TRAIL OF BITS

Sigstore for Python Packaging: Next Steps for Adoption

William Woodruff
Trail of Bits

hello

- **william woodruff (@8x5clPW2)**
  - senior security engineer @ Trail of Bits
  - Homebrew, pip-audit, sigstore-python maintainer, contributor to many things
- **Trail of Bits**
  - ~~small~~ “midsized” security and research consultancy (~110 people, ~80% remote)
  - areas: program analysis research, security audits, open source cryptography and security engineering
  - we’re hiring! join us!

agenda

- **python packaging: the 30,000 foot view**
  - or: “my package index is (almost) old enough to drink”
- **pre-existing security practices in python packaging**
  - ...and how sigstore can replace nearly all of them
- **where sigstore is currently being used in python**
  - ...and how **you** can use it
- **the bright horizon**
  - sigstore + TUF
    - ...in PyPI and pip and your CI
    - ...for assured verification policies

a modest proposal

- **everybody in this room cares about {software, supply chain} security**
  - that means, ultimately, that we all care about **adoption**
- **the hard truth: most engineers do not, *nor is it their job to***
  - where we see new security practices, they see surprise job responsibilities and obstacles
- **therefore: it is our job to make security *as easy as possible***
  - if we don't do this, engineers will not use what we give them
  - if they're *made* to us it, they'll resent it and misuse it
- **this applies doubly for open source (aka free labor)**
- **pictured: we are all gildrig**

python packaging: the 30,000 foot view

- **2000: distutils is added to Python 1.6.1**
- **2001: PEP 241 standardizes the packaging metadata (“Metadata 1.0”)**
- **2002: PEP 301 establishes the first central package index**
- **2003: The first service known as PyPI is publicly available**
  - PEP 314: Metadata 1.1
- **2004: `easy_install` is announced**
  - Python eggs as an sdist alternative
- **2005: PEP 345 standardizes Metadata 1.2**
- **2008: `pip` (neé `pyinstall`) is released**

python packaging: the 30,000 foot view

“the modern era”

- **2012: PEP 427 standardizes Python Wheels**
- **2015: PEP 503 describes the extant PyPI repository API**
- **2017: PEP 566 standardizes Metadata 2.1**
- **2018: Warehouse (the modern PyPI backend) is deployed**
- **2020: PEP 458 (signed repository metadata) is accepted**
- **2021: 354 of the top 360 Python packages are distributed as wheels**
  - pip 21.0 enables a modern dependency resolver
- **2022: you are here**

python package security: prior art

quick reminder: we care about authority, integrity, authenticity

- **authority:** users/identities can only publish packages that they own
- **integrity: packages delivered to users are not modified in transit or on the index**
- **authenticity: a package "is what it says it is"**
  - not just unmodified, but ***provably*** *created by a trusted entity*

cryptosystems compose these properties, e.g. giving us transitive integrity through authenticity!

python package security: authority

- **PyPI provides user accounts with traditional username/password auth**
  - users have roles on projects: either “owner” (full admin) or “maintainer” (release only)
  - users also have one or more emails on their account; **not** tied to email metadata in packages
- **historically, username + password was the only way to auth to PyPI**
  - OpenID and Google Auth were supported at one point, but removed due to disuse
  - including for package upload with twine! lots of user credentials stuffed into CI pipelines!
- **2019: PyPI grows two-factor authentication (2FA) and API tokens**
  - 2FA: users can use TOTP (6 digit codes) or WebAuthn (security keys) to secure their accounts
  - API tokens: users can use a scoped credential instead of their username and password!
- **status quo: on par with or better than other major packaging ecosystems!** 🎉

python package security: integrity

- **PyPI was historically an HTTP index, referencing files stored on other HTTP or FTP hosts**
  - no transport integrity! no host integrity, for either the index or file servers!
  - HTTPS was added at some point, providing transport integrity
  - full hosting of distributions eliminated poor hosting practices by packagers
- **pip 8.0 (2016) added a hash-checking mode**
  - fetched distributions are checked against locally-specified hashes (SHA256 preferred)
  - all or nothing: if a single requirement has hashes, all must have hashes
  - hashes have poor UX; users gravitate to 3p tools like pip-compile to maintain hashed deps
- **wheel distributions contain per-file hashes in the RECORD file**
- **status quo: close to other package managers, but harder to use 😐**

```
1  pyrage-1.0.1.dist-info/METADATA,sha256=bii8anpXwJ7U1hW4DFgI2RfoeRZHMJ56sm8cBvyOES8,2346
2  pyrage-1.0.1.dist-info/WHEEL,sha256=I6HWtVqES-SaAXynnsSTpnVdA8DyHckZCetaVGOn-0w,145
3  pyrage/__init__.py,sha256=pNWoDP3ueNZGAK41QghutnNbFgvyYeGPwqSUw2weqho,107
4  pyrage/pyrage.abi3.so,sha256=fZed01paF4ELR5cD1SZGkm5wCXl8t2B4okXJgOW1AjM,2398716
5  pyrage-1.0.1.dist-info/RECORD,,
```

python package security: authenticity

- **lol**
- **PyPI has *optionally* supported PGP signatures on distributions for years**
  - Packagers are expected to upload an adjacent `.asc` file for each dist
  - Why do you trust that signature? Anybody can sign for a package!
    - Something something web of trust?
- **wheels have *optional* support for JWS-JS and PKCS#7 (S/MIME) signatures**
  - Same problems as PGP: why would you trust these?
  - Virtually no adoption, since support is purely optional
    - `pip` simply doesn't verify these
- **status quo: authenticity is vestigial and impractical** 😭

> A wheel installer is not required to understand digital signatures but MUST verify the hashes in RECORD against the extracted file contents. When the installer checks file hashes against RECORD, a separate signature checker only needs to establish that RECORD matches the signature.

sigstore!
now and forwards!

what does sigstore do for python packaging?

- **sigstore solves codesigning's UX problems while preserving its best properties:**
  - **key management**: packagers no longer need to securely store long-lived signing keys; verifiers no longer need to perform keyring maintenance
  - **(in)agility**: sigstore picks the right cryptographic primitives; verifiers no longer need to worry about verifying against weak keys or broken schemes
  - **identity**: sigstore signatures can be rooted to a public identity (like email address or GitHub handle); verifiers no longer need to muck about in the Web of Trust to determine whether they should trust a signature
- **sigstore will also *strengthen* pre-existing authority and integrity properties in python packaging!**
  - authority: making PyPI its own OIDC IdP is on the development roadmap
  - integrity: sigstore's authenticity guarantees provide transitive integrity properties

sigstore: where we are now

- **we have a reference python-implementation: [sigstore-python]**
- **our reference implementation is being used to sign for CPython itself:**

> Starting with the Python 3.7.14, Python 3.8.14, Python 3.9.14, and Python 3.10.7 releases, CPython release artifacts are additionally signed with Sigstore (in addition to existing GPG signatures).

- **we have a straightforward CLI (it really is this simple):**
  - we still need more flags/options for configuring different verification policies!

sigstore: where we are now

- **sigstore-python supports “ambient” OIDC credentials**
  - GitHub Actions, Google Cloud Build, others soon (CircleCI and GitLab)
  - we have sigstore/gh-action-sigstore-python for automatically signing with GitHub’s OIDC!
  - we can publish signing artifacts (signature, certificate, etc.) to GitHub Releases automatically

codesign.yml

```yaml
jobs:
  codesign:
    runs-on: ubuntu-latest
    steps:
      - uses: sigstore/gh-action-sigstore-python@v0.0.9
        with:
          inputs: ./some/target/to/sign
```

sigstore: where we want to be

- **we can sign and verify things with sigstore-python; we want those signing artifacts to be published on PyPI alongside distributions**
  - ...and, eventually, cross-checked via inclusion in a TUF repository for PEP 458
- **to support this, PyPI needs to let us upload a `{dist}.sig` and `{dist}.crt` for each `{dist}`**
  - ...or a single sigstore bundle, once those are stabilized
  - PEP 694 (Upload API 2.0) will enable this by giving us the ability to attach the signature and certificate as metadata before the actual file upload
- **once on PyPI, we can deliver verification materials to all users**
  - ...once PEP 503 is superseded by PEP 691, or augmented with another metadata tag

sigstore: where we want to be

- **server-side isn't enough: we want users to also benefit from verification on the client side!**
  - ideally: `` `pip install …` `` will do sigstore signature verification under the hood
    - challenge: `pip` needs to run everywhere, so sigstore-python needs to be pure Python
  - users should not have to be aware that anything has changed!
  - *if* they want to be aware, they can opt into more powerful verification functionality (like custom policies for restricting packages to only a set of valid identities)
- **bigger picture: sigstore fits into the larger signing/"high assurance" constellation for *all* package ecosystems**
  - we want our efforts in Python to be a litmus test for other ecosystems evaluating sigstore

sigstore: what we need

- **UX**
  - it's one thing to verify signatures, another to *trust identities*
  - sigstore is **conceptually** better than blind verification; use should be **practically** better
  - PyPI and individual packages have lots of metadata; maybe some easy wins there
    - package maintainer emails ↔ email identities in sigstore?
    - PyPI is getting OIDC publishing support; OIDC provider registration ↔ URL identities in sigstore?
    - longer term: allow users to configure machine-readable verification policies for their projects; commit those policies to TUF with the rest of the package index!
- **threat modeling**
  - sigstore is conceptually complex
    - users should not have to understand RFC 6962 or 5280 to use it securely
    - users should not have to understand PKI, ephemeral keys, etc. to understand how they benefit from sigstore

thank you!
ask me questions!

contact:
william@trailofbits.com
@8x5clPW2