Hackowanie
zamrożonych binariów

Piotr Tynecki
@ptynecki

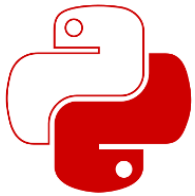
PyCon PL 2015

PyStok

Białostocka Grupa Użytkowników Pythona

python™

polish python coders group



PyConPl'15

Stowarzyszenie Polska Grupa Użytkowników Pythona

PyData

programistok

Agenda

0x0. Wprowadzenie

0x1. Tradycyjny model wykonaczy kodu CPythona

0x2. Kod bajtowy, charakterystyka plików .pyc / .pyo

0x3. Zamrożone binaria CPythona

0x4. Exe-packery

0x5. Dekompilatory pythonowego kodu bajtowego

0x6. Inżynieria wsteczna binariów

0x7. „Co robić, jak żyć?”

Agenda

0x0. Wprowadzenie

0x1. Tradycyjny model wykonaczy kodu CPythona

0x2. Kod bajtowy, charakterystyka plików .pyc / .pyo

0x3. Zamrożone binaria CPythona

0x4. Exe-packery

0x5. Dekompilatory pythonowego kodu bajtowego

0x6. Inżynieria wsteczna binariów

0x7. „Co robić, jak żyć?”

Wprowadzenie

Analiza pliku wykonywalnego

∇ (dekompresja np. UPX)

Deserializacja kodu bajtowego

Wypakowanie archiwum ZIP / zlib

∇

Dekompilacja kodu bajtowego

∇ (deobfuskacja)

Odczyt plików źródłowych,

konfiguracyjnych i multimedialnych

Agenda

0x0. Wprowadzenie

0x1. Tradycyjny model wykonaczy kodu CPythona

0x2. Kod bajtowy, charakterystyka plików .pyc / .pyo

0x3. Zamrożone binaria CPythona

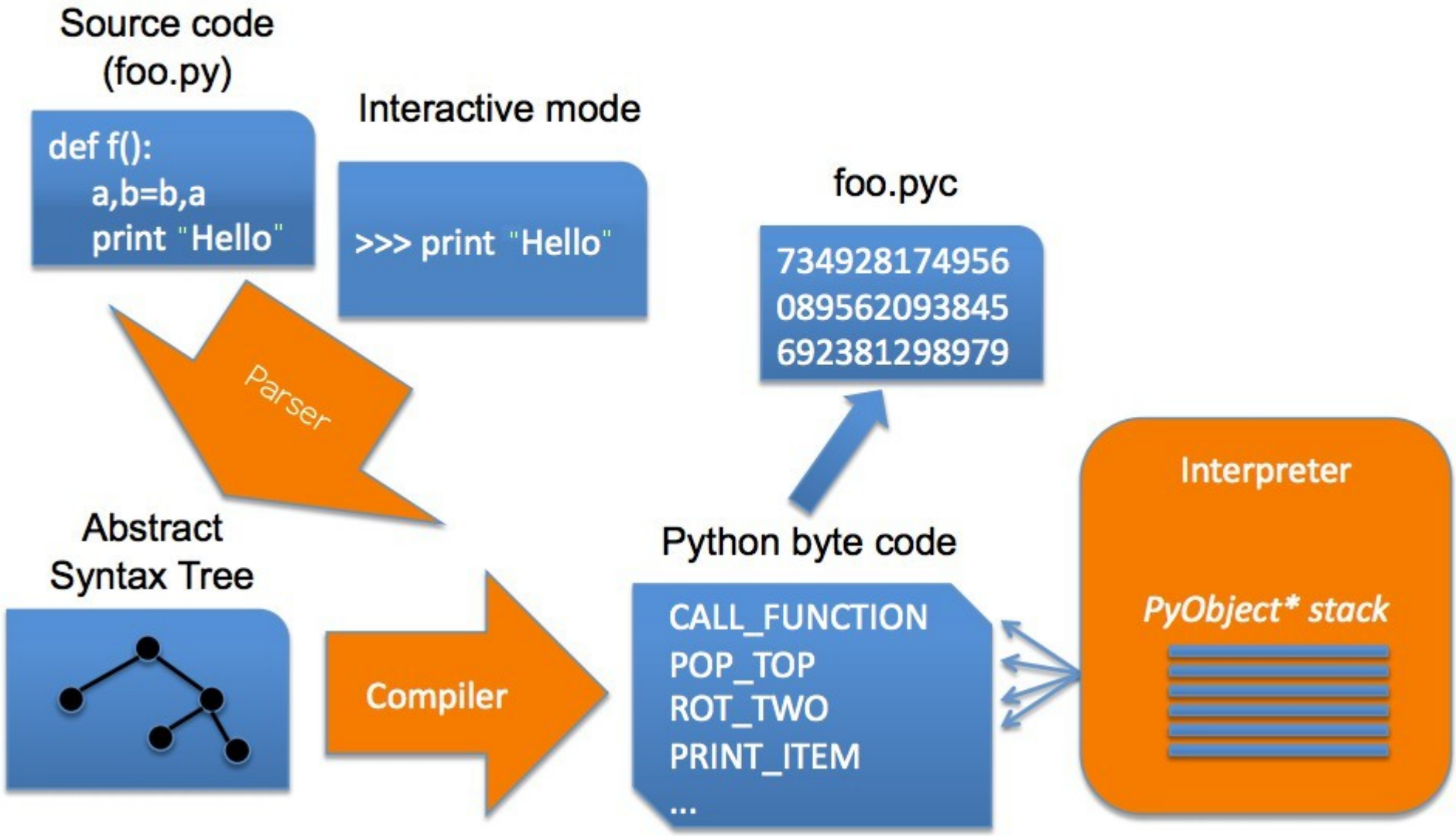
0x4. Exe-packery

0x5. Dekompilatory pythonowego kodu bajtowego

0x6. Inżynieria wsteczna binariów

0x7. „Co robić, jak żyć?”

CPython Compiler & Interpreter



Agenda

0x0. Wprowadzenie

0x1. Tradycyjny model wykonaczy kodu CPythona

0x2. Kod bajtowy, charakterystyka plików .pyc / .pyo

0x3. Zamrożone binaria CPythona

0x4. Exe-packery

0x5. Dekompilatory pythonowego kodu bajtowego

0x6. Inżynieria wsteczna binariów

0x7. „Co robić, jak żyć?”

Kod bajtowy

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

Kod bajtowy CPythona - 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

```
In [1]: code_str = """
print("Hello PyCon PL 2015")
print("2 ** 10 = {}".format(2 ** 10))
"""

In [2]: code_obj = compile(code_str, '<string>', 'exec')

In [3]: exec code_obj
Hello PyCon PL 2015
2 ** 10 = 1024
```

Kod bajtowy

```
In [4]: import dis

In [5]: def foo(x, y):
   ....:     return x + y
   ....:

In [6]: dis.dis(foo)
  2           0 LOAD_FAST                0 (x)
              3 LOAD_FAST                1 (y)
              6 BINARY_ADD
              7 RETURN_VALUE
```

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 objects

Serializowane obiekty Pythona (marshal)

Charakterystyka plików .pyc / .pyo

.pyc != .pyo

.pyo to „optimized bytecode”, czyli:

- __debug__ na False
- Pominięta asercja (assert)
- Brak docstringów

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

Charakterystyka plików .pyc / .pyo

.pyc != .pyo

.pyo to „optimized bytecode”, czyli:

- __debug__ na False
- Pominięta asercja (assert)
- Brak docstringów

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

+ Krótszy czas ładowania kodu bajtowego

+ Redukcja wielkości pliku rzędu kilku / kilkudziesięciu KB

Charakterystyka plików .pyc / .pyo

```
In [7]: 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', 3330: 'Python 3.5a1'
   ....: }
```

Charakterystyka plików .pyc / .pyo

```
In [8]: import struct

In [9]: def print_python_version(filename):
   ....:     with open(filename, 'rb') as bytecode:
   ....:         magic_number_hex = bytecode.read(4)
   ....:         magic_number_value = struct.unpack('H2B', magic_number_hex)
   ....:         python_version = magic_numbers.get(magic_number_value[0], '')
   ....:         print python_version
   ....:

In [10]: print_python_version('pycon-pl-2015-27.pyc')
Python 2.7a0

In [11]: print_python_version('__pycache__/pycon-pl-2015-34.cpython-34.pyc')
Python 3.4rc2
```

Charakterystyka plików .pyc / .pyo

```
In [12]: def print_code_objects(filename):
   ....:     with open(filename, 'rb') as source:
   ....:         code_objects = marshal.loads(source.read()[8:])
   ....:         print dir(code_objects)
   ....:

In [13]: print_code_objects('pycon-pl-2015-27.pyc')
['__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

0x0. Wprowadzenie

0x1. Tradycyjny model wykonaczy kodu CPythona

0x2. Kod bajtowy, charakterystyka plików .pyc / .pyo

0x3. Zamrożone binaria CPythona

0x4. Exe-packery

0x5. Dekompilatory pythonowego kodu bajtowego

0x6. Inżynieria wsteczna binariów

0x7. „Co robić, jak żyć?”

Zamrożone binaria CPythona

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 kilkudziesięciu MB

Mogą być także dystrybuowane jako pojedyncze pliki binarne

Agenda

0x0. Wprowadzenie

0x1. Tradycyjny model wykonaczy kodu CPythona

0x2. Kod bajtowy, charakterystyka plików .pyc / .pyo

0x3. Zamrożone binaria CPythona

0x4. Exe-packery

0x5. Dekompilatory pythonowego kodu bajtowego

0x6. Inżynieria wsteczna binariów

0x7. „Co robić, jak żyć?”

Exe-packery

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 – 3.5	2.4 – 2.7	2.4 – 2.7 3.3 – 3.5	2.4 – 2.7	2.4 – 2.7 3.3 – 3.5
Single executable file	Tak	Tak	Nie	Nie	Tak
Algorytm kompresji	marshal ZIP	ZIP, DMG	ZIP	ZIP	marshal zlib
Licencja	MIT	MIT	PSF	MIT	GPL

Agenda

0x0. Wprowadzenie

0x1. Tradycyjny model wykonaczy kodu CPythona

0x2. Kod bajtowy, charakterystyka plików .pyc / .pyo

0x3. Zamrożone binaria CPythona

0x4. Exe-packery

0x5. Dekompilatory pythonowego kodu bajtowego

0x6. Inżynieria wsteczna binariów

0x7. „Co robić, jak żyć?”

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 3.5

https://github.com/zrax/**pycdc**

Agenda

0x0. Wprowadzenie

0x1. Tradycyjny model wykonaczy kodu CPythona

0x2. Kod bajtowy, charakterystyka plików .pyc / .pyo

0x3. Zamrożone binaria CPythona

0x4. Exe-packery

0x5. Dekompilatory pythonowego kodu bajtowego

0x6. Inżynieria wsteczna binariów

0x7. „Co robić, jak żyć?”

Inżynieria wsteczna binariów

```
katharsis@toshi:~/workspace/pycon-pl-2015/$ ./pycdc/pycdc pycon-pl-2015-27.pyc
# Source Generated with Decompyle++
# File: pycon-pl-2015-27.pyc (Python 2.7)
print u'Przykład 2'
print map(float, xrange(20))
```

```
from uncompyle2 import uncompyle_file
from uncompyle2 import Walker
f = open('pycon-pl-2015-27.py', 'w')
try:
    uncompyle_file('pycon-pl-2015-27.pyc', f)
except (IndexError, Walker.ParserError):
    raise Exception("Decompilation failed")
f.close()
```

```
from unpyc3.unpyc3 import dec_module
with open('pb-obrona-34.py', 'w') as f:
    f.write(str(dec_module('__pycache__/pycon-pl-2015-34.cpython-34.pyc')))
```

unfrozen_binary

https://github.com/Katharsis/unfrozen_binary

Agenda

0x0. Wprowadzenie

0x1. Tradycyjny model wykonaczy kodu CPythona

0x2. Kod bajtowy, charakterystyka plików .pyc / .pyo

0x3. Zamrożone binaria CPythona

0x4. Exe-packery

0x5. Dekompilatory pythonowego kodu bajtowego

0x6. Inżynieria wsteczna binariów

0x7. „Co robić, jak żyć?”

„Co robić, jak żyć?”

Pomysł I

Portowanie wrażliwej logiki biznesowej
klienta desktopowego
na rozwiązanie webowe

„Co robić, jak żyć?”

Pomysł II

http://cython.org/

„Co robić, jak żyć?”

example.py

Plik źródłowy zgodny z syntaktyką Cythona

▽

example.c

Generowanie pliku z optymalnym kodem C

▽

example.so / example.pyd

Skompilowana biblioteka wspólna

gotowa do użycia z poziomu CPythona

„Co robić, jak żyć?”

Pomysł III

Modyfikacje na poziomie PVM

(custom opcode.h,

zmiana magic number i import hooka)

+ obfuskacja kodu Pythona

http://bits.citrusbyte.com/protecting-a-python-codebase-part-3/

„Co robić, jak żyć?

Pomysł III

https://github.com/kholia/AntiDrive

https://pl.python.org/dropbox,desktop,client,zhakowany.html

DZIĘKUJĘ

Pytania?