TRAIL OF BITS

Imagining a zero-trust future for PyPI

William Woodruff, Trail of Bits

hello!

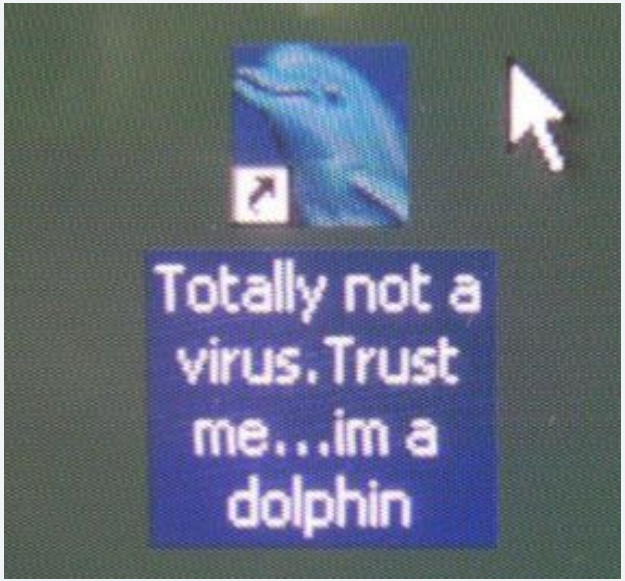
- **William Woodruff (william@trailofbits.com)**
  - open source engineering group director @ trail of bits
  - long-term OSS contributor and maintainer (Homebrew, LLVM, Python, PyPI, pip-audit, sigstore-python, etc)
- **Trail of Bits**
  - ~130 person R&D firm, headquartered in NYC
  - specialties: cryptography, compilers, program analysis research, “supply chain”, OSS security, package management, general high assurance software development

Agenda

- **Open Source and trust**
- **Quick background on Python, packaging, and PyPI**
- **What does it mean for the Python community to trust PyPI?**
- **Why should we even want a “zero-trust” PyPI?**
- **How do users currently trust PyPI?**
- **How has PyPI reduced the need for user trust?**
- **What *could* PyPI do to further reduce user trust?**
  - 1 year from now? 5 years? 10 years?

Open Source and Trust

- **OSS is everywhere**
  - Governments, airports, hospitals, FAANG, internet-connected toilets
- **An unknown, unaccountable, and indeterminate number of people are maintaining it for us**
  - Professional SWEs, hobbyists, people in friendly nation states, people in hostile nation states
- **We trust them to do it, whether we want to or not**



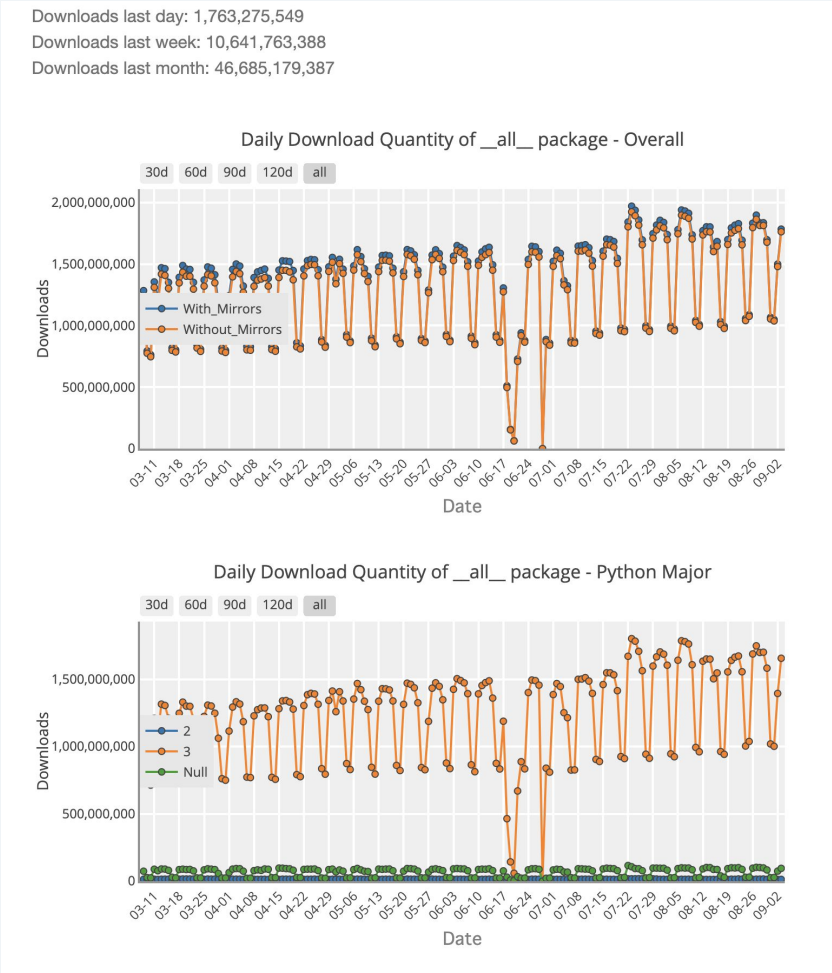
Trust in OSS is a paradox

- **To trust OSS more, we have to be *able* to trust it less!**
  - Or: the less we *have* to trust OSS, the more we *can* trust it!
- **The more we can *quantify* our trust in OSS, the better we can identify ways to trust it less!**
  - Classic example in signing: trusting each member of a keyring to sign for its own projects, or trusting the entire keyring to sign for everything

Python packaging

PyPI 

- **The Python Package Index**
  - “pie-pea-eye”
- **The default source of Python package distributions, i.e. the thing `pip install` is pulling from**
- **<u>Not</u> a dumb index/host:**
  - Source *and* binary distributions
  - User + organization management
  - Open but administered namespace



PyPI and Trust

PyPI occupies a unique position of trust within the Python community:

- 🔒 **Package integrity: already-published files don't change**
- 🔒 **AuthN/Z integrity: users/orgs can only upload to projects they control**
- 🔒 **Availability: `pip install` (almost) always works**
- 💣 **Overall package quality: "on PyPI == high quality and trustworthy"**

PyPI and Trust

An attacker who can compromise one or more of these properties poses both a security *and* a sustainability risk!

- **Security: compromising machines, inducing security fatigue**
- **Sustainability: unreliable or untrustworthy PyPI endangers the scale that Python's users (nonprofits, companies, governments) operate at**

How are we doing now?

Now

Package Integrity

Or: “can a malicious or compromised PyPI serve you malicious files?”

- ✅ **PyPI serves files over HTTPS**
- ✅ **PyPI *promises* that files don’t change after upload!**
  - ✅ Files are served with strong (= SHA-2/256 digests) for clients to check
- ✅ **Clients *can* audit PyPI for package changes via hash pinning**
  - 💀 But many don’t, due to the lack of a standard Python packaging lockfile format
- ❗ **Files don’t change, but *releases* can, resulting in surprising malleability**
  - 💀 Attacker can upload a new *more precise* file to an old release, resulting in new resolutions

Now

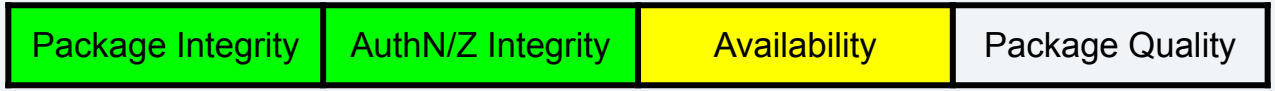
AuthN/Z Integrity

Or: “how easy is it for an attacker to take over a project, and can they be detected?”

- ✅ **PyPI has had strong MFA (TOTP + WebAuth) since 2019**
  - ✅ Mandatory since 2024!
- ✅ **PyPI requires API tokens, not passwords, for uploading**
  - ✅ 2023+: major CI/CD providers (GitHub, GitLab, etc.) can use Trusted Publishing instead!
- ✅ **PyPI has a mature policy mechanism (PEP 541) for ownership transfers**
  - 💀 …but ownership changes are not easy to audit
  - 💀 Normal looking ownership transfers are exactly how xz-utils happened

Now

Availability



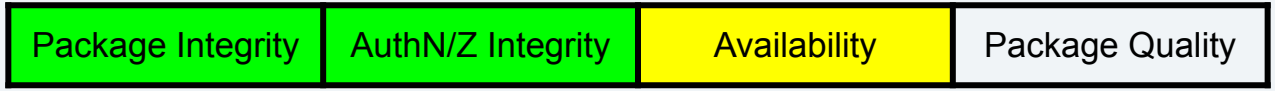
Or: “Can an attacker disrupt the overall availability and smooth operation of Python package installation?”

Multifaceted problem:

- **If PyPI has downtime/availability issues, users may temporarily or permanently switch to less trustworthy/unauditable mirrors**
- **The *perception* of a less reliable PyPI may induce worse security practices in users (e.g. updating less often)**

Now

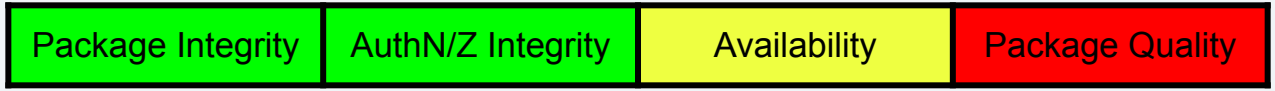
Availability



- ✅ **Demand-side protection: extensive CDN use + cache friendly design**
  - 💀 Double edged: PyPI *appears* to work effortlessly, meaning that major consumers (e.g. GitHub Actions) haven't pursued their own caches
  - 💀 PyPI's architectural viability without free CDN from Fastly is questionable
- ✅ **Supply-side protection: moderation, admin levers for disabling uploads**
  - 💀 Double edged: disabling logins/uploads is a risk when projects need to publish urgent security updates!

Now

Package Quality

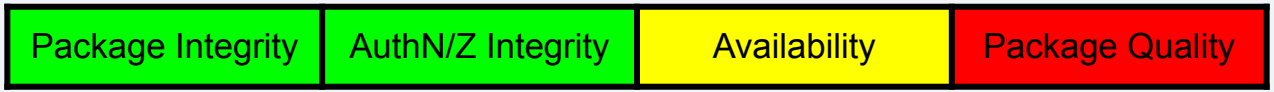


Trustworthy PyPI doesn't imply that *packages* on PyPI are trustworthy, but not all users know or believe this!

- **Driven in part by Python being a "big tent," with large independent subcommunities that don't know/care about Python packaging**
  - Sometimes framed as "everything else about Python is so magical/perfect for my $work, so I just assumed that PyPI does everything for me!"
- **Also driven by wishful security thinking**
  - "I know I shouldn't trust random code from the internet without reviewing it first, but it works out 99.9% of the time"

Now

Package Quality



Persistent misconceptions about package trustworthiness:

- 💀 **“If it’s on PyPI, it’s been reviewed or curated by the community!”**
  - Reality: PyPI is entirely uncurated and anybody can upload to it
- 💀 **“If a package is vulnerable or dangerous, PyPI will remove it!”**
  - Reality: PyPI does not remove packages with known vulnerabilities or footguns
- 💀 **“If a package is malicious, PyPI will remove it!”**
  - Reality: PyPI *tries* to remove detected malware, but makes no timeliness or completeness guarantees!

What *could* we do for a zero-trust future?

Package Integrity

- **Goal: make PyPI's *promise* of immutable files *enforceable*!**
  - Meaning users can *trust but verify* the contents of files served by PyPI!
  - Artifact transparency!
- **Artifact transparency!**
  - Go's sumdb model is likely workable within existing Python packaging standards
  - Significant blocker: no standard lockfile format
    - Hashed + pinned requirements files as a stop-gap?

Package Integrity

- **Goal: reduce (or eliminate) the incidence of surprising resolutions!**
  - Meaning `foo==1.2.3` *cannot* resolve beyond a fixed file set (no surprising new files)
- **Make releases immutable after an initial upload window!**
  - Dovetails nicely with artifact transparency
  - Semi-assumed already by non-standard tooling like Poetry
  - May break/require changes to a small number of package release workflows
    - e.g. “long-lived” releases where new Python version wheels are uploaded to the same release

AuthN/Z Integrity

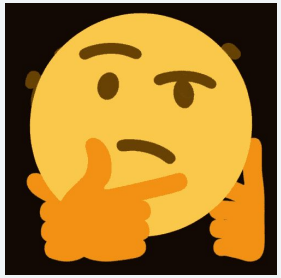
Package Integrity	AuthN/Z Integrity	Availability	Package Quality

- **PyPI known which identities are authorized to publish to which projects, but third parties can't audit that mapping!**
  - Making it public by default is thorny (e.g. identities can change, be PII, etc.)

🚧 **With PEP 740, ownership/publish relationships will become auditable for packages that use Trusted Publishing**

- 😺 Could be extended to email identities in the future!
- 😿 Still fundamentally opt-in, meaning no blanket publish transparency
  - Artifact transparency helps here?

Availability



Package Integrity	AuthN/Z Integrity	Availability	Package Quality

- 🤔 **Making this more transparent/trustworthy is challenging!**
- 🤔 **Idea 1: More explicit official CDNs/fallthrough mirrors**
  - Who will run these? Pay for them?
- 🤔 **Idea 2: Tie in with artifact transparency?**
  - In practice collapses the index/mirror distinction, as long as mirrors are both discoverable and verifiable
  - Challenge: transparency services have uptime requirements too!

Package Quality

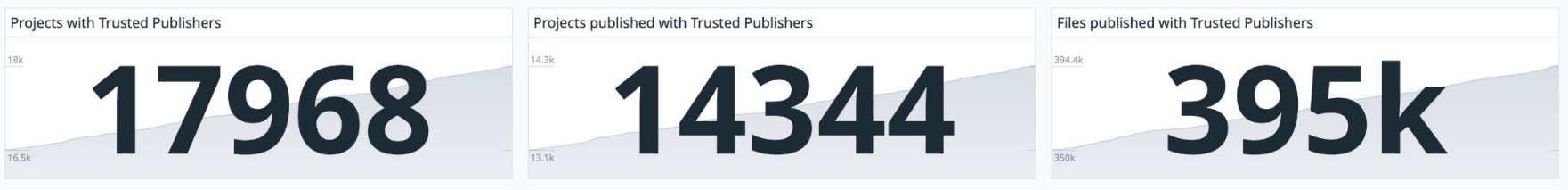
- **The goal is to require less trust in PyPI, so we can't have PyPI be the arbiter of package quality!**
  - It's also simply too big of a task
- **Transparency + community reviews offer a path forwards here**
  - Signatures over structured reviews/attestations of trust?
  - Stapled to releases/files and mirrored
  - Opens a whole new can of worms around trusted reviewers, webs of trust, etc.

What could the future look like?

The future

What's currently tractable?

- **More package attestations!**
  - PEP 740 is in its final stages of deployment
  - Once deployed fully, ~18000 packages will generating Sigstore-based attestations overnight
  - Zero-touch, no configuration required (besides Trusted Publishing)
- **Binary transparency!**
  - Empirically successful in Go, minimally invasive, requires only marginal standards considerations from Python packaging
  - Zero-touch, no configuration required (besides lockfiles)



What *might* be tractable?

- **Removing “open-ended” releases?**
  - Technically trivial, but involves changing expectations/packaging habits
- **Trustworthy mirroring?**
  - Attestations + binary transparency might enable this on a basic integrity/authenticity level, but availability is equally important for a strong mirror ecosystem
- **Namespacing?**
  - Either “hard” or “soft”
  - Widely desired by the community; mixed consensus on technical approach
  - Doesn’t reduce index trust, but *does* simplify package <-> identity tracking

What *isn't* (yet) tractable?

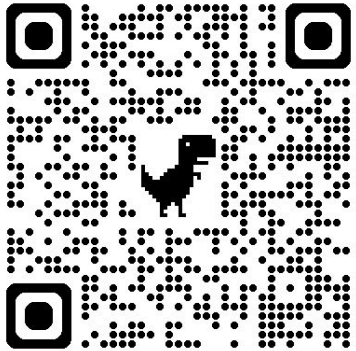
- **Removing the user's need to make informed installation decisions**
  - "You still need to not `pip install malware`"
  - Current architectures for doing this look too much like PGP webs of trust, with the same bottlenecks/pain points (e.g. transitive trust)
- **Turn-key log monitoring + witnessing**
  - For signing identities (= publishers) and artifact names (= package distributions)
  - Determined parties can already do this, but they're a (tiny) minority

Call to action

- **Lots of people working on package security, lots of people working on transparency, not very many people working on the intersection thereof!**
  - Packaging ecosystems (besides Go) have a lot to glean from CT, Go sumdb, etc.
- **Building this stuff is “easy” compared to actually getting it into users’ hands**
  - Much less making it the default
  - Trusted Publishing is the rare exception to this, because it made users lives *easier*
  - We need help thinking of creative solutions to this!

Thank you!

Contact 👉



Slides 👉