TRAIL OF BITS

PEP 740 and PyPI: Bootstrapping Provenance for the Python Ecosystem

William Woodruff, Trail of Bits

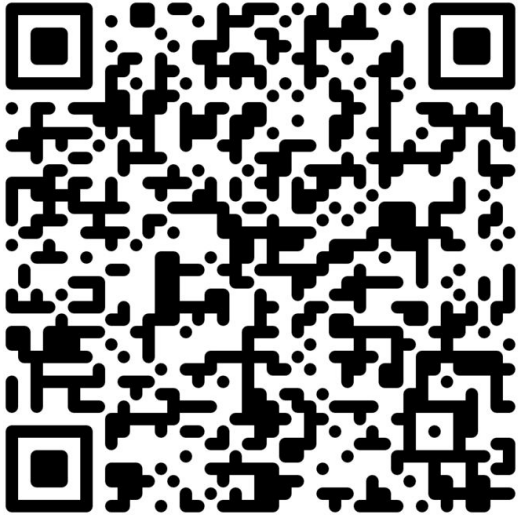
Hello!

- **William Woodruff (william@trailofbits.com)**
  - open source group engineering director @ trail of bits
  - long-term OSS contributor (Homebrew, LLVM, Python) and maintainer (pip-audit, sigstore-python)
  - @yossarian@infosec.exchange
- **Trail of Bits**
  - ~130 person cybersecurity engineering and auditing consultancy
  - specialities: cryptography, compilers, program analysis research, “supply chain”, OSS package management, general high assurance software development

PSA / plug

SigstoreCon 2024! Next month!

- **November 12**
- **Salt Lake City**
- **Co-located with (before) KubeCon NA**
- **Topics:**
  - Sigstore (duh)
  - TUF & TUF implementations
  - SBOMs
  - All other supply chain goodness



the entire talk, in one slide

- **PyPI has had a strong notion of source identity via *Trusted Publishing* since 2023**
  - widely successful because it was a ***usability win*** for Python developers, not just a security win
- **we can *bootstrap* signed provenance on top of Trusted Publishing's identity primitives**
  - in the form of Sigstore bundles with containing SLSA, etc. attestations
  - ***and*** we can do this without changing Trusted Publishing's usability wins!
- **in fact, we already did it, and it's public!**
  - and ***you*** can enable it today (before it becomes the default)

agenda

- **background on PyPI/Python packaging/Python community**
- ***systematically* securing Python packaging**
  - or: “treating security first and foremost as a usability problem”
- **intro to Trusted Publishing**
- **from Trusted Publishing to provenance**
  - PEP 740 & Sigstore
- **tying it all together**
- **what’s next?**

PyPI

- **the Python Package Index**
  - “Pie-pea-eye”
- **the index behind `pip install`**
- **~860K users, ~575K projects, ~6.1M releases, ~12.2M files**
- **~51B downloads/month, >2TB traffic/day**
  - inflated by tools and large CI/CD providers not caching as much as they could!

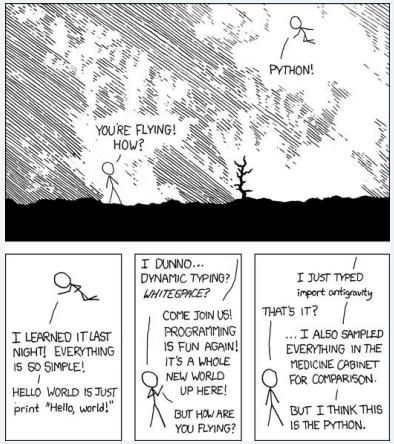
Downloads last day: 2,016,247,697
Downloads last week: 11,901,433,314
Downloads last month: 50,946,221,288

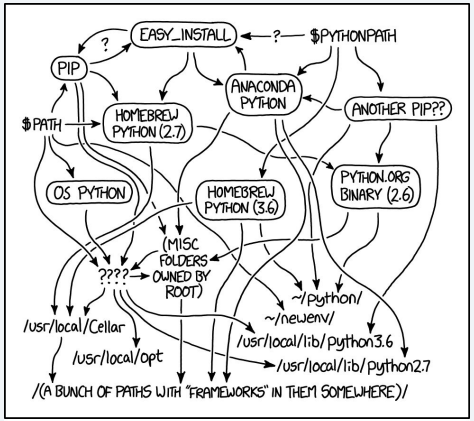
Python packaging/community

- **PyPI is just one piece of the Python packaging constellation**
  - installers: `pip`, `uv`, `poetry`, etc.
  - build tools and backends: `setuptools`, `hatch`, `flit`, `poetry`, etc.
  - upload tools: `twine`, `uv`, `poetry`, etc.
  - very few single tools/toolchains do everything, although this is slowly changing
- **Python packaging is standards driven**
  - Python Enhancement Proposals (PEPs) become living PyPA standards once accepted
  - **not** top-down or tool-driven, unlike Rust (`cargo`) and JS (`npm`)
- **Python's community is massive and diverse**
  - every possible combination of experience, background, interest, skill, time commitment, etc.
  - this makes it hard to establish a common denominator for *anything*, much less security!
  - more closely resembles a country/countries than most other ecosystems!

systematically securing Python packaging

- **diversity is both Python's strength and also a challenge to *systematic* security changes**
  - a lack of systematic planning is ***self-fulfilling***, since new systematic plans need to accommodate conventions/expectations
  - systematic planning ***can be dangerous*** to the things that keep Python's community healthy: onerous security requirements scare newcomers away!





MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

systematically securing Python packaging

to make systematic progress on packaging security, we need to treat security as a *usability* problem!

two core demographics:

- **enthusiasts: 1-5% who will take the initiative to opt into things**
  - good news: most of the top packages are maintained by this demographic!
  - bad news: they're still a tiny overall percentage of package maintainers
- **everyone else: 95-99% who only care about the reliability of their workflow; security changes must not require significant behavioral changes unless those changes make things easier for them**
  - good news: ***if*** we make things easier for them, then they'll happily adopt changes

one big problem

codesigning is the exact opposite of this:

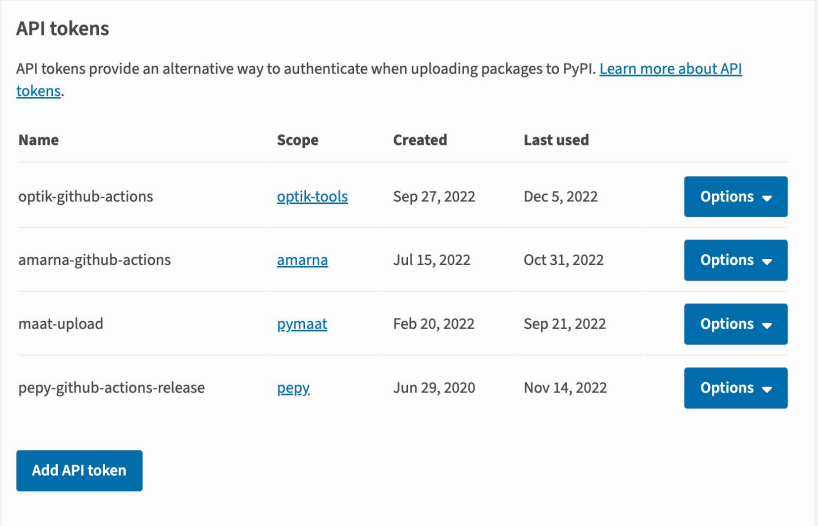
- **publishers must generate and and *keep secure* long-term signing keys**
  - spoiler: almost nobody does this
- **index (PyPI) must accept signatures for uploaded files**
  - if the index validates signatures, requires a key directory = more operational complexity
  - if not, users will upload random garbage in the shape of a signature blob
- **impossible to default consumers into**
  - consumers must establish keys + trusted identities on their own, can't be done for them
  - implies all of the operational mess of PKIs: key expiry, revocation, etc.

***any* approach to digital signatures on PyPI has to avoid *all* of these problems!**

Trusted Publishing

trusted publishing: background

- **since 2019, PyPI uses API tokens for upload authentication**
  - user/password was used before that
- **API tokens are created manually and can be scoped to projects**
  - results in a ***chicken-and-egg problem***: to create a properly scoped token, the project needs to already exist
  - ...meaning that users have to use insecure wide-scoped tokens to first create their projects
  - users then forget/neglect to make a new scoped token, defeating the whole point!



Trusted Publishing: background

- **API tokens are *good*, but we can do *better*!**
- **goals:**
  - ***misuse-resistance***: tokens should be ***self-scoping*** and minimally scoped
    - no accidental user-wide scoping
  - ***compromise-resistance***: tokens should be ***self-expiring***
    - prevents an attacker from hoarding credentials for future use
  - ***fatigue-resistance***: tokens should require as few context switches as possible
    - one configuration phase on PyPI instead of 2+

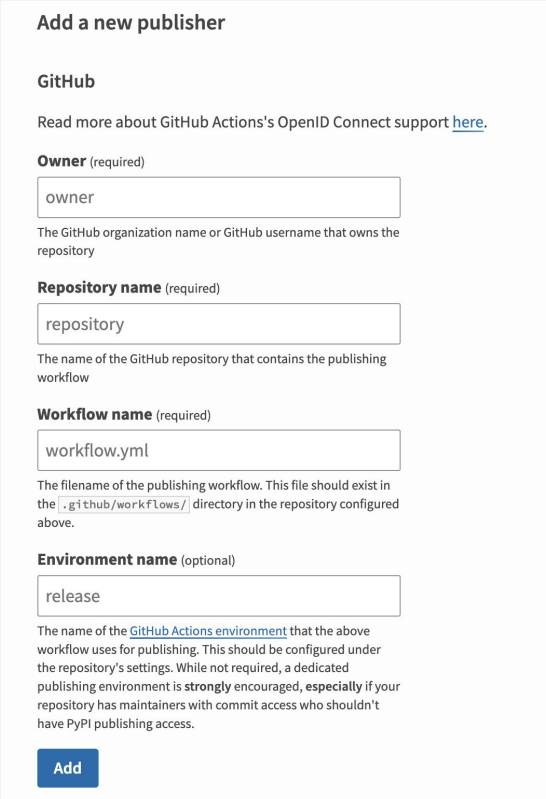
***trusted publishing* is our solution to the above**

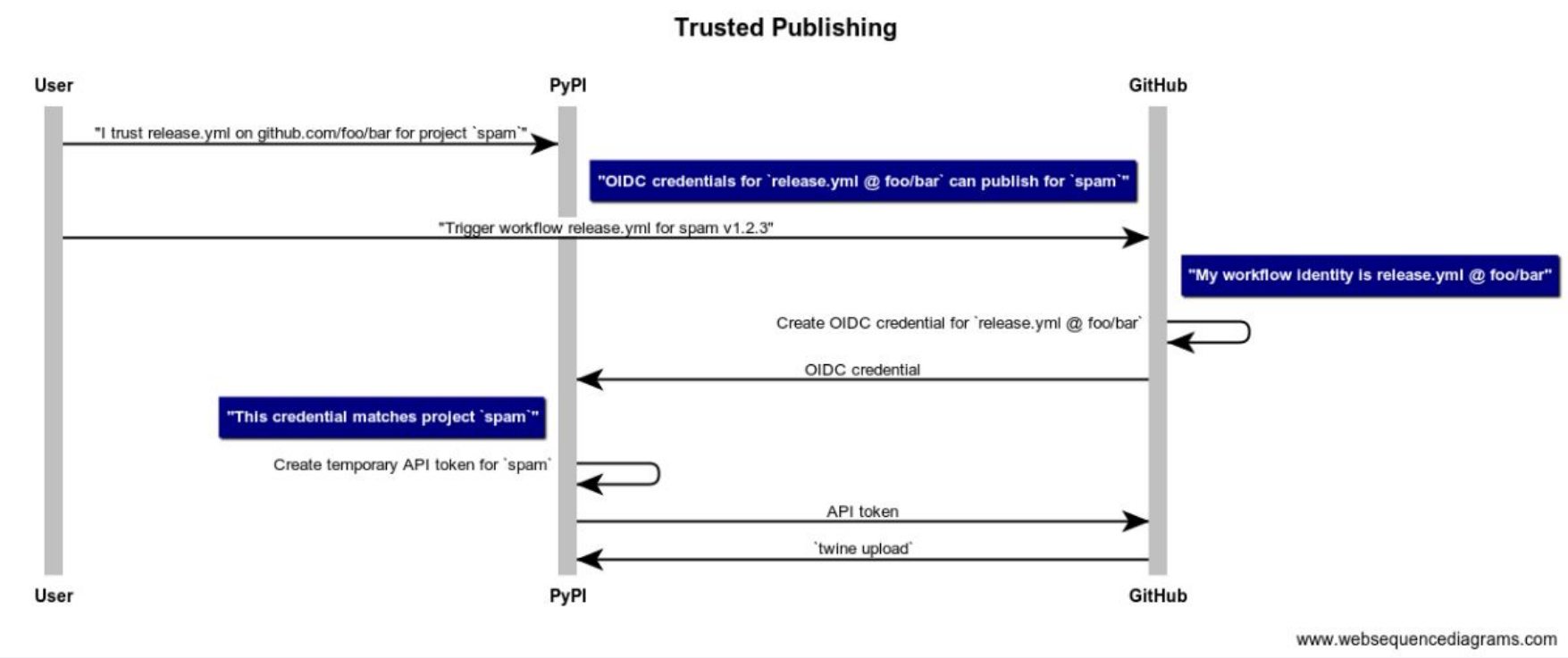
Trusted Publishing

- **basic idea: use OpenID Connect to authenticate to PyPI**
- **CI/CD providers like GitHub and GitLab allow workflows to generate OIDC credentials**
  - these credentials are ***publicly verifiable*** and are strongly bound to the "identity" of the system (e.g. `owner/repo/.github/workflows/release.yml` for GitHub)
- **OIDC credentials are short-lived and scoped to specific audiences**
  - meaning PyPI can reject OIDC creds intended for other services

Trusted Publishing

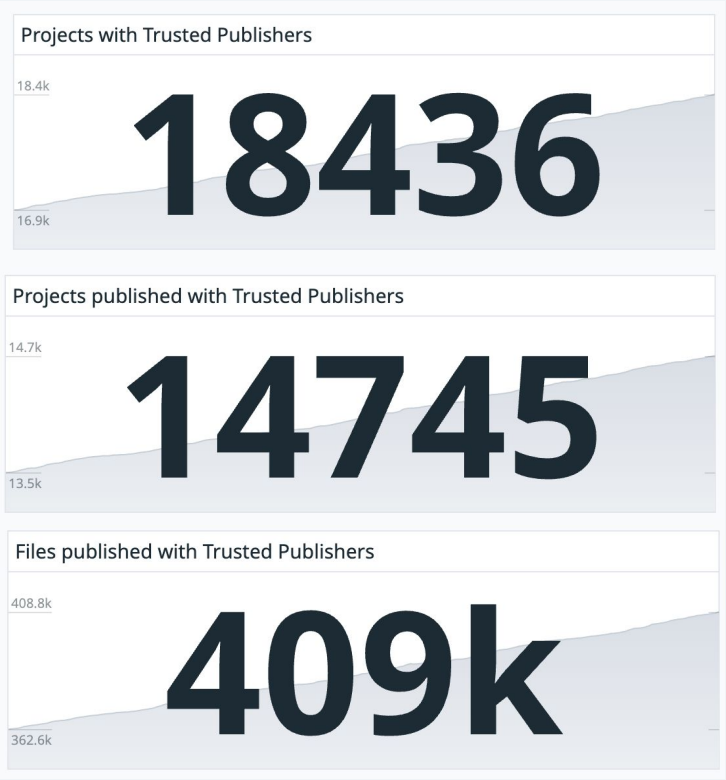
- **users configure a *trust relationship* between an OIDC provider (e.g. GitHub) and their PyPI project**
  - this trust relationship is ***public information***, so leaks are harmless
- **during publishing, OIDC provider creates an OIDC credential**
  - PyPI verifies the credential and ***exchanges*** it for a short-lived PyPI API token scoped to the project
  - API token looks identical to normal PyPI tokens, meaning that all existing tooling just works!




Trusted Publishing
User
PyPI
GitHub
"I trust release.yml on github.com/foo/bar for project `spam`"
"OIDC credentials for `release.yml @ foo/bar` can publish for `spam`"
"Trigger workflow release.yml for spam v1.2.3"
"My workflow identity is release.yml @ foo/bar"
Create OIDC credential for `release.yml @ foo/bar`
OIDC credential
"This credential matches project `spam`"
Create temporary API token for `spam`
API token
`twine upload`
User
PyPI
GitHub
www.websequencediagrams.com

success!

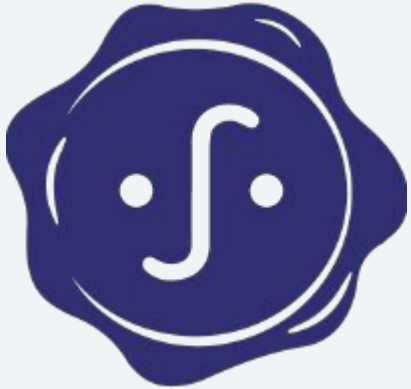
- **Trusted Publishing *makes users' lives easier*, so adoption has been rapid!**
  - especially among popular/top packages
- **successfully aligns *security goals* (misuse resistance, self-expiry) with user incentives (virtuous laziness, editing GitHub repo settings/logging into PyPI as little as possible)**



from Trusted Publishing to provenance

from Trusted Publishing to provenance

- **recap: previous attempts at digital signatures on PyPI failed because they didn't prioritize usability & default adoption**
- **Trusted Publishing gives us a new route forwards:**
  - OIDC credentials are ***proofs of identity***
  - trusted publishing ***effectively binds*** a PyPI project identity to an external machine identity (e.g. a GitHub repo)

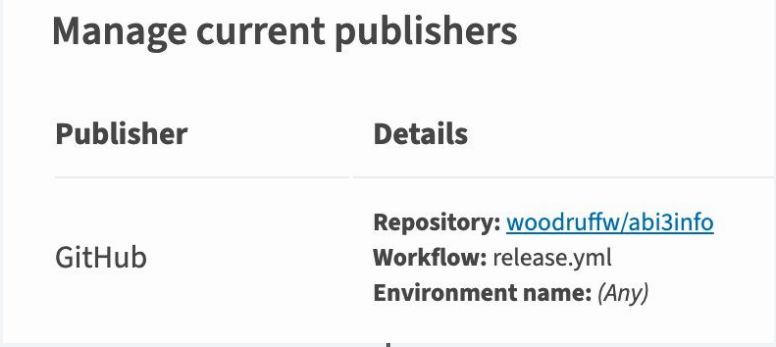


sounds like Sigstore!

from Trusted Publishing to provenance

Trusted Publishing and Sigstore are peanut butter and jelly:

- **both OIDC, so existing trusted publishing workflows can *obtain signing certificates automatically***
  - can be made the ***transparent default*** in existing publishing workflows!
- **Trusted Publisher identity closely maps to *provenance identity***
  - specifically, ***publish provenance***




```
Source Repository URI: https://github.com/woodruffw/abi3info
Source Repository Digest: c1b8afd3d6ca038dc2bb512503c619fcab489d2b
Source Repository Ref: refs/tags/v2024.10.08
Source Repository Identifier: '538293197'
Source Repository Owner URI: https://github.com/woodruffw
Source Repository Owner Identifier: '3059210'
```

PEP 740

- **PEP 740 is the standard that aligns Sigstore with Python packaging**
- **specifies:**
  - ***attestation objects*** (approx. Sigstore bundles), which contain a signature over a DSSE bundle containing the distribution name and digest
  - ***provenance objects***, which are attestations rolled up with their trusted publisher identities
  - changes to the HTML and JSON indices to serve provenance objects


```
{
  "_type": "https://in-toto.io/Statement/v1",
  "subject": [
    {
      "name": "abi3info-2024.10.8.tar.gz",
      "digest": {
        "sha256": "f81f9513cba039a5231d6523ca58b071d1d96afe430cd576bfda3e73c8b4e55f"
      }
    }
  ],
  "predicateType": "https://docs.pypi.org/attestations/publish/v1",
  "predicate": null
}
```

tying it all together

- **uploader-side implementation is complete**
  - overwhelming majority of Trusted Publisher usage comes from `pypa/gh-action-pypi-publish`, which supports PEP 740
  - users must explicitly pass `attestations: true`, but this will become the default soon!
- **index-side (PyPI) implementation is ~~almost~~ complete**
  - users can upload attestations, PyPI will verify them and publicly expose them under appropriate API endpoints and JSON/HTML indices
  - last piece: uploaded attestations are ***not yet*** reflected in the web UI itself
    - soon™
- **ecosystem-wide**
  - once the uploader default is switched, we expect similar signing volumes as total Trusted Publishing traffic (>= 400k attested uploads over the next year)
  - all without users having to lift a finger!

what's next?

the elephant in the room

- **despite all of this, something big is missing**
  - in fact, the ***most important component*** of digital attestations
- **verification!**
- **index-side verification is *~useless* unless clients also verify**
  - strictly a win in terms of transparency, but we ***can and should*** expect more!
  - that means `pip`, `uv`, etc. should support PEP 740, ***ideally by default***
- **Python packaging's devolved nature means this will be an incremental process**
  - `pip` has architectural limitations (can't depend on native code) that make integrating with `sigstore-python` challenging
    - our current workaround: developing a plugin system for `pip` that can then inject `sigstore-python`

other bits

- **PEP 740 currently specifies publish attestations and SLSA provenance**
  - but there are other kinds of useful attestations, including 3p attestations!
  - current spec leaves space for these, but they're still an open design space
- **current identity approach is tightly tied to Trusted Publishing identities**
  - but there are other things PyPI can conceptually bind with:
    - verified account emails
    - domain names
    - even self-held signing keys (SSH? minisign?)
- **reducing (and eventually eliminating) index trust**
  - Python packaging needs a suitable lockfile format that can enable TOFU with PEP 740
    - PEP 751 ☝️
  - blanket index-wide binary transparency, too boot?

we need your help!

- **do you do supply chain stuff™? consume PyPI's PEP 740 endpoints!**
  - `https://pypi.org/integrity/PROJECT/VERSION/FILENAME/provenance`
    - NB: needs `Accept: application/json`
  - let us know how we can accommodate your use case!
- **do you publish Python packages? enable TP/attestations now!**
  - you'll get them by default if you're already using a Trusted Publisher soon, but being early doesn't hurt!
- **do you consume Python packages at work? try verifying attestations!**
  - jump ***way*** ahead of the curve and try verifying them directly without installing client support, if your work setup enables this kind of introspection

thank you!

- **these slides will soon be available here:**

https://yossarian.net/publications#soss-fusion-2024

resources:

- **PEP 740: https://peps.python.org/pep-0740/**
- **Trusted Publishing docs: https://docs.pypi.org/trusted-publishers/**

contact:

- **william@trailofbits.com**
- **@yossarian@infosec.exchange**