TRAIL OF BITS

ergonomic codesigning for the Python ecosystem

William Woodruff

hello

- **William Woodruff**
  - 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**
  - ~150 person cybersecurity auditing and engineering consultancy
  - specialities: cryptography, compilers, program analysis, “supply chain”, general high assurance software development

agenda

This talk

- **“Python is everywhere, with everything that entails”**
  - The “why does everything have 5000 dependencies” part of the talk
- **codesigning (for Python packaging): history and past attempts**
  - PGP and other things I’m trying to forget
- **Sigstore as an emerging standard for codesigning**
  - …and how it resolves problems of identity and key management
- **Sigstore’s now & future role in Python packaging, and what it means for ✨you✨**
  - Where we’re currently at and where we’re planning to go

Python is everywhere

- **this part is obvious from our vantage point at PyCon**
- **…but the implications bear consideration:**
  - ~everyone on earth (to a first approximation) interacts with software written in Python
  - ~billions of machines run Python, and that Python is critical to their intended operation
- **Python wins the decision lottery at companies & universities, in open source communities & activist groups, with statisticians and journalists, etc., etc..**

… because of OSS + universal packaging

- **Python's OSS community is massive, and engages heavily with the packaging ecosystem**
- **(nearly) everything is a single `pip install` away, which has consequences:**
  - Python packaging tools are used extensively by non-programmers;
  - Packaging tool (ab)use is heavily influenced by extensive public use, third party documentation, and Python's limited ability to restrict private APIs;

meaning: Python packaging is heavily constrained by existing use; changes/additions in functionality must not break existing uses*

this includes security improvements!

let's talk about "supply chain security"

- **new ~~buzzword~~ industry term for "don't run strangers' code"**
- **at least two parses:**
  - "don't ***run*** strangers' code": know ***what*** code you're using before you actually run it
    - static and/or dynamic analyses to detect malicious and/or exploitable changes
    - lots of companies doing work (and selling products) in this space
  - "don't run ***strangers'*** code": know ***whose*** code you're running
    - Python development involves multiple implicitly trusted third parties: PyPI, each package's author on PyPI, your ISP
    - Each implicitly trusted party should be:
      - *Made explicit*: ***who*** are they?
      - *Scoped and enumerated:* ***what*** can they do without detection?
- **we're going to use codesigning to address that second parse!**

codesigning: a quick overview

- **codesigning = digital signatures for code (rather than files, docs, etc.)**
- ***digital signatures* are cryptographic objects that provide *strong cryptographic proof* of*:***
  - **integrity**: the input (= code) being signed for **has not been altered** since signing
  - **authenticity**: the input (= code) being signed for **was signed by a specific identity**
- **"identity" in digital signature schemes is defined by *key possession*:**
  - private key holder is the *signer*; they can create signatures for anything they have access to
  - public key holder is the *verifier*; they can verify any signature that they have the corresponding input for*

codesigning for packaging ecosystems

- **verifying digital signatures on packages means we can erase one of our implicit trust relationships**
  - if Bob trusts Alice (= Alice's private key) to sign for package `foo`, PyPI **cannot** deliver a modified `foo` without Bob noticing!
- **this relationship is tricky:**
  - Bob ***must*** know Alice's public key ***ahead of time***: if PyPI is trusted to deliver the public key, then a compromised PyPI can deliver a modified `foo` signed with a different private key!
  - Bob ***must*** know ***what*** Alice is trusted to sign for: trusting Alice for `foo` should not imply trusting Alice for `bar` if she doesn't rightfully control/own `bar`
    - This might not matter when `foo` and `bar` are adjacent dependencies, but it does matter if Bob does `pip install bar; sudo python -m bar`!

codesigning for packaging ecosystems

this is where ecosystems like PGP *fail miserably*:

verifiers **are expected to maintain *keyrings*, which configure trust in a set of identities; this is hard to do correctly!**

signers **must navigate gpg's terrible CLI to create signing keys; strong key generation is not intuitive and frequently not the default (due to PGP's age)**

signers must additionally store their key material securely, must prepare for key expiry and recovations, etc.

PGP attempted to solve identity trust with the "strong set," which was fully removed from GPG after extensive keyserver abuse

codesigning for Python packaging: status quo

- **codesigning support in Python is extremely fragmented:**
  - **PyPI, and `twine` (kind of) support PGP signatures, and some packages (<10%) use them**
    - **PyPI will let you upload a `.asc` signature for each release distribution; `twine` supports this**
    - **…but `pip` will (reasonably!) not download or verify these signatures**
      - **Long tail of extremely old (= weak keys), and there's no way to verify the signing key anyways**
  - **wheels (PEP 427) support embedded JOSE (JWS) signatures**
    - **virtually unused (only a small number of wheels on PyPI); not supported by `pip`**
    - **same identity issues as PGP; inherits many of JWT's mistakes ("alg: none")**
  - **…and also S/MIME (PKCS#7/CMS) signatures?**
    - **"RECORD.p7s is allowed as a *courtesy* to anyone who would prefer to use S/MIME signatures to secure their wheel files."**
    - **I can't find any (public) evidence that anyone has ever used these on PyPI**
    - **Same identity issues as PGP; brings the evil that is PKCS#7 to Python packaging**

can we do better?

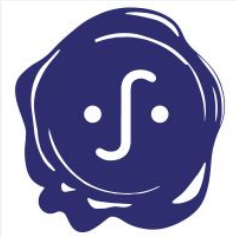
let's re-evaluate assumptions made in PGP and JOSE/JWS:

- **assumption: both humans and computers verify through public keys**
  - **reality**: computers verify keys, but humans only care about ***identities***
  - "I don't care *which* key identifies Alice, as long as I can be convinced that it's Alice's key"
- **assumption: humans are good at maintaining long-term secrets**
  - this has literally never been true
  - **reality**: humans lose PGP keys, 2FA tokens, passwords ***all the time***
- **assumption: users understand basic cryptography**
  - **reality**: they might, but they shouldn't have to!
  - similarly: users should not have to understand PKI (revocation, etc.) to sign/verify safely

can we do better? Yes, with Sigstore!

Sigstore is…

- **…a PKI* ecosystem (X.509 CA, RFC 6962 CT)**
  - "Let's Encrypt but for code signing"
- **…a framework for *binding keys to public identities***
  - Leveraging OpenID Connect to communicate with well-known identity providers (IdPs) like Google, GitHub, etc.
- **…a client ecosystem**
  - Per-language/ecosystem bindings, like for Python!
- **…a Linux Foundation project**
  - Developed and maintained by members of the OSS community



solving identity and key management with Sigstore

Sigstore solves the identity binding and key management issues by binding *ephemeral signing keys* to *OpenID Connect identities*

- **Alice generates a local signing *keypair*** (K_{pub}, K_{priv})
- **Alice performs an OIDC flow, receiving an *OIDC*** $Token$ **attesting that she controls** `alice@example.com`
- **Alice submits** $(K_{pub}, Token)$ **to Sigstore's CA and receives a Certificate** C_{pub} ***binding the public key to*** $alice@example.com$
- **Alice *signs*** $Input$ **using** K_{priv}**, producing** Sig_{Input}
- **Alice *publishes*** $Input$**,** C_{pub} **and** Sig_{Input} **and *destroys*** $\mathbf{K_{priv}}$

demo time!

That was a lot of internal details, including things that *need* to be abstracted away to provide a usable codesigning ecosystem.

Let's see if Sigstore (via `sigstore-python`) succeeds at providing that abstraction!

“sunlight is the best disinfectant”

- **What’s the new trust model?**
  - Sigstore’s CA is trusted, since Alice has no fixed key: Bob must trust that only the real Alice can convince the CA to issue a certificate for `alice@example.com`
  - Comparable to Web PKI: users must trust that only the real Google can convince a CA to issue a certificate for `google.com`
- **We *reduce* trust in Sigstore’s CA by *requiring transparency*:**
  - Bob ***refuses*** to trust Alice’s certificate ***unless*** it appears in appeared in a publicly auditable CT log at roughly the same time it was issued
  - An attacker targeting Bob might still be able to fool Sigstore’s CA, but not without ***losing their stealth***: CT makes their attack against Bob ***globally visible and publicly detectable***

This the same technique the Web’s PKI relies on!

what does any of this mean ***for Python***?

Python packagers and users should not have to be experts in cryptographic primitive selection or secure key storage to benefit from codesigning.

- **Sigstore *solves* problems of identity and key management that previous codesigning attempts in Python (PGP, JWS) *fundamentally can't*.**
- **Sigstore is *misuse resistant* and *reduces unnecessary trust* in the package index (including its admins, ISP, CDNs, etc.).**

Sigstore for Python: where we are

- **we have a mature Sigstore impl. for Python (`sigstore-python`) that can be used to sign anything (including Python distributions)**
- **we have a GitHub Action (`gh-action-sigstore-python`) that makes signing *completely painless* on CI**
- **CPython itself is signing its official releases with Sigstore, using each release maintainers' email identity:**

https://www.python.org/download/sigstore/

Sigstore for Python: where we want to be

- **PyPI should allow uploading of Sigstore bundles next to their associated release files**
  - ...including with a pairing/TOFU* scheme against the package's pre-existing unauthenticated metadata
  - Related PEPs: 694, 691, an unwritten one for the TOFU scheme
- **`pip` should (optionally, at first) verify Sigstore bundles during download/install**
  - 2FA for critical projects rollout as a template: mandate signatures for the top `N` projects

Sigstore for Python: how you can help

- **packagers:**
  - give our Sigstore client a try: `pip install sigstore && sigstore --help`
  - tell us about your usability/privacy/etc. concerns!
- **(Python) cryptographers and security engineers:**
  - Sigstore is not a panacea: it punts policy management to the client
    - help us come up with a reasonable trusted metadata scheme for PyPI!
  - Sigstore is still maturing as an ecosystem, and needs help:
    - PQ-ready signatures?
    - privacy enhancements to Sigstore (blinded identities, reducing online CT lookups)?

thank you!

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

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

- **resources:**
  - sigstore.dev: the official Sigstore project website
  - sigstore/sigstore-python: the official Sigstore client for Python
  - "We sign code now": our blog post on Sigstore's server and client internals
- **Contact:**
  - william@trailofbits.com
  - @yossarian@infosec.exchange