TRAIL OF BITS

What does it look like to code-sign for an entire packaging ecosystem?

William Woodruff, Trail of Bits

What ~~does~~ will it look like to code-sign for an entire packaging ecosystem?

What ~~does~~ will it look like to code-sign for ~~an entire packaging ecosystem~~ PyPI?

Hello!

- **William Woodruff (william@trailofbits.com)**
  - open source group engineering director @ trail of bits
  - long-term OSS contributor (Homebrew, LLVM, PyPI) and maintainer (pip-audit, sigstore-python)
  - @yossarian@infosec.exchange
- **Trail of Bits**
  - ~150 person cybersecurity engineering and auditing consultancy
  - specialities: cryptography, compilers, program analysis research, "supply chain", OSS package management, general high assurance software development

TRAIL OF BITS

Why PyPI?

- **I am not a PyPI maintainer, and these are not the opinions of PyPI, PyPA, PSF, etc!**
  - Just a contributor with lots of Opinions
  - Nothing in this presentation is “blessed” as the current approach
- **PyPI is a great motivating example:**
  - Official index for one of the world’s most popular languages + largest package communities (~500K projects, ~750K registered users, ~20B downloads/month)
  - Storied history with code-signing attempts (more on this soon)
  - Lots of policies driven by development and community feedback; lots of “scar tissue”
  - Maintainers care deeply about security
    - …and are *particularly* interested in getting code-signing right
  - Selfishly: I know it well (comfortable frame of reference!)

Problem space

Problem Statement

“When I `pip install sampleproject` from PyPI, I am confident that the package I install is the one created by its maintainer”

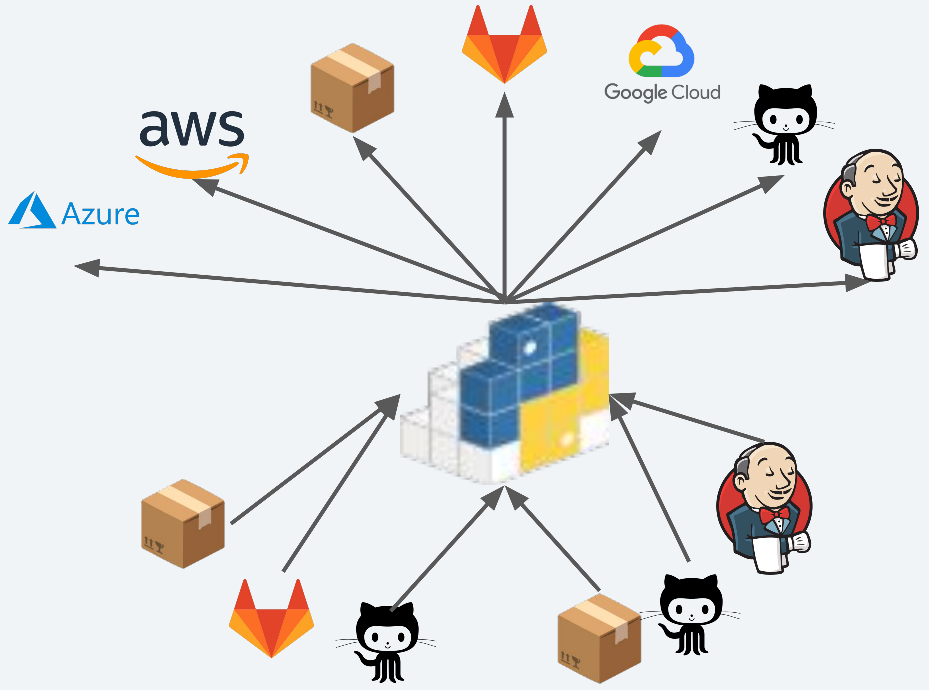
Problem Statement

"When I `pip install` `sampleproject` from PyPI, I am confident that the package I install is the one created by its maintainer"

Problem space

Actors

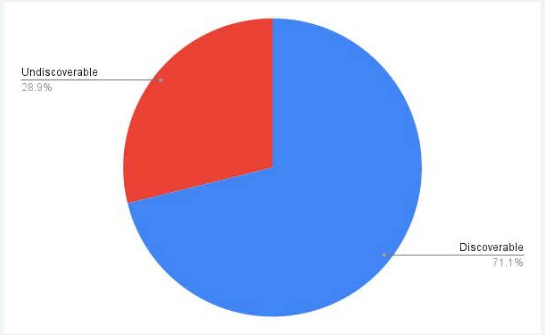
- Upstreams (publishers)
- Package index (PyPI)
- Package installer (`pip`)
- Downstreams (developers, CI/CD, production builders)



Observation: Code-signing on OSS package indices is multi-agent distributed trust problem!

Problem space

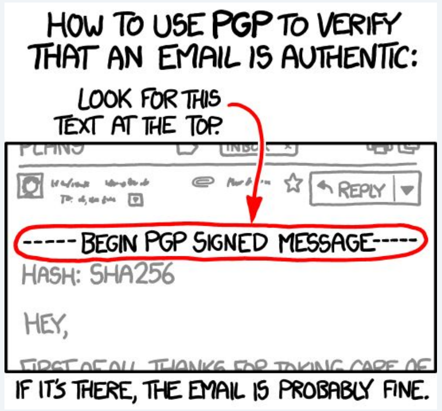
Actor: Upstreams (publishers)



- **Everyone from Fortune 500 to XKCD Nebraska Guy**
- **Traditional solution: make individual publishers maintain signing keys**
  - Examples: self-signed X.509 certs in RubyGems, PGP in many ecosystems
  - This has not gone well in any "open" signing ecosystem
- **Problem: ~nobody maintains/distributes signing keys correctly**
  - Publishers **should not have to understand** keyrings, revocations, etc. to sign for their packagers!
  - For ecosystems like Python: ease of getting started is a big draw
- **Problem: bad tooling, insufficiently opinionated standards**
  - PGP is obscenely complicated, GPG is hard to use correctly
  - Package indices don't bother to ensure minimal key strengths or similar policies because of this
- **Problem: identity mapping and distribution**
  - No intrinsic mapping between human-readable identities and public keys
  - No obvious way to distribute even an informal version of that mapping

Actor: Package Index

- **Stereotypically volunteer-run, with minimal resources**
  - Conclusion: significant operational constraints!
- **The central hub of informal trust in OSS packaging**
  - The package index *is* the root of trust for the package name ←→ contents mapping
- **Traditional solution: naively accept and broadcast signature-shaped inputs**
  - Can't be verified, because the package index doesn't know which key is right
- **Problem: allowing users to upload keys means index becomes a keyserver (with expiries, revocations, etc.)**
  - We want *fewer* operational burdens, not more!

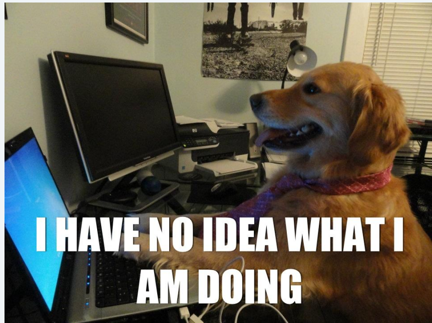

HOW TO USE PGP TO VERIFY THAT AN EMAIL IS AUTHENTIC:
LOOK FOR THIS TEXT AT THE TOP.
REPLY
-----BEGIN PGP SIGNED MESSAGE-----
HASH: SHA256
HEY,
IF IT'S THERE, THE EMAIL IS PROBABLY FINE.

Actor: Package Installer

- **Also volunteer run, similar operational constraints**
  - Not always run or maintained by the same people who maintain the index!
- **Cannot break, *ever***
- **Traditional solution: shell out to `gpg --verify`**
  - Punts all identity mapping, management, etc. to end user
  - No `gpg` binary? No codesigning!
- **Problem: being at the bottom of the dep tree means that the installer is subject to unusual build and runtime constraints**
  - For `pip`: all deps must be pure Python, must run equally well everywhere Python runs, must be vendorable into `pip` itself, etc.
  - Conclusion: signature verification needs to be pure Python (or similar), or otherwise needs to get around this constraint

Actor: Downstreams

- **Less sophisticated than publishers, 1000x larger population**
  - The "silent majority" of every packaging ecosystem
  - Not interested in operational matters: all they care is that `pip install` continues to work
- **Traditional solution: try and make downstreams care about signatures**
  - Spoiler: they don't care (and again: **should not have to care** about revocations, etc.)
  - They don't know (and don't care) about SLSA, Sigstore, TUF, in-toto, etc.
- **Problem: downstreams don't/won't maintain their own identity mappings and trusted sets**
  - Even when willing, doing so is operationally painful (manual for each trusted identity)
  - Failure mode is "everything still works," so there's no point



Bringing it all together

Our known constraints:

- ***Packagers*** **can't be expected to maintain long-lived signing keys**
- ***Package Indices*** **are operationally constrained and already informally trusted**
- ***Package Installers*** **are operationally constrained and have non-trivial runtime/vendoring constraints**
- ***Consumers*** **can't be expected to maintain keyrings or identity mappings**

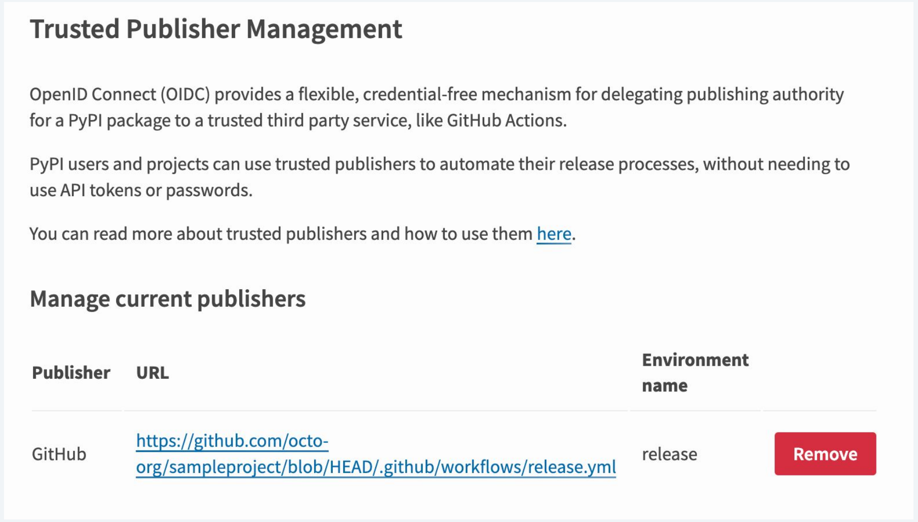
A *successful* scheme needs to address (or *sidestep*) all of these!

Solution space

Solution Space

Bootstrapping identity with Trusted Publishing

- **Two problems:**
  - Downstreams don't want to/can't maintain signing keys
  - PyPI doesn't want to be a PKI or have PKI problems
- **Solution: Sidestep both with *Trusted Publishing***
  - OIDC based authentication to PyPI from CI/CD providers (e.g. GitHub Actions)
  - Deployed ~8 months ago, used by 5% of critical projects (but disproportionate downloads)



Trusted Publisher Management

OpenID Connect (OIDC) provides a flexible, credential-free mechanism for delegating publishing authority for a PyPI package to a trusted third party service, like GitHub Actions.

PyPI users and projects can use trusted publishers to automate their release processes, without needing to use API tokens or passwords.

You can read more about trusted publishers and how to use them here.

Manage current publishers

Publisher	URL	Environment name	
GitHub	https://github.com/octo-org/sampleproject/blob/HEAD/.github/workflows/release.yml	release	Remove

Bootstrapping identity with Trusted Publishing

- **Trusted Publishing uses the same OIDC credential as Sigstore!**
  - CI workflows can sign using the same machine identity, no changes or interaction required
  - Gets us SLSA Level 2 for free, thanks to CI claims in Sigstore certificates
  - Minimal operational burden, since Sigstore does “keyless” signing
- **PyPI can accept Sigstore-based signatures and cross-check them against the project’s Trusted Publishers before redistributing them**
  - Solves the “garbage in, garbage out” problem with PGP signatures

Takeaway: we can make identity management *and* signing *automatic* for projects, without increasing the amount of trust placed in PyPI!

Trust distribution and verification

- **Three different kinds of verification we can do:**
  - Upstream side: “PyPI is hosting an authentic distribution of my package”
  - Index side: “We only accept TP’d packages with signatures we can verify”
  - Downstream side: “I only install packages from identities I trust”
- **Downstream is still the hardest!**
  - `pip` can’t vendor sigstore-python at the moment (because of native/Rust deps)
  - Underlying identity trust problem is not solved in the average (non-corporate) case
  - Medium-term: TOFU with locking
    - PyPI delivers initial trusted identities, updates that change identities can be reviewed on a manual or policy basis

Outstanding problems

Outstanding problems

Reducing trust in PyPI

- **Using Trusted Publishing to bootstrap codesigning means that we trust PyPI to distribute signatures and identities**
  - PyPI is already trusted absolutely so this doesn't *broaden* its scope, but it still isn't ideal!
- **Long term: we need to reduce trust in PyPI**
  - Ideally in ways that keep mirroring simple/increase confidence in mirrors as well
- **Divide and conquer: we want to reduce trust in:**
  - PyPI itself (the Warehouse backend)
  - Auxiliary services (CDNs, caches)

Outstanding problems

Reducing trust in PyPI

- **Index-level Transparency**
  - PyPI could *countersign* for uploads + commit countersigs to a transparency log
  - Compare: NPM's "publish attestations," but binding over TP sig as well
- **Complements "free" transparency from Sigstore-based signatures**
  - Allows timerange validation: "PyPI must not have updated the index more than X minutes after the the package was published"
- **Problem: What claims to expose to exist in log monitoring/auditing?**
- **Problem: Transparency has a "mushy" security model**
  - Half protective, half remediative ("trust but verify")
  - Hard to communicate/explain to users!

Reducing trust in PyPI

- **TUF**
  - Many options: “full” RUF, RSTUF, TUF-on-CI, etc.
  - Hierarchy of keys (from offline to online) ultimately signing for each index endpoint
- **Makes secure mirroring easier + protects against a compromised CDN**
- **Problem: Makes PyPI assume the role of a PKI**
  - We still don’t want this if we can avoid it!
  - One option: sidestep this by piggybacking on the Sigstore TUF repo (like NPM does)
- **Problem: Operational complexity/long-term maintenance risk**
  - Features added to PyPI have to survive 5+ years (possibly 15+)
  - Long-term key rotation/signing ceremonies are operationally heavy & brittle, especially for unpaid/community-run services!

Conclusions

- **Package indices are not closed ecosystems**
  - Code-signing has to be a “carrot,” not a “stick”
  - Specifically: >50% of users will not adopt an optional technology if it makes them do more work, not less
- **Trusted Publishing lets us eventually cover the *majority* of package downloads without requiring users to change anything!**
  - We will **never** get to 100%, per above
- **Eliminating trust in PyPI is much harder than just delivering signatures**
  - This is where novel research is needed!
  - Research **must** be informed by operational constraints: unpaid maintainers, limited attention, 15+ year feature cycles, dis-interested upstreams and downstreams, package installer constraints, etc. etc.

Thank you!

Slides will be available here:

https://yossarian.net/publications#acmscored-2023

https://github.com/trailofbits/publications

Resources:

- A Roadmap for Trusted Publisher Signing on PyPI — Early, pre-standard design roadmap
- Publishing to PyPI with a Trusted Publisher — Official PyPI docs on Trusted Publishing

Contact:

- Email: william@trailofbits.com
- Mastodon: @yossarian@infosec.exchange