Hackowanie zamrożonych binariów

Piotr Tynecki

Białystok 2014

Agenda

1. Inżynieria wsteczna oprogramowania
2. Tradycyjny model wykonawczy kodu Pythona
3. Kod bajtowy, charakterystyka plików .pyc / .pyo
4. Dekompilatory pythonowego kodu bajtowego
5. Exe-packery, „frozen binaries”
6. „All in one”, żeby haczyło się lepiej

Agenda

1. Inżynieria wsteczna oprogramowania
2. Tradycyjny model wykonawczy kodu Pythona
3. Kod bajtowy, charakterystyka plików .pyc / .pyo
4. Dekompilatory pythonowego kodu bajtowego
5. Exe-packery, „frozen binaries”
6. „All in one”, żeby haczyło się lepiej

Inżynieria wsteczna

Proces odwrócenia cyklu produkcji oprogramowania, pozwalający opisać działanie oprogramowania na wyższej warstwie abstrakcji (np. diagramy ERD, UML)

Inżynieria wsteczna oprogramowania

Zbiór czynności działających łącznie, cofających pracę kompilatorów do postaci kodu języka programowania zrozumiałego dla człowieka

Inżynieria wsteczna pythonowych binariów

Plik wykonywalny (np. exe)

↓ (dekompresja np. UPX)

Wypakowanie archiwum ZIP / zlib

↓

Dekompilacja kodu bajtowego .pyc / .pyo

↓

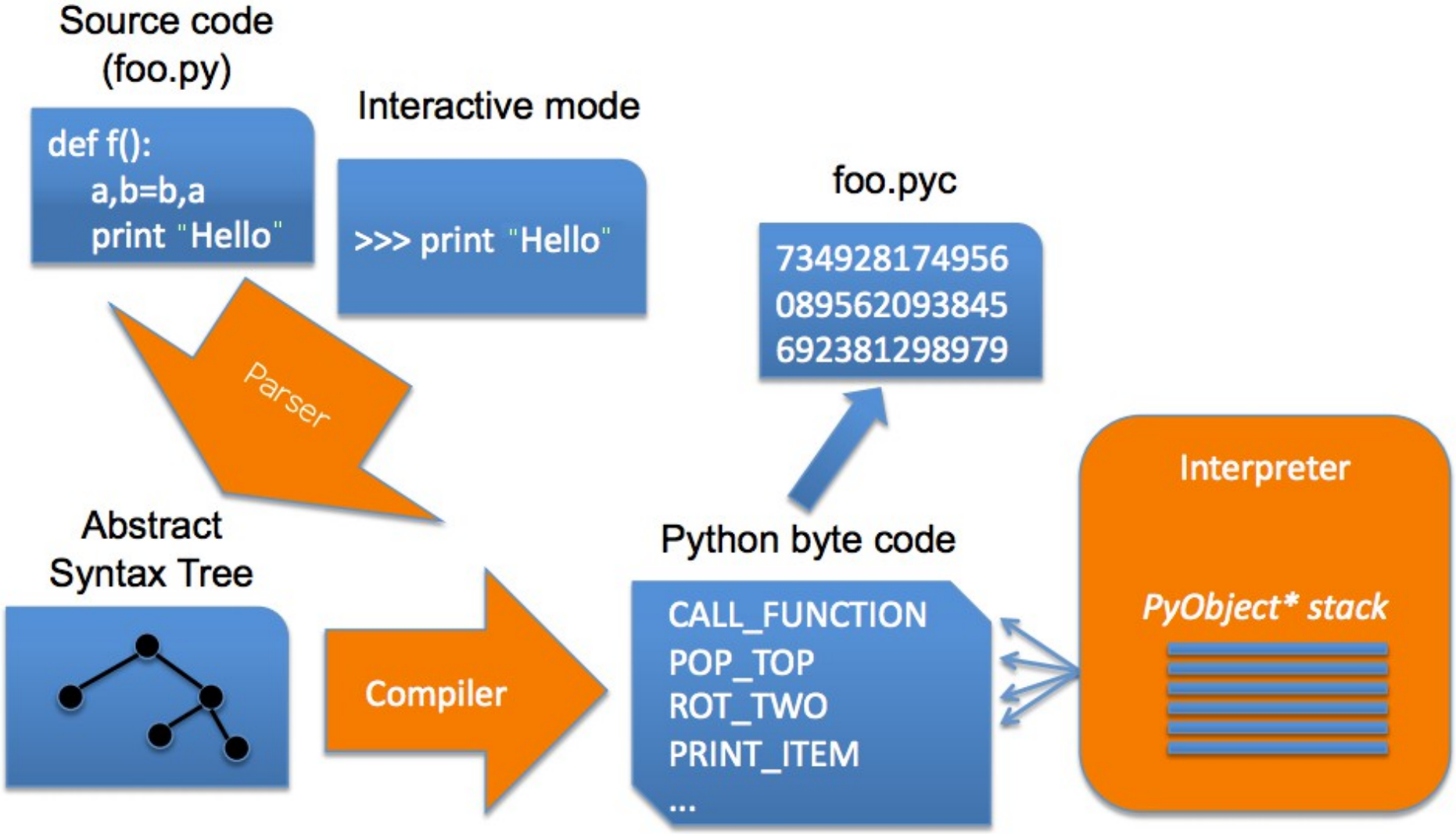
Oryginalna struktura katalogów

z plikami źródłowymi .py i multimediami

Agenda

1. Inżynieria wsteczna oprogramowania
2. Tradycyjny model wykonawczy kodu Pythona
3. Kod bajtowy, charakterystyka plików .pyc / .pyo
4. Dekompilatory pythonowego kodu bajtowego
5. Exe-packery, „frozen binaries”
6. „All in one”, żeby haczyło się lepiej

CPython Compiler & Interpreter



Agenda

1. Inżynieria wsteczna oprogramowania
2. Tradycyjny model wykonawczy kodu Pythona
3. Kod bajtowy, charakterystyka plików .pyc / .pyo
4. Dekompilatory pythonowego kodu bajtowego
5. Exe-packery, „frozen binaries”
6. „All in one”, żeby haczyło się lepiej

Kod bajtowy, charakterystyka plików .pyc / .pyo

Kod bajtowy - niskopoziomowa, niezależna od platformy reprezentacja kodu źródłowego wykonywana przez VM

Kod bajtowy (C)Pythona - sekwencja serializowanych obiektów Pythona w postaci **code objects**

Code objects – nieedytowalne, wykonywalne obiekty Pythona reprezentujące kawałki kodu bajtowego (opcodes) wraz ze zmiennymi lokalnymi, stałymi i metadanymi

Kod bajtowy, charakterystyka plików .pyc / .pyo

```
>>> code_str = """
... print("Hello PyStok #1!")
... print(u"2 ** 10 = {0}".format(2 ** 10))
... """
>>> code_obj = compile(code_str, '<string>', 'exec')
>>> exec code_obj
Hello PyStok #1!
2 ** 10 = 1024
>>>
```

Kod bajtowy, charakterystyka plików .pyc / .pyo

```
>>> def foo(x, y):
...     return x + y
...
>>> dis.dis(foo)
  2           0 LOAD_FAST                0 (x)
              3 LOAD_FAST                1 (y)
              6 BINARY_ADD
              7 RETURN_VALUE
>>>
```

Kod bajtowy, charakterystyka plików .pyc / .pyo

Pierwsze 4 bajty - **magic number**

Wartość określająca wersję Pythona, użytą do kompilacji kodu (dwa ostatnie bajty to CR i LF)

Kolejne 4 bajty - **timestamp**

Data modyfikacji pliku źródłowego .py

Pozostała część pliku - **code object**

Serializowane obiekty Pythona (marshal)

Kod bajtowy, charakterystyka plików .pyc / .pyo

.pyc != .pyo

.pyo to:

__debug__ na False

Pomija asercję (assert)

Usuwa docstringi

.pyo powstaje na wskutek użycia flagi -O lub -OO

+ Krótszy czas ładowania kodu bajtowego

+ Redukcja wielkości pliku rzędu kilku / kilkunastu KB

Kod bajtowy, charakterystyka plików .pyc / .pyo

```
>>> magic_numbers = {
        20121: 'Python 1.5.x', 50428: 'Python 1.6', 50823: 'Python 2.0.x',
        60202: 'Python 2.1.x', 60717: 'Python 2.2', 62011: 'Python 2.3a0',
        62021: 'Python 2.3a0', 62041: 'Python 2.4a0', 62051: 'Python 2.4a3',
        62061: 'Python 2.4b1', 62071: 'Python 2.5a0', 62081: 'Python 2.5a0',
        62091: 'Python 2.5a0', 62092: 'Python 2.5a0', 62101: 'Python 2.5b3',
        62111: 'Python 2.5b3', 62121: 'Python 2.5c1', 62131: 'Python 2.5c2',
        62151: 'Python 2.6a0', 62161: 'Python 2.6a1', 62171: 'Python 2.7a0',
        62181: 'Python 2.7a0', 62191: 'Python 2.7a0', 62201: 'Python 2.7a0',
        62211: 'Python 2.7a0', 3000: 'Python 3000', 3010: 'Python 3000',
        3020: 'Python 3000', 3030: 'Python 3000', 3040: 'Python 3000',
        3050: 'Python 3000', 3060: 'Python 3000', 3061: 'Python 3000',
        3071: 'Python 3000', 3081: 'Python 3000', 3091: 'Python 3000',
        3101: 'Python 3000', 3103: 'Python 3000', 3111: 'Python 3.0a4',
        3131: 'Python 3.0a5', 3141: 'Python 3.1a0', 3151: 'Python 3.1a0',
        3160: 'Python 3.2a0', 3170: 'Python 3.2a1', 3180: 'Python 3.2a2',
        3190: 'Python 3.3a0', 3200: 'Python 3.3a0', 3210: 'Python 3.3a0',
        3220: 'Python 3.3a1', 3230: 'Python 3.3a4', 3250: 'Python 3.4a1',
        3260: 'Python 3.4a1', 3270: 'Python 3.4a1', 3280: 'Python 3.4a1',
        3290: 'Python 3.4a4', 3300: 'Python 3.4a4', 3310: 'Python 3.4rc2',
        3320: 'Python 3.5a0',
    }
```

Kod bajtowy, charakterystyka plików .pyc / .pyo

```
>>> import struct
>>> with open('pystok-27.pyc', 'rb') as f:
...     magic_number_hex = f.read(4)
...     magic_number_value = struct.unpack('H2B', magic_number_hex)
...     python_version = magic_numbers.get(magic_number_value[0], '')
...     print(python_version)
...
Python 2.7a0
```

Kod bajtowy, charakterystyka plików .pyc / .pyo

```
>>> import marshal
>>> source = open('pystok-27.pyc', 'rb').read()
>>> code_objects = marshal.loads(source[8:])
>>> dir(code_objects)
['__class__', '__cmp__', '__delattr__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__le__', '__lt__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'co_argcount', 'co_cellvars', 'co_code',
'co_consts', 'co_filename', 'co_firstlineno', 'co_flags', 'co_freevars',
'co_lnotab', 'co_name', 'co_names', 'co_nlocals', 'co_stacksize',
'co_varnames']
```

Agenda

1. Inżynieria wsteczna oprogramowania
2. Tradycyjny model wykonawczy kodu Pythona
3. Kod bajtowy, charakterystyka plików .pyc / .pyo
4. **Dekompilatory pythonowego kodu bajtowego**
5. Exe-packery, „frozen binaries”
6. „All in one”, żeby haczyło się lepiej

Dekompilatory pythonowego kodu bajtowego

Python 2.5 – 2.6

http://sourceforge.net/projects/unpyc

Python 2.7

https://github.com/wibiti/uncompyle2

Python 3.2

https://code.google.com/p/unpyc3

Python 2.6 - 2.7

https://github.com/MyNameIsMeerkat/pyREtic

Python 1.0 – Python 3.3 (niebawem 3.4)

https://github.com/zrax/pycdc

Dekompilatory pythonowego kodu bajtowego

```
>>> from uncompyle2 import uncompyle_file
>>> from uncompyle2 import Walker
>>> f = open('pystok-27.py', 'w')
>>> try:
...     uncompyle_file('pystok-27.pyc', f)
... except (IndexError, Walker.ParserError):
...     raise Exception("Decompilation failed")
>>> f.close()
```

Dekompilatory pythonowego kodu bajtowego

```
>>> from unpyc3.unpyc3 import dec_module
>>> with open('pystok-34.py', 'w') as f:
...     f.write(str(dec_module('__pycache__/pystok-34.cpython-34.pyc')))
...
>>>
```

```
katharsis@toshi:~/pystok$ ./pycdc/pycdc pystok-2-27.pyc
# Source Generated with Decompyle++
# File: pystok-2-27.pyc (Python 2.7)

print u'PyStok #1 - przyk\xc5\x82ad 2'
print map(float, xrange(20))
```

Agenda

1. Inżynieria wsteczna oprogramowania
2. Tradycyjny model wykonawczy kodu Pythona
3. Kod bajtowy, charakterystyka plików .pyc / .pyo
4. Dekompilatory pythonowego kodu bajtowego
5. Exe-packery, „frozen binaries”
6. „All in one”, żeby haczyło się lepiej

Exe-packery, „frozen binaries”

Zamrożone binaria Pythona – samodzielne pliki wykonywalne (np. .exe, .app), obejmujące kod bajtowy skryptów Pythona, maszynę wirtualną (PVM) oraz biblioteki dynamiczne (np. .dll, .so), niezbędne do ich działania

W rzeczywistości są to uruchamialne archiwa ZIP / zlib
o rozmiarze sięgającym od kilku do kilkunastu MB

Niektóre mogą być dystrybuowane jako pojedyncze pliki binarne

Exe-packery, „frozen binaries”

Exe-packer	py2exe	py2app	cx_Freeze	bbfreeze	PyInstaller
Platforma	Windows	Mac OS X	Windows, Mac OS X, GNU/Linux	Windows, GNU/Linux	Windows, Mac OS X, GNU/Linux
Python	2.4 - 2.7 3.3 - 3.4	2.4 - 2.7	2.4 - 2.7 3.3 - 3.4	2.4 - 2.7	2.4 - 2.7
Single executable file	Tak	Tak	Nie	Nie	Tak
Algorytmy kompresji	marshal, ZIP	ZIP, DMG	ZIP	ZIP	marshal, zlib
Licencja	MIT	MIT / PSF	PSF	MIT	GPL

Inżynieria wsteczna binarów - py2exe

Zlokalizowanie w nagłówku pliku binarnego sekcji `PYTHONSCRIPT`

↓ (pefile, DIRECTORY_ENTRY_RESOURCE)

Wykonanie zrzutu zasobów

↓

Deserializacja osadzonego wewnątrz binariów kodu bajtowego (marshal) do postaci plików .pyc z uwzględnieniem magic number i timestamp

↓

Wypakowanie archiwum .zip lub pliku binarnego

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

Inżynieria wsteczna binarów - py2exe

Zlokalizowanie w nagłówku pliku binarnego sekcji `PYTHONSCRIPT`

↓

Wykonanie zrzutu zasobów

↓ (pefile.get_data)

Deserializacja osadzonego wewnątrz binariów kodu bajtowego (marshal) do postaci plików .pyc z uwzględnieniem magic number i timestamp

↓

Wypakowanie archiwum .zip lub pliku binarnego

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

Inżynieria wsteczna binarów - py2exe

Zlokalizowanie w nagłówku pliku binarnego sekcji `PYTHONSCRIPT`

↓

Wykonanie zrzutu zasobów

↓

Deserializacja osadzonego wewnątrz binariów kodu bajtowego (marshal) do postaci plików .pyc z uwzględnieniem magic number i timestamp

↓ (marshal.loads / marshal.dumps)

Wypakowanie archiwum .zip lub pliku binarnego

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

Inżynieria wsteczna binarów - py2exe

Zlokalizowanie w nagłówku pliku binarnego sekcji `PYTHONSCRIPT`

↓

Wykonanie zrzutu zasobów

↓

Deserializacja osadzonego wewnątrz binariów kodu bajtowego (marshal) do postaci plików .pyc z uwzględnieniem magic number i timestamp

↓

Wypakowanie archiwum .zip lub pliku binarnego

↓ (zipfile)

Dekompilacja kodu bajtowego do postaci źródłowej .py

Inżynieria wsteczna binarów - py2exe

Zlokalizowanie w nagłówku pliku binarnego sekcji `PYTHONSCRIPT`

↓

Wykonanie zrzutu zasobów

↓

Deserializacja osadzonego wewnątrz binariów kodu bajtowego (marshal) do postaci plików .pyc z uwzględnieniem magic number i timestamp

↓

Wypakowanie archiwum .zip lub pliku binarnego

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

DEMO

unfrozen_binary_py2exe

Inżynieria wsteczna binarów - cx_Freeze

Wypakowanie archiwum .zip lub pliku binarnego

↓ (zipfile)

Dekompilacja kodu bajtowego do postaci źródłowej .py

↓

Zmiana nazwy `__main__.py` lub `<nazwa_skryptu>__main__.py`

na nazwę zawartą w komentarzu źródła

Inżynieria wsteczna binarów - cx_Freeze

Wypakowanie archiwum .zip lub pliku binarnego

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

↓

Zmiana nazwy `__main__.py` lub `<nazwa_skryptu>__main__.py`

na nazwę zawartą w komentarzu źródła

Inżynieria wsteczna binarów - cx_Freeze

Wypakowanie archiwum .zip lub pliku binarnego

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

↓

Zmiana nazwy `__main__.py` lub `<nazwa_skryptu>__main__.py`

na nazwę zawartą w komentarzu źródła

Inżynieria wsteczna binarów - cx_Freeze

DEMO

unfrozen_binary_cx_Freeze

Inżynieria wsteczna binarów - bbfreeze

Wypakowanie archiwum `library.zip`

↓ (zipfile)

Dekompilacja kodu bajtowego do postaci źródłowej .py

↓

Zmiana nazwy `__main__<nazwa_skryptu>.py`

na `<nazwa_skryptu>.py`

Inżynieria wsteczna binarów - bbfreeze

Wypakowanie archiwum `library.zip`

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

↓

Zmiana nazwy `__main__<nazwa_skryptu>.py`

na `<nazwa_skryptu>.py`

Inżynieria wsteczna binarów - bbfreeze

Wypakowanie archiwum `library.zip`

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

↓

Zmiana nazwy `__main__<nazwa_skryptu>.py`

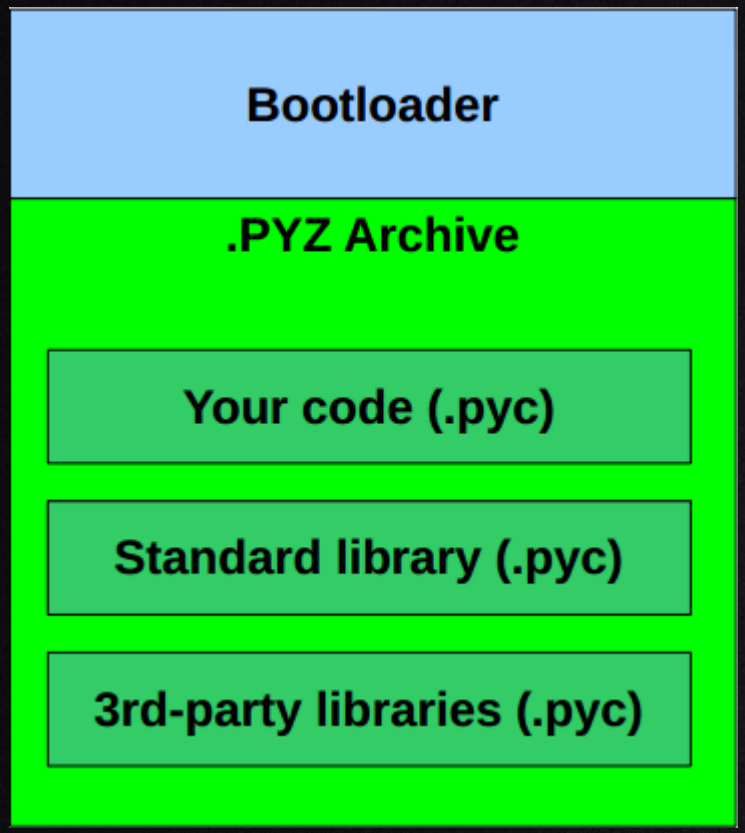
na `<nazwa_skryptu>.py`

Inżynieria wsteczna binarów - bbfreeze

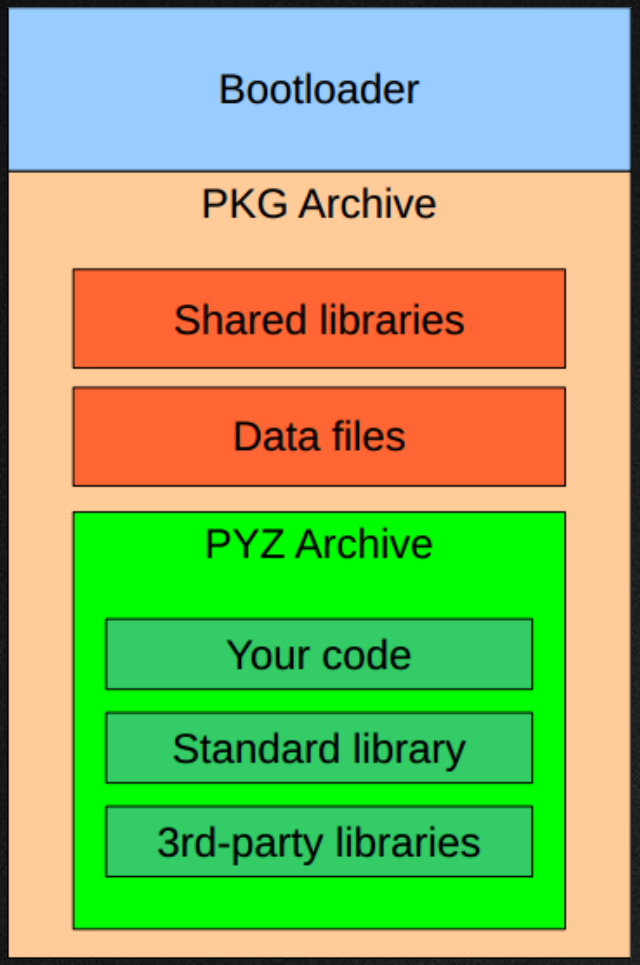
DEMO

unfrozen_binary_bbfreeze

Inżynieria wsteczna binarów - PyInstaller



PyInstaller domyślnie



PyInstaller *Single Executable File*

Inżynieria wsteczna binarów - PyInstaller

Określenie pozycji ciągu `MEI\014\013\012\013\016`

↓ (EOF, COOKIE_SIZE)

Wypakowanie CArchive

↓

Wypakowanie ZlibArchive (PYZ)

↓

Dodanie magic i timestamp dla plików .pyc

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

Inżynieria wsteczna binarów - PyInstaller

Określenie pozycji ciągu `MEI\014\013\012\013\016`

↓

Wypakowanie CArchive

↓ (usunięcie ostatniego znaku dla źródeł .py)

Wypakowanie ZlibArchive (PYZ)

↓

Dodanie magic i timestamp dla plików .pyc

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

Inżynieria wsteczna binarów - PyInstaller

Określenie pozycji ciągu `MEI\014\013\012\013\016`

↓

Wypakowanie CArchive

↓

Wypakowanie ZlibArchive (PYZ)

↓ (marshal.load, zlib.decompress)

Dodanie magic i timestamp dla plików .pyc

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

Inżynieria wsteczna binarów - PyInstaller

Określenie pozycji ciągu `MEI\014\013\012\013\016`

↓

Wypakowanie CArchive

↓

Wypakowanie ZlibArchive (PYZ)

↓

Dodanie magic i timestamp dla plików .pyc

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

Inżynieria wsteczna binarów - PyInstaller

Określenie pozycji ciągu `MEI\014\013\012\013\016`

↓

Wypakowanie CArchive

↓

Wypakowanie ZlibArchive (PYZ)

↓

Dodanie magic i timestamp dla plików .pyc

↓

Dekompilacja kodu bajtowego do postaci źródłowej .py

DEMO

unfrozen_binary_PyInstaller

Agenda

1. Inżynieria wsteczna oprogramowania
2. Tradycyjny model wykonawczy kodu Pythona
3. Kod bajtowy, charakterystyka plików .pyc / .pyo
4. Dekompilatory pythonowego kodu bajtowego
5. Exe-packery, „frozen binaries”
6. „All in one”, żeby haczyło się lepiej

„All in one”, żeby haczyło się lepiej

unfrozen_binary

- pełna automatyzacja dekompresji i dekompilacji,
- wsparcie dla binariów py2app,
- backend dla dekompilatora pycdc,
- raportowanie (INFO, DEBUG, FAILED),
- dokumentacja.

„All in one”, żeby haczyło się lepiej

https://github.com/Katharsis/unfrozen_binary

Dziękuję

Pytania?