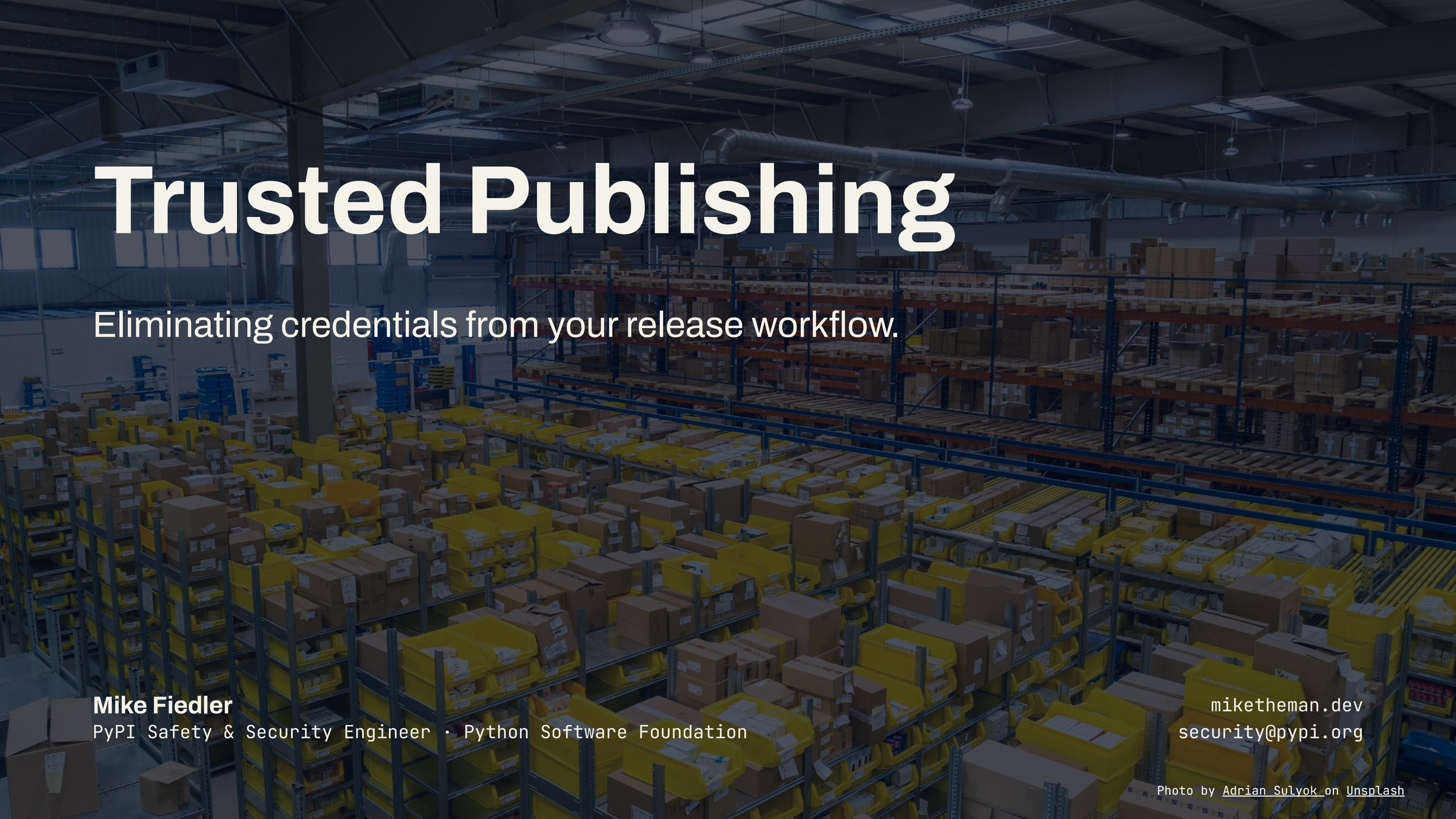


Trusted Publishing

Eliminating credentials from your release workflow.

Mike Fiedler
PyPI Safety & Security Engineer · Python Software Foundation

miketheman.dev
security@pypi.org

Photo by Adrian Sulyok on Unsplash

HI, I'M

↑ that's me

Mike Fiedler

PyPI's first-ever Safety & Security Engineer · on duty since August 2023 · funded by Alpha-Omega.

- → miketheman.dev
- → security@pypi.org
- → @miketheman

python™ Package Index

THE REGISTRY NEVER SLEEPS

Things are getting bigger, *fast*

13B

REQUESTS / DAY

Every day. pip install, security scanners, CI builds.

10PB

PER/DAY · ~1 TBPS PEAK

Thank you, Fastly. We would not exist without you.

800+

NEW PROJECTS / DAY

Also 1M+ accounts and 700k+ packages and counting.

THE STORY IN ONE CHART

2,200,000+ files published with Trusted Publishing

Across 66,000+ projects - in Feb 2024 Trusted Publishing was 10% of uploads. May 2026: **over 36%**.

PART 01

The problem with credentials

Every stolen token in the last decades had one thing in common:

it existed long enough to get stolen.

THE RELEASE-DAY RITUAL

You've done this dance

~/.pypirc

On your laptop

```
[pypi]
username = __token__
password = pypi-AgEIcHlwaS5vcmci...
```

GitHub Actions secret

In your CI

```
secrets:
  PYPI_API_TOKEN: pypi-AgEI...
# pasted in once, two years ago
```

env var

Everywhere else

```
export TWINE_PASSWORD=pypi-AgEI...
twine upload dist/*
```

“ A long-lived API token is just a password in a nicer hat.”

- a thing I keep saying out loud

WHERE LONG-LIVED TOKENS GO TO DIE

Four failure modes, all real

01

Committed to git

02

Printed in CI logs

03

Stolen by a dep

04

Phished

CASE STUDY · DECEMBER 2024

Ultralytics · a poisoned build

> **A tainted `ultralytics` release shipped to millions of users.**

The attacker never needed the password. They just needed the CI system to trust them for a few seconds.

FURTHER READING blog.pypi.org/posts/2024-12-11-ultralytics-attack-analysis/

CASE STUDY · NOVEMBER 2025-ONGOING

Shai-Hulud · the worm

> **The worm scanned for API tokens. Projects using Trusted Publishing had none to steal.**

Hundreds of npm packages compromised in hours.

PyPI's blast radius was a fraction.

FURTHER READING blog.pypi.org/posts/2025-11-26-pypi-and-shai-hulud/

CASE STUDY · APRIL 2026

LiteLLM & Telnyx · credential cascade

> **One leaked npm secret. Two ecosystems compromised.**

Both came through the same Trivy exploit chain. Long-lived tokens don't respect ecosystem boundaries.

FURTHER READING blog.pypi.org/posts/2026-04-02-incident-report-litellm-telnyx-supply-chain-attack/

THREE INCIDENTS, ONE PATTERN

Someone held a long-lived secret. Someone else got it.

Fix the class of bug, not the instance.

PART 02

How Trusted Publishing works

OIDC - OpenID Connect - lets your CI *prove* who it is to PyPI, every time it publishes.

Trust the pipeline,
not the password.

OIDC, IN PLAIN TERMS

Three properties of the new, but same, token

01

Short-lived

`~ 15 minutes`

Useless to an attacker tomorrow.

02

Signed

`GitHub · GitLab · Google`

PyPI verifies the signature against a public key it already trusts.

03

Scoped

`one repo · one workflow`

Can only upload to the projects you explicitly connected.

THE EXCHANGE

What actually happens on publish

CI runner	Identity Provider	PyPI
GitHub Actions	GitHub OIDC	upload endpoint

1 "Give me an ID token"

2 signed JWT `(iss, sub, repo, workflow)`

3 "Trade this JWT for an upload token"

4 verify + issue 15-min upload token

5 **twine upload** no secret ever stored on disk

BEFORE · THE STATUS QUO

A token lives in your repo, forever

The token is valid until you remember to rotate it. You won't remember.

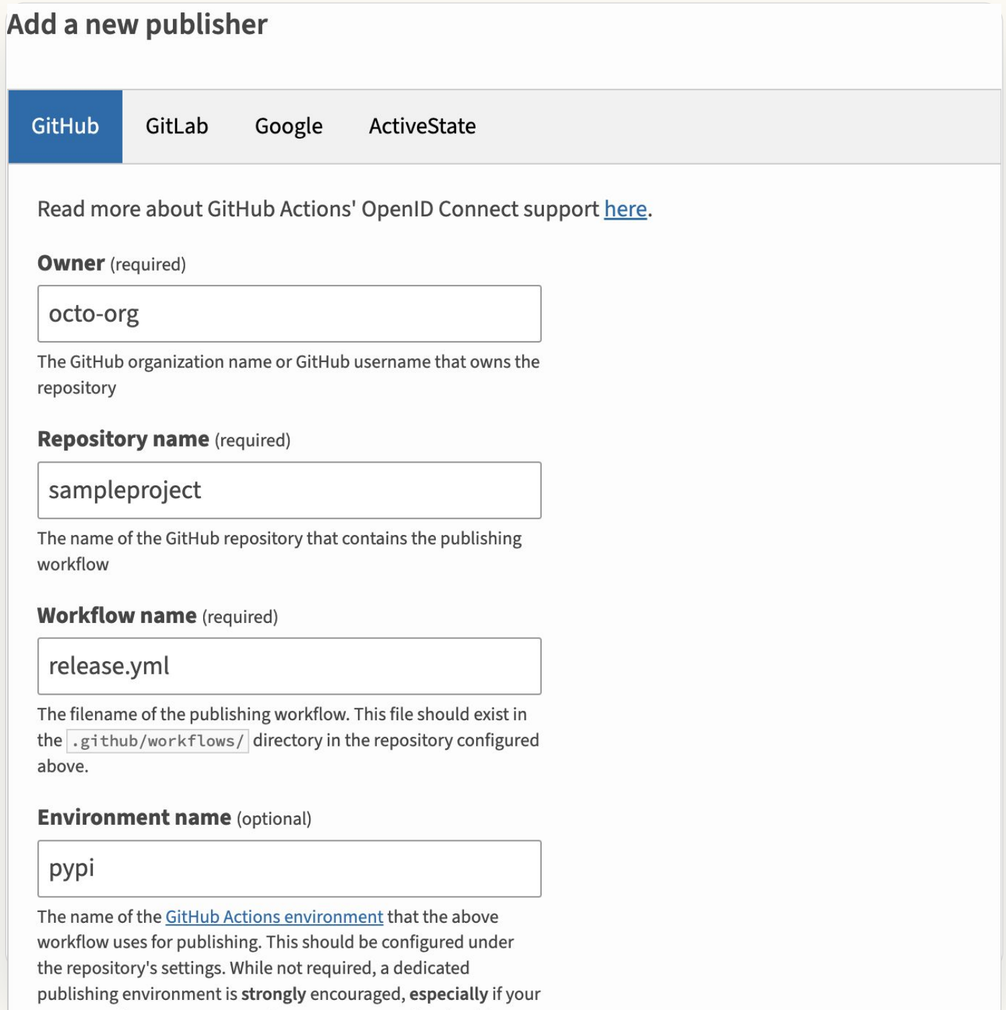
AFTER · TRUSTED PUBLISHING

A token is minted on the fly, and dies in 15 minutes

No secret stored. No secret to leak. Every publish earns its trust, fresh.

SET UP · 1 OF 3 · ON PYPI

Add a publisher to your project



`pypi.org/manage/project/<pkg>/settings/publishing`

OWNER	GitHub org or user
REPOSITORY	the repo that publishes
WORKFLOW	`release.yml`
ENVIRONMENT	optional gate

Four strings. That's the whole security model.

SET UP · 2 OF 3 · WHAT THOSE FIELDS ACTUALLY MEAN

PyPI will only issue a token when all four match

FIELD	WHAT PYPI CHECKS	WHY IT MATTERS
owner	`repository_owner`	One GitHub org or user.
repository	`repository`	One specific repo.
workflow	`job_workflow_ref`	One specific workflow file.
environment	`environment`	Optional gate.

SET UP · 3 OF 3 · IN YOUR REPO

Twelve lines of YAML

.github/workflows/release.yml

```yaml
name: publish
on:
  release: { types: [published] }

jobs:
  publish:
    runs-on: ubuntu-latest
    environment: pypi          # matches PyPI "environment" field
    permissions:
      id-token: write        # ← asks GitHub for an OIDC JWT
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
      - run: python -m build
      - uses: pypa/gh-action-pypi-publish@release/v1
        # no password. no token. no secret to phish.
```

ENVIRONMENTS · THE EXTRA DEADBOLT

A GitHub environment gates *who* and *when*

- ✓ Required reviewers
- ✓ Protected branches only
- ✓ Wait timers
- ✓ Environment-scoped secrets

Optional on PyPI. Strongly encouraged in practice.

NOT JUST PYPI

Six registries, same pattern

PyPI
PYTHON

npm
JAVASCRIPT

RubyGems
RUBY

crates.io
RUST

NuGet
.NET

Packagist
PHP

NOT JUST GITHUB ACTIONS

Identity providers PyPI accepts

SUPPORTED

GitHub Actions

SUPPORTED

GitLab CI/CD

SUPPORTED

Google Cloud

SUPPORTED

ActiveState

BETA

GitLab Self-Managed

GLSM & GHES

IN REVIEW

CircleCI

COMMON PITFALLS · WHAT WILL BITE YOU ON DAY ONE

Three things to check first

01 Missing `id-token: write`

Results in a vague 403. The fix is one line.

02 Workflow filename mismatch

`release.yml` vs. `releases.yml` - one character, the JWT won't match.

03 Running from a fork PR

Fork PRs don't get `id-token` by design. Good. Also surprising.

UNDER THE HOOD

What PyPI actually sees

decoded JWT payload (abridged)

```
{
  "iss": "https://token.actions.githubusercontent.com",   # who signed it
  "aud": "pypi",                                          # for whom
  "sub": "repo:your-org/your-repo:environment:pypi",
  "repository": "your-org/your-repo",
  "repository_owner": "your-org",
  "job_workflow_ref": "your-org/your-repo/.github/workflows/release.yml@refs/tags/v1.4.0",
  "environment": "pypi",
  "ref": "refs/tags/v1.4.0",
  "sha": "a3b1f2c…",
  "exp": "2026-04-22T19:04:00Z"                 # ~15 minutes from now
}
```

PART 03

Security benefits

When there's no token, the whole threat model changes.

Let's replay the last three incidents.

WHAT CHANGES WHEN THERE'S NO TOKEN?

Entire attack categories just disappear

ATTACK	WITH A LONG-LIVED TOKEN	WITH TRUSTED PUBLISHING
Committed to git	Attacker publishes	Nothing to commit
Dumped in CI logs	Scraped in seconds	Nothing to scrape
Read by a bad dep	Exfil → upload	No env var
Maintainer phished	Session + token	Session alone fails
Worm scans for secrets	Finds it	Finds nothing

ULTRALYTICS · REPLAYED, WITH TRUSTED PUBLISHING

Where the attack would have stopped

✗

PR branch ≠ trusted workflow ref

No matching `job_workflow_ref` → no OIDC token.

✗

No secret to exfiltrate

You can't steal a publisher record on PyPI.

✗

Environment reviewer gate

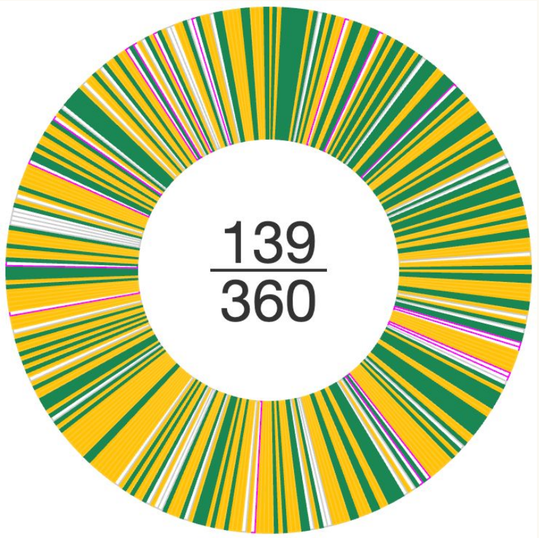
A human still has to click approve.

PEP 740 · SIGSTORE

Every artifact ships with a provenance receipt

WHO	repo + workflow
WHEN	commit SHA + timestamp
PROOF	Sigstore · Rekor log

Forensic archaeology → database query.



Are we PEP 740 yet? - live tracker of the 360 most-downloaded PyPI packages.

trailofbits.github.io/are-we-pep740-yet

THE MOST OVERLOOKED STEP

Setting up Trusted Publishing is only half the fix…

Delete the old token!

Belt and suspenders, only the suspenders are on fire.

A trusted publisher *and* a leftover `PYPI_API_TOKEN` means your threat model is still the token. The attacker will always choose the weaker door.

NUANCE

When an API token is still the right call

- ✓ **Self-hosted CI** without OIDC
- ✓ **Manual or emergency release**

Scope it narrow. Make it short-lived. Delete it after.

Bootstrap? Use a **Pending Publisher** instead - it converts to a real one on first use.

PART 04

Migration path

From "we have a token in a secret" to "we don't, and we never will again." Five steps. One afternoon, for most projects.

MIGRATION · FIVE STEPS

One project, start to finish

01

Add the publisher on PyPI

Manage project → Publishing → Add new publisher.

02

Add `id-token: write` to the workflow

Swap `twine upload` for `pypa/gh-action-pypi-publish`.

03

Cut a test release

A `.dev0` tag is fine. Confirm the wheel lands on PyPI.

04

Revoke the old token

On pypi.org/manage/account. Delete the GitHub secret. The step most people skip.

05

Update your process docs

Tell your contributors. Tell your future self.

AT SCALE · IF YOU MAINTAIN DOZENS OF PACKAGES

Two shortcuts (for now)

① **Wait for org-level publishing**
coming soon

② **Until then:** script the browser setup, or do it per-project

Not yet: reusable workflows are WIP. No public API for creating publishers.

THE ASK

Remove one token this week

Pick your noisiest package.

Set up Trusted Publishing.

Delete the old secret.

HOW YOU CAN SUPPORT THIS WORK

PyPI security runs on funded time

DIRECT

Python Software Foundation

A non-profit with 11 employees stewarding Python, PyPI, and the community.

python.org/psf/donations

CROSS-ECOSYSTEM

Alpha-Omega

Funds security engineers across the most critical open-source ecosystems - including my role.

alpha-omega.dev

Thanks!

Questions?

web	miketheman.dev
mail	security@pypi.org
blog	blog.pypi.org
docs	docs.pypi.org/trusted-publishers

Funded by Alpha-Omega · alpha-omega.dev

miketheman.dev - hi!

python™ Package Index