TRAIL OF BITS

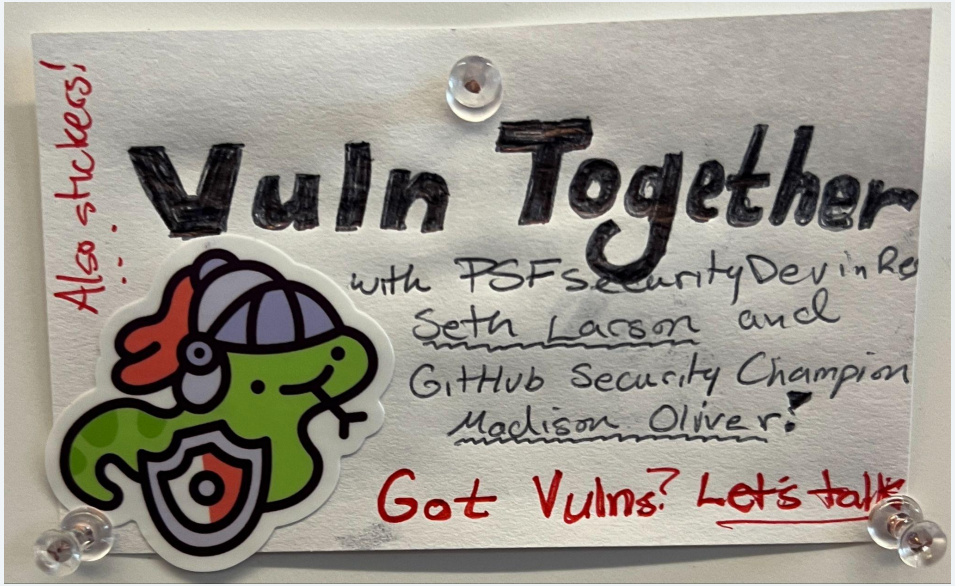
Building a Rusty path validation library for PyCA Cryptography

William Woodruff, Trail of Bits

psa: open space

Open Space in Room 320, 2PM

- How to talk with vulnerability reporters?
- What do the acronyms mean?
- Where does vulnerability data come from?
- How can I set my project up for success?
- I'm stressed (about vulns) help!
- (or anything else you want to talk about!)



hello!

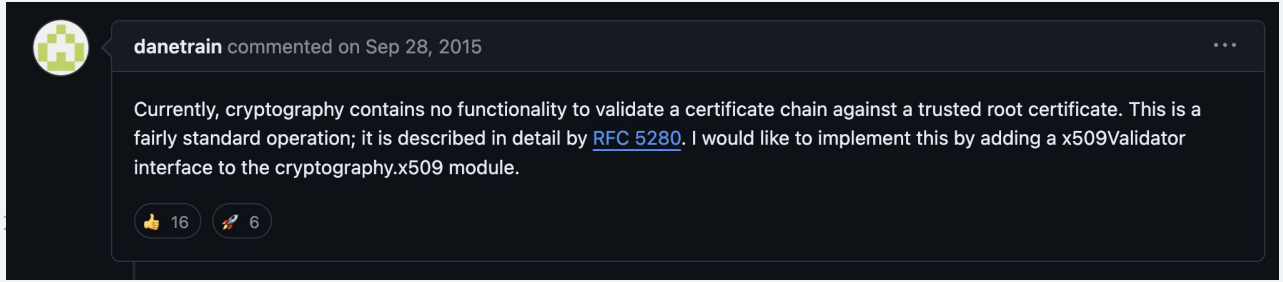
- **William Woodruff (william@trailofbits.com)**
  - open source group engineering director @ trail of bits
  - long-term OSS contributor (Homebrew, LLVM, Python) and maintainer (pip-audit, sigstore-python)
  - @yossarian@infosec.exchange
- **Trail of Bits**
  - ~130 person R&D firm, NYC based
  - specialities: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

agenda / questions

- **“why does this matter?”**
  - introduction to X.509 and path validation
  - “the scary underworld beneath your REST stack”
- **“why another implementation?”**
  - pre-existing implementations
  - design principles and constraints
  - our implementation, testing approach, etc.
- **shipping it**
  - improving other implementations through our work
  - lessons learned
  - future ideas and work

thank-yous

- **this work was funded by the Sovereign Tech Fund**
  - themselves funded by 🇩🇪 via SPRIN-D
- **PyCA maintainers (Paul Kehrer and Alex Gaynor) defined the bounds of our approach, answered questions, and reviewed (and fixed) our changes throughout ~9 months of development**
  - x509-limbo is their idea, we just built it for them
- **Filippo Valsorda inspired the testvector design and allowed us to home x509-limbo under C2SP**
- **Andrew Pan, Facundo Tuesca, Alex Cameron all designed, discussed, and engineered components of the implementation or tests**



banner items

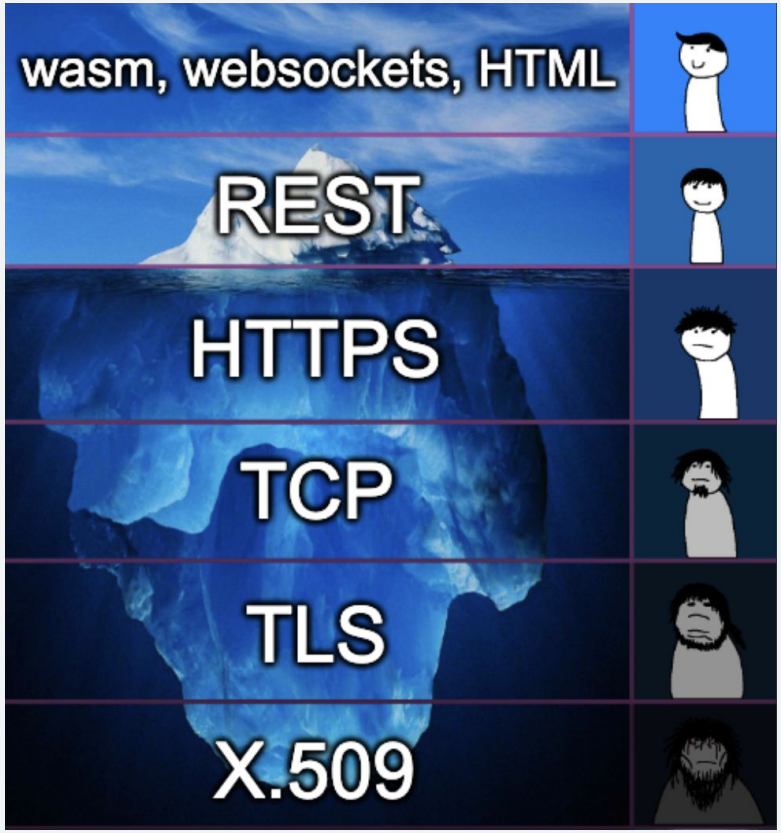
- **Python Cryptography now has X.509 path validation APIs!**
  - work right out of the box with `pip install cryptography`!
  - written in Rust, bound to Python with PyO3!
  - ~2500 new lines of code total
  - no OpenSSL*!
  - use it to replace your pyOpenSSL cruft!
- **X.509 validation is scary, so we tested the crap out of it!**
  - x509-limbo: a new testsuite for path validation implementations!
  - it's already found bugs/discrepancies in other implementations!

`openssl-1.1`	✅	`unhandled critical extension`
`openssl-3.1.5`	✅	`unhandled critical extension`
`gocryptox509-go1.22.0`	❌ (unexpected success)	`validation: chain built`
`rust-webpki`	✅	`trusted certs: trust anchor extraction failed`

why does this ~~boring crap~~ matter?

you & your users are here

your security assumptions are here



why does this matter?

transport security (TLS, the “S” in HTTPS) is table stakes on the modern internet

- **users (rightfully) expect their cat pics, financials, etc. to be protected over the wire**

transport security has a fundamental chicken-and-egg problem:

- **to securely connect to a server, I need that server’s public key**
- **…but I don’t have every possible public key, so I need to obtain them**
- **…but to securely obtain them, I need to securely connect to a server**

🌈X.509 exists to solve this problem🌈

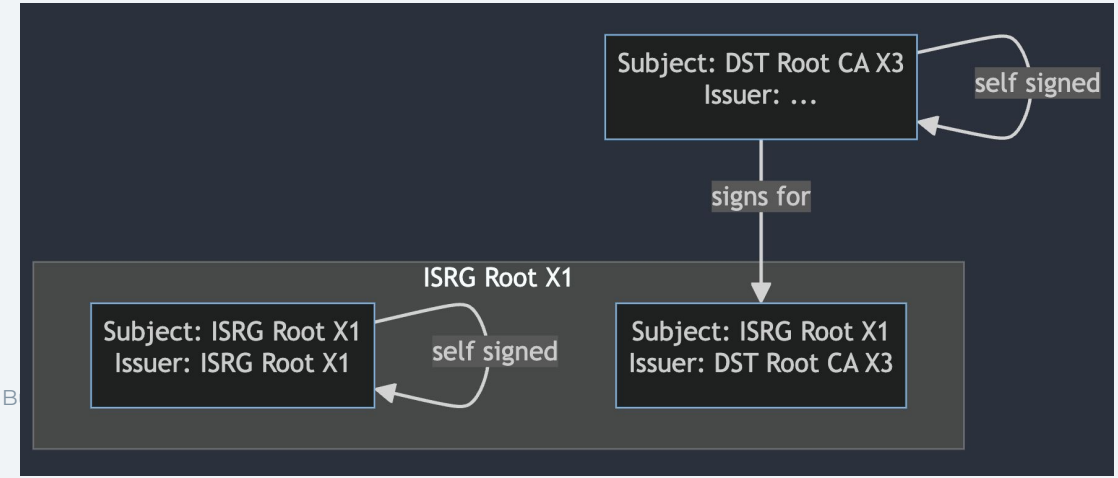
X.509 is…

- **…a public key conveyance & identity-binding format**
  - "Public key `0xblahblahblah` is bound to subject `freestuff.example.com`"
  - Metadata includes expiration, machine & human-readable policy, expected usage, etc.
- **(pubkey, metadata) is signed over by a private key**
  - Usually a different private key than the signed-over pubkey's private half!

```
X509v3 extensions:
    X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
    X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication
    X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Subject Key Identifier:
        79:EF:1A:7B:B3:FB:A5:2C:B3:B1:91:5C:41:44:00:E9:79:4A:E1:3B
    X509v3 Authority Key Identifier:
        14:2E:B3:17:B7:58:56:CB:AE:50:09:40:E6:1F:AF:9D:8B:14:C2:C6
    Authority Information Access:
        OCSP - URI:http://r3.o.lencr.org
        CA Issuers - URI:http://r3.i.lencr.org/
    X509v3 Subject Alternative Name:
        DNS:www.x509-limbo.com, DNS:x509-limbo.com
```

X.509 is…

- **… a chaining primitive:**
  - Certificate A can sign for Certificate B by conveying $pubkey_B$
- **…*not* a one-way key ⟷ cert mapping**
  - A single CA can have multiple keypairs, each with its own certificate
    - …for rotation, fall over, or “just because” reasons
  - A *single* keypair can have multiple certificates!
    - …for policy reasons (e.g. expiry renewal without key rotation)
    - …for cross-issuance purposes (e.g. a CA certificate signed by a *different* CA)



X.509 is…

- **… a protocol for securely introducing principals to each other:**
  - chaining allows us to go from a set of locally trusted entities (the OS/language trust store) to a remotely *untrusted* entity that we know (e.g. by DNS name) but don't previously trust
  - vice versa for mutual authentication ("client" auth in TLS land)
- **principals can be introduced through untrusted intermediates!**
  - `example.com` can be securely connected to regardless of how many intermediate certs it sends, *so long as* the final chain construction is shaped like:

```
[example.com, …, untrusted, trusted]
```

X.509 is...

- **... a complete dumpster fire of an ecosystem**
- **75% of the world runs on OpenSSL and forks (LibreSSL, BoringSSL)**
  - forks are often better but severely constrained by API/ABI compatibility
  - OpenSSL's X.509 implementation is half RFC 3280/5280, half 🤷
  - the other 25% is OS implementations, mbedTLS, WolfSSL, GnuTLS, vendor crap, mystery meat, etc.
  - a bunch of this stuff runs on middleboxes and appliances that are **deployed and promptly forgotten about**
- **...a playground for exploitable logic and memory safety bugs**
  - geometric combination of ASN.1/DER parsing bugs and X.509 protocol/policy errors
  - severity ranges from remote triggerable DoS (moderately bad) to false positive verification (very bad) to remote disclosure of process memory (very, very bad)
  - some of these bugs are baked into the specs themselves!
    - CVE-2023-0464, CVE-2023-23524

can we do X.509 better*?

**** for python***

why *another* implementation?

most Python X.509 APIs are built on OpenSSL:

- **stdlib `ssl` is closely tied to SSL/TLS, not a fully generic implementation**
  - public API exposes OpenSSL implementation details
- **pyOpenSSL provides a generic API**
  - even lower-level and implementation-specific than `ssl`

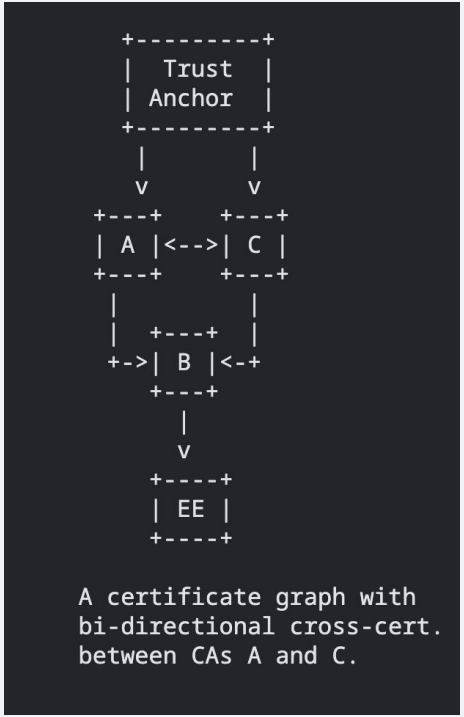
building on OpenSSL means inheriting its spotty security history, bad defaults, performance tarpits, hoping that APIs don't change too much between releases, etc.

we want a clean slate to build *confidently* on!

design principles

above everything, follow Sleevi's laws:

- **there is no one true chain**
- **treat path building as an abstract DFS problem**
  - with dynamic constraints (e.g. accumulated Name Constraints)
- **all rejections must be encoded into the search itself**
  - rejecting a chain after it's built means potentially leaving other chains undiscovered
- **"know your limits"**
  - don't allow arbitrarily deep chains, excessive name/policy comparisons, or anything else that would not appear in a real PKI



design constraints

minimal

- **X.509 is a massive ecosystem with many "profiles," most of which are bad**
- **implement the parts people actually use and care about**

correct

- **almost all path validators are buggy; need *some* bug compat while targeting a compliant implementation**
- **build confidence empirically with tests, including testing other implementations that lack tests**
- **PyCA maintainers (rightfully) require 100% coverage for all changes**

fast

- **basic goal: be faster than OpenSSL**

minimality

in practice, there are only two X.509 profiles that matter for 99.99% of Python users:

- **RFC 5280: the generic "sane defaults" profile for X.509v3**
- **CABF (aka "CA/B TLS BRs"): the Web PKI profile, used by TLS on the web**
  - Superset* of RFC 5280

limiting support to these two profiles has other benefits:

- **hard rejection of the overwhelming majority of key/signature pairings in favor of safe, conservative defaults**
  - in practice: ECDSA and RSA, maybe EdDSA when widely accepted by browsers
- **hard rejection of X.509v1 certs and well-known security pitfalls that older validators often accept**
  - malformed SANs, zeroed serial numbers, etc.

correctness

validation in theory:

- **perform a DFS over the acyclic graph of `leaf -> {intermediates} -> {root}`**
- **while searching, apply path length, name, etc. constraints to reject chains that don't match the overall policy or profile**
- **return first full chain, but exhaust all possible candidates**

correctness

validation in practice:

- **attackers can force cycles in the graph!**
  - classic source of DoS/uncontrolled recursion in X.509
- **pathlen/name constraints are buggy in most implementations**
  - sometimes in load-bearing ways, so we need bug compatibility
- **incomplete chains are present in the real world**
  - e.g. due to websites forgetting to serve their intermediates
- **lots of implementations give up if the first chain candidate is invalid**
  - leaving valid chains undiscovered

performance

path validation performance is largely dominated by parsing and profile validation, not the cryptography itself

significant optimizations:

- **100% of the path building hotpath is compiled Rust***
- **cache parsed and raw keys aggressively**
- **minimize number of extension searches**
- **not calling back into OpenSSL (via rust-openssl) for SPKI unpacking**

```
<Alex_Gaynor> Replaced OpenSSL's public key parser with our own pure rust one (that constructs OpenSSL key types). This made _certificate path building_ 60% faster. It didn't make key parsing 60% faster. It made CERTIFICATE PATH BUILDING 60% FASTER.
```


shipping it

confidence boost

writing a path validator is a fundamentally scary thing to do

…plus we need to get to 100% coverage

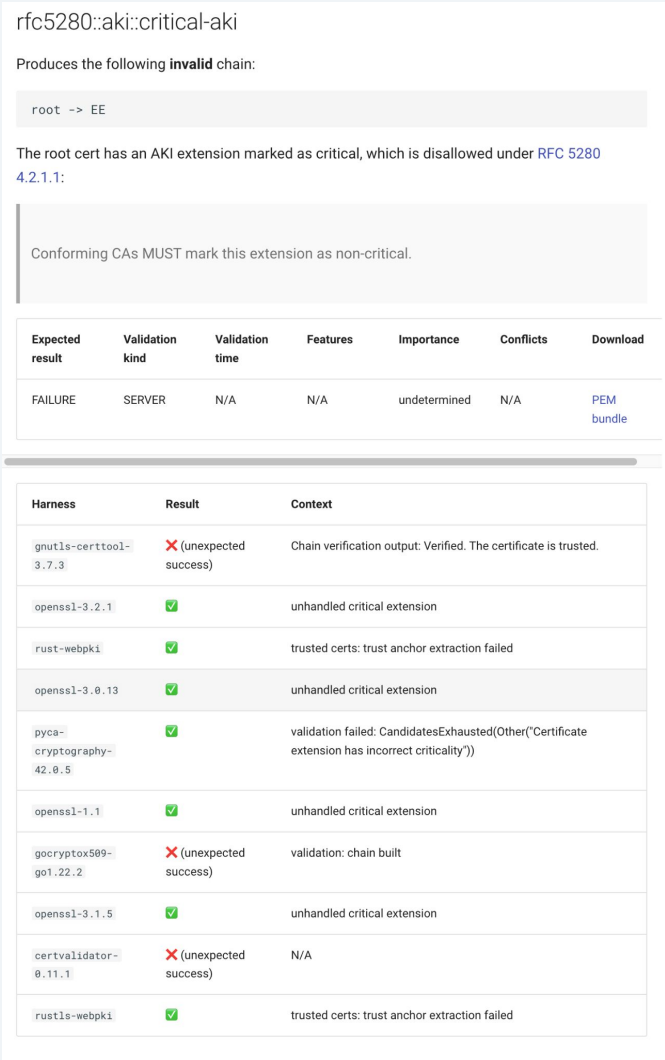
need a more general approach than one-off unit tests:

- **need to express complex graphs, chains that fail deep into validation**
- **need to categorize tests as**
- **need to *generically compare* our behavior to other implementations**
  - both for doing better **and** for bug compatibility when we need it

x509-limbo

x509-limbo is our generic test suite + testvector format

- **each testcase is contains chain state + expected result**
- **compiled into a giant blob of JSON for easy reuse**
- **contains both 3p tests (BetterTLS) and our own tests**
- **grouped into namespaces, e.g. `webpki::` for the CABF profile**



rfc5280::aki::critical-aki

Produces the following **invalid** chain:

```
root -> EE
```

The root cert has an AKI extension marked as critical, which is disallowed under RFC 5280 4.2.1.1:

> Conforming CAs MUST mark this extension as non-critical.

Expected result	Validation kind	Validation time	Features	Importance	Conflicts	Download
FAILURE	SERVER	N/A	N/A	undetermined	N/A	PEM bundle

Harness	Result	Context
gnutls-certtool-3.7.3	❌ (unexpected success)	Chain verification output: Verified. The certificate is trusted.
openssl-3.2.1	✅	unhandled critical extension
rust-webpki	✅	trusted certs: trust anchor extraction failed
openssl-3.0.13	✅	unhandled critical extension
pyca-cryptography-42.0.5	✅	validation failed: CandidatesExhausted(Other("Certificate extension has incorrect criticality"))
openssl-1.1	✅	unhandled critical extension
gocryptox509-go1.22.2	❌ (unexpected success)	validation: chain built
openssl-3.1.5	✅	unhandled critical extension
certvalidator-0.11.1	❌ (unexpected success)	N/A
rustls-webpki	✅	trusted certs: trust anchor extraction failed

x509-limbo

- **brought us to 100% coverage without too much manual testcase work**
- **found a bunch of bugs in other validators in the process, including one CVE**
  - …we've only triaged a few so far

```
tests/x509/__init__.py                                0      0      0      0   100%
tests/x509/test_name.py                              21      0     10      0   100%
tests/x509/test_ocsp.py                             671      0    194      0   100%
tests/x509/test_x509.py                            2136      0    545      0   100%
tests/x509/test_x509_crlbuilder.py                  356      0     98      0   100%
tests/x509/test_x509_ext.py                        2169      0    258      0   100%
tests/x509/test_x509_revokedcertbuilder.py          102      0     34      0   100%
tests/x509/verification/__init__.py                   0      0      0      0   100%
tests/x509/verification/test_limbo.py                67      0     28      0   100%
tests/x509/verification/test_verification.py         92      0     30      0   100%
-------------------------------------------------------------------------------------
TOTAL                                             20902      0   7600      0   100%
```

🏆 Trophy case 🏆

This page tracks notable bugs (and vulnerabilities) identified in various X.509 path validators thanks to x509-limbo.

Have you found or fixed an X.509 validation bug thanks to x509-limbo? Help us out by telling us about it!

Legend:

Symbol	Meaning
💀	CVE or other public vulnerability finding
🧸	Public bugfix

GnuTLS

- 💀 CVE-2024-28835: remote crash caused by an OOB `memcpy` due to a long X.509 chain.

Go (`crypto/x509`)

- 🧸 CL#561615
- 🧸 CL#562341
- 🧸 CL#562342
- 🧸 CL#562343
- 🧸 CL#562344
- 🧸 CL#562975
- 🧸 CL#585076

API design

users shouldn't be able to (accidentally) express post-rejection conditions

- **common source of overly conservative searches with OpenSSL et al.**

fluency, not flags

- **do you want `X509_V_FLAG_PARTIAL_CHAIN` in OpenSSL? the answer may surprise you!**

domain separation

- **server (i.e. DNS) validation and client validation are different and should not be confusable**
- **trust root/anchors are their own data type, to prevent confusion with untrusted certificates**

```python
from cryptography.x509 import Certificate, DNSName, load_pem_x509_certificates
from cryptography.x509.verification import PolicyBuilder, Store
import certifi
from datetime import datetime

with open(certifi.where(), "rb") as pems:
   store = Store(load_pem_x509_certificates(pems.read()))

builder = PolicyBuilder().store(store)
builder = builder.time(verification_time)

verifier = builder.build_server_verifier(DNSName("cryptography.io"))
chain = verifier.verify(peer, untrusted_intermediates)
```


Lessons learned & future work

Lessons learned

RFCs and CABF say one thing, implementations do another

- **CABF says to prefer SAN always; many implementations check Subject or both**
- **many implementations don't bother with serial number checks**
  - Guessable/fixed serials were what made Flame (2008) possible!
- **most implementations have at least one NC bug, causing false negatives**
  - typically in cross-issuance/self-issuance graphs
- **most implementations handle self-issued cert pathlens incorrectly**
- **many implementations ignore/don't enforce presence of SKI/AKI**

Lessons learned

X.509 has plenty of quirks to lawyer over

- **negative 20-octet serial numbers that are really 21 octets when DER encoded, valid or not?**
- `0.999… != 1` **according to RFC 5280**
- **“root” certs are a lie, only “trust anchors” exist**
  - unless you’re in CABF, in which case trust anchors are a lie and only roots exist
- **unclear whether conveying a TA as a certificate means respecting its exts/constraints**
  - PyCA respects constraints on roots, other impls (notably rustls) do not
  - …but some browsers do? Some of the time?
- **unclear whether the leaf’s pubkey is subject to CABF constraints**

Lessons learned

despite complexity and ambiguity, things are Pretty Good™:

- **the Web PKI is in *much* better shape in 2024 than 2014, thanks to CABF, shorter validities, Certificate Transparency, and *higher expectations***
  - no more secretly issued intermediate CAs, no more toothless audit failures
  - no more BER or malformed DER roots, no more V1 roots
  - only a small handful of invalid serials left
- **hardest part was determining and testing PyCA's "break budget" vs. other implementations, not the actual code itself**
  - core path building is <400 lines of well-commented Rust
- **some bugs still snuck through!**
  - root key strength check bug, unknown NC handling bug, datetime object TZ bug, …
  - each got a new x509-limbo case, revealing bugs in other impls too!

Future work

- **other profiles? more configuration knobs?**
  - Intel SGX, Authenticode, etc.
  - users want even more control over which key/signature algorithms are allowed
    - sometimes users want bad things
  - guiding principle: future knobs/config points must not make the current APIs harder to use/easier to misuse
- **revocation APIs (CRLs, OCSPs, etc.)**
  - Cryptography already has APIs for these, but not checked during validation
- **spread the gospel of x509-limbo**
  - get more implementations to adopt it upstream
  - cannibalize more related test suites
  - support profiles other than RFC 5280 and CABF

Thank you!

Slides are available here:

https://yossarian.net/publications#pycon-2024

Resources:

- **ToB blog:** We build X.509 chains so you don't have to
- **Ryan Sleevi:** Path Building vs Path Verifying: The Chain of Pain
- **Andrew Ayer:** Fixing the Breakage from the AddTrust External CA Root Expiration
- **Robert Alexander:** Name "Constrain't" on Chrome
- **Many specs:**
  - RFC 5280 (X.509 PKIX)
  - RFC 6125 (domain-based identities, wildcards in PKIX)
  - CABF BRs