIを狙うだけ
- 最小権限と権限分離はやる必要がある

ありがとうございました

参考リンク

- npm OIDC: efcl.info/2025/09/07/npm-oidc/
- 公開例: github.com/azu/simple-oidc-example-package
- staged例: github.com/azu/simple-npm-staged-publish-package-example
- Bitwarden CLI侵害: GMO Flatt Security Blog
- TanStack侵害(cache poisoning): tanstack.com/blog/npm-supply-chain-compromise-postmortem
- Mini Shai-Hulud(SLSAの境界): slsa.dev/blog/2026/05/mini-shai-hulud-what-slsa-can-and-cannot-do
- npm staged publishing: docs.npmjs.com/staged-publishing
- GitHub changelog: `pull_request_target` and environment branch protections

その他

Require 2FA and disallow tokens

Publishing access

Requiring an additional authentication method adds another level of security for your package.

- ○ Require two-factor authentication or a granular access token with bypass 2fa enabled
- ◉ Require two-factor authentication and disallow tokens (recommended)

Note about trusted publishers: All publishing access options above are compatible with OIDC trusted publishers. If you have configured trusted publishers for this package, they will continue to work regardless of which option you select. For maximum security, we recommend using trusted publishers with the most restrictive token option.

Update Package Settings

Require 2FA and disallow tokens

- npm Trusted Publisherとセットで設定する
- これを設定すると、npmのアクセストークンでのpublishができなくなる
- OIDCでないとpublishできない状態にする = CIからの公開に寄せられる

Cache Poisoning

- Release Workflowではキャッシュを使わない
- Cache Poisoning攻撃を防ぐため
- リリースに関するWorkflowはactions/checkoutとactions/setup-nodeだけ
- 他はghやスクリプトを書くことで、外部依存は可能な限り減らす

Workflow execution protections

- Workflow execution protections
- 「誰が」が「どのイベント」のWorkflowを発火できるかを制御できるポリシー
- 特定のWorkflowだけを対象にはできないので、ちょっと大雑把