Wounded hero revived

Lessons learned from porting M2Crypto to Py3k

Matěj Cepl <mcepl@cepl.eu>

https://matej.ceplovi.cz/clanky/PyCon18-m2crypto/

Origin

- There are two goals of this talk:
  - pulling dead projects out of their graves
  - porting to Python 3

Origin

- M2Crypto was a leftover from a failed project Chandler.
- One of its problems was trying to do everything on their own, and so they wrote their own crypto binding.
- Project collapsed 2009, but final funds for the M2Crypto maintainer run out on 2011.

M2Crypto

- Maintained in Red Hat by Miloslav Trmač; not having upstream collected all patches in the packaging repository.
- I expected in May 2015 that its few users can use an upstream consisting from original and a few patches on top.
- But users were not few, so I needed to do something thinking about the future of the project. Something like SWOT analysis.

Strengths and weaknesses

- Backed up by stable C library
- Rather large coverage of OpenSSL API
- Surprisingly widespread use
- Large test suite

- Unmapped issues
- Python 3
- M2Crypto API too close to OpenSSL
- No support for Mac OS X and Windows

Opportunities & Threats

- Satisfying current user base
- Replacing PyCrypto & co. (home-brew implementations of crypto algorithms)
- Goal of maintenance is to maintain API
- Extend support on non-Linux platforms

- Python `ssl` module
- Python cryptography project.

Strategy

- Two layers: Python & C API
- Documentation strings and type hints
- CI
- Extension of platform support

C API

- All Unicode/bytes translation happens on C level as well
- Two targets (py2k and py3k) and two backends (OpenSSL 1.0 and 1.1 API)
- Based on `swig`, which fortunately natively supports `--py3`.
- Minimize use of `#ifdef` s and rather use included shims for missing functions.

C shims of missing functions

- For OpenSSL < 1.1
- For Python 2
  - `PyLong_FromLong()` and `PyUnicode_AsUTF8()` just simple `#defines`.
  - All Pythons >= 2.6 contain whole set of Py3k function stubs in `bytesobject.h`.
- For Python 3
  - `PyFile_AsFile()` I have no idea, why it was removed from py3k API

Type Hints

- PEP 484 providing **optional** type annotations. Quite controversial, but clearly very useful for libraries
- Native for Python >= 3.5, but supports py2k compatible syntax:

```
def sum(x, y):
    # type: (int, int) -> int
    return x + y
```

  Especially useful for our situation: marking types helps us to analyze what individual py2k `str` actually mean.

Python porting (shims again)

The same principles apply as with C functions, “shims, not `#ifdefs`“.

- Plenty of issues are resolved by using `six` (or `modernize`, `future`), so use them.
- Do not hesitate to create your own shims. So I have for example, `bin_to_hex` and `oct_to_num` or padding functions there.
- http://python-future.org/compatible_idioms.html

Questions?

- https://gitlab.com/m2crypto/m2crypto/
- `mcepl@cepl.eu` or `@mcepl`