PyCon China 2020

PyConChina 2020
PYTHON 中国开发者大会 2020

Python For Good

如何优雅的加密Python脚本

赵俊德

西安德新软件

常用保护方式

- 编译成为 .pyc/.pyo
- 打包成为可执行文件
- 转换成为扩展模块 .pyd/.so

在我的眼里，你没有秘

在我的眼里，你没有秘密

我

我缺少安全感

- decompyle
- uncompyle

- PyInstaller
- py2exe
- Nuitka
- Cython

PyArmor 功能特点

- 无缝替换
- 动态加密
- 设置加密脚本的许可方式

无缝替换

- foo.py

```
print('Hello PyCon 2020')
```

无缝替换

- dist/foo.py
- dist/py   sform.so

我，依然是我

```
from pytransform import pyarmor
pyarmor(__name__, __file__, b'\x50\x59\x41\x52\x4d\x4f\x52\x00\
\x41\xf9\xa3\x0d\xb3\x55\x80\x05\x2c\x17\xc4\x40\xf8', 2)
```

动态加密

```python
# write Fibonacci series up to n
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

动态加密

```
# 加密后的等价形式

def fib(n):
    __armor_enter__()
    try:
        # 这里是加密后的代码
        xxxxxxx
    finally:
        __armor_exit()
```

我有一种被拥抱的感觉

设置加密脚本许可方式

- 设置使用期限
- 设置允许运行的设备
- 扩展其他认证方式

好暖心哦！

应用范围

- 加密 PyQt 等桌面应用
- 加密OpenCV/Numpy的嵌入式设备
- 加密 Flask/Django等Web框架应用
- 加密云服务器和Docker

- X86/64
- ARM
- PPC
- MIPS

- RaspBerry Pi
- Banana Pi
- Orange Pi
- Android

- Windows
- Mac
- Linux
- FreeBSD

PyCon China 2020

Python For Good

PyConChina 2020 | PYTHON 中国开发者大会 2020

PyArmor 基本用法

安装和启动

```
pip install pyarmor

pyarmor
```

```
(venv) bogon:demo jondy$ pip install pyarmor
Requirement already satisfied: pyarmor in ./venv/lib/python3.7/site-packages (6.5.4)
(venv) bogon:demo jondy$ pyarmor
usage: pyarmor [-h] [-v] [-q] [-d] [--home HOME] [--boot BOOT]  ...
pyarmor: error: too few arguments
(venv) bogon:demo jondy$ 
```

加密脚本

- 子命令 obfuscate

```
pyarmor obfuscate foo.py
```

demo — -bash — 80×24

```
fibo.py                                 pytransform.cpython-37m-darwin.so
foo.py
(venv) bogon:demo jondy$ cat foo.py
print('Hello PyCon 2020')
(venv) bogon:demo jondy$ cat dist/foo.py
from pytransform import pyarmor
pyarmor(__name__, __file__, b'\x50\x59\x41\x52\x4d\x4f\x52\x00\x00\x03\x07\x00\x
42\x0d\x0d\x0a\x04\x00\x00\x01\x00\x00\x01\x00\x00\x40\x00\x00\x00\x
aa\x00\x00\x00\x0b\x00\x00\x18\xb5\x8e\x08\x75\x26\xfd\xbc\xe0\x92\xca\xdd\x86\x
3c\x57\x06\xa3\x00\x00\x00\x00\x00\x00\x00\x00\x00\x4c\x08\xe9\x24\x5b\xb7\x95\x
09\x7e\x01\x19\x06\x61\x6b\x12\x30\x2c\xce\xdd\x5d\x3d\x5d\x59\x33\x82\x13\x34\x
58\xd3\x3e\x04\x46\x97\xb2\x19\x82\x6a\x20\x76\x6c\x8f\x9c\xcc\x56\x4b\x2a\xd6\x
0d\x32\x2f\xf4\x11\x3e\xd6\x18\xc8\x49\xb9\x19\xba\xa0\x17\x81\x2e\x70\xd7\xb4\x
8c\xa3\x7b\xe4\x40\xea\xeb\x62\x53\xdb\xaa\x11\x11\x5c\x8f\xe0\x3a\xb5\x9c\x4a\x
7b\xba\x57\xa3\xb7\xb9\x57\xe4\xe4\x07\xd6\x9f\xcf\xf4\xb1\x20\x44\x7b\x0f\x00\x
a8\x68\x56\x45\x6d\xa1\x9a\x23\x92\x40\x46\x2c\x21\xac\xca\xa2\xcb\x0e\x3e\x94\x
4b\xfa\xc2\xbc\x32\xd8\x2f\x6f\x2f\xd3\x32\x9a\xec\x19\xcf\xc8\x31\x3f\x31\xbd\x
96\xd4\x5a\x06\xbe\xca\x75\xf7\x78\x69\xed\x11\x10\x34\x36\x6d\xac\x57\x0e\x90\x
5f\x89', 2)(venv) bogon:demo jondy$
(venv) bogon:demo jondy$ ls dist/pytransform.cpython-37m-darwin.so
dist/pytransform.cpython-37m-darwin.so
(venv) bogon:demo jondy$ python dist/foo.py
Hello PyCon 2020
(venv) bogon:demo jondy$
```

设置许可

- 子命令 licenses

- 设置有效期

```
pyarmor licenses -e 2020-12-31 r001
pyarmor obfuscate –with-license licenses/r001/license.lic foo.py
```

- 绑定到网卡

```
pyarmor hdinfo
pyarmor licenses –bind-mac “aa:00:a4:21:b9:01” r002
```

demo — -bash — 81×24

```
Ip address: "192.168.121.100"

Domain name: ""

Change logs

        v6.2.0(r21): Remove trailing dot from harddisk serial number
        v6.4.2(r34): Support binding multiple mac addresses
        v6.5.3(r37): Support binding named harddisk

(venv) bogon:demo jondy$ pyarmor l --bind-mac f8:ff:c2:27:00:7f r002
INFO     PyArmor Trial Version 6.5.4
INFO     Generate licenses with capsule /Users/jondy/.pyarmor/.pyarmor_capsule.zi
p ...
INFO     Output path of licenses: licenses
INFO     The license file is generated in restrict mode
INFO     The license file is generated in period mode disabled
INFO     Make path: licenses/r002
INFO     Generate license: *IFMAC:f8:ff:c2:27:00:7f*CODE:r002
INFO     Write license file: licenses/r002/license.lic
INFO     Write information to licenses/r002/license.lic.txt
INFO     Generate 1 licenses OK.
(venv) bogon:demo jondy$
```

加密打包脚本

- 子命令 pack

```
pyarmor pack foo.py
```

src — -bash — 81×24

```
4179 INFO: Building PKG (CArchive) PKG-00.pkg
6631 INFO: Building PKG (CArchive) PKG-00.pkg completed successfully.
6634 INFO: Bootloader /Users/jondy/Documents/pyarmor/pycon2020/demo/venv/lib/pyth
on3.7/site-packages/PyInstaller/bootloader/Darwin-64bit/run
6634 INFO: checking EXE
6634 INFO: Building EXE because EXE-00.toc is non existent
6634 INFO: Building EXE from EXE-00.toc
6634 INFO: Appending archive to EXE dist2/foo
6639 INFO: Fixing EXE for code signing dist2/foo
6642 INFO: Building EXE from EXE-00.toc completed successfully.
INFO     ==================== End command ====================

INFO     Remove .spec file foo.spec
INFO     Remove patched .spec file foo-patched.spec
INFO     Remove build path dist2/obf
INFO     Final output path: dist2
INFO     Pack obfuscated scripts successfully.
(venv) bogon:src jondy$ ls
build   dist    dist2   foo.py
(venv) bogon:src jondy$ ls dist2/
foo
(venv) bogon:src jondy$ dist2/foo
Hello PyCon 2020
(venv) bogon:src jondy$ 
```

图形界面

```
pip install pyarmor-webui

pyarmor-webui
```

Safari File Edit View History Bookmarks Develop Window Help
100% Tue 7:10 AM
localhost
Favorites
PyArmor
简体中文

PyArmor

首页

我的工程

我的许可证

关于

首页

加密脚本向导

使用工程加密

加密打包向导

使用工程打包

时间限制许可证

指定设备许可证

全部特征许可证

注册 PyArmor

PyCon China 2020

Python For Good

PyConChina 2020 | PYTHON 中国开发者大会 2020

PyArmor 加密原理

加密原理

```
pyarmor obfuscate foo.py
```

加密（1）— 编译

```
char *filename = "foo.py";
char *source = read_file( filename );
PyCodeObject *co = Py_CompileString( source, "<frozen foo>", Py_file_input );
```

Code Object

```
/* Bytecode object */
typedef struct {
    PyObject_HEAD
    int co_argcount;            /* #arguments, except *args */
    int co_kwonlyargcount;      /* #keyword only arguments */
    int co_nlocals;             /* #local variables */
    int co_stacksize;           /* #entries needed for evaluation stack */
    int co_flags;               /* CO_..., see below */
    int co_firstlineno;         /* first source line number */
    PyObject *co_code;          /* instruction opcodes */
    PyObject *co_consts;        /* list (constants used) */
    PyObject *co_names;         /* list of strings (names used) */
    PyObject *co_varnames;      /* tuple of strings (local variable names) */
    PyObject *co_freevars;      /* tuple of strings (free variable names) */
    PyObject *co_cellvars;      /* tuple of strings (cell variable names) */
    Py_ssize_t *co_cell2arg;    /* Maps cell vars which are arguments. */
    PyObject *co_filename;      /* unicode (where it was loaded from) */
    PyObject *co_name;          /* unicode (name, for reference) */
    PyObject *co_lnotab;        /* string (encoding addr<->lineno mapping) See
                                   Objects/lnotab_notes.txt for details. */

    ...
} PyCodeObject;
```

加密（1）—— 编译

```python
# fibo.py: Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

```
>>> co.co_names
('print',)
```

```
>>> co.co_consts
(None, (0, 1), ' ', ('end',))
```

```
>>> co.co_code
b'd\x01\\\x02}\x01}\x02x&|\x01|\x00k\x00r.t\
\x03\x8d\x02\x01\x00|\x02|\x01|\x02\x17\x00\
\x02q\nW\x00t\x00\x83\x00\x01\x00d\x00S\x00'
```

加密（2）— 函数加密

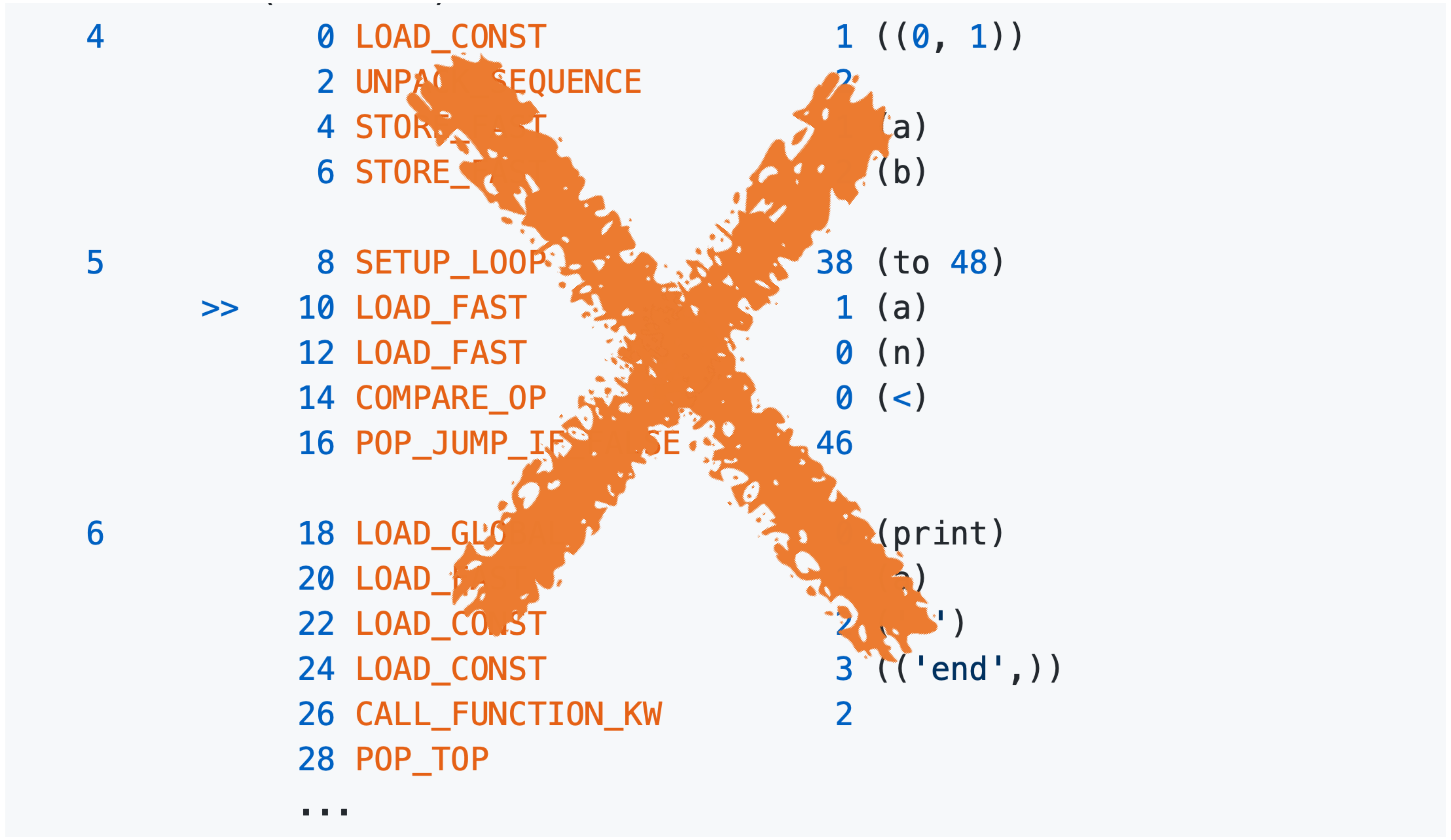
```c
static void
obfucate_code_object( PyCodeObject *co)
{
  // 1. 加密 co_code
  obfuscate_co_code( co->co_code );

  // 2. 添加自定义函数名称
  patch_co_names( co->co_names );

  // 3. 递归加密 Code Object
  for ( i = 0; i < PyTuple_Size( co->co_consts ); i++ ) {
    PyObject *pobj = PyTuple_GetItem( co->co_consts, i );
    if ( PyObject_TypeCheck( pobj, PPyCode_Type ) )
      obfuscate_co_code( pobj )
  }
}
```

加密（2）— 函数加密

```
>>> import dis
>>> dis.dis(co.co_code)
```

```
>>> co.co_code
b'd\x01\\\x02}\x01}\x02x&|\x01|\x00k\x00r.t\
\x03\x8d\x02\x01\x00|\x02|\x01|\x02\x17\x00\
\x02q\nW\x00t\x00\x83\x00\x01\x00d\x00S\x00'
```

加密（2）— 函数加密



```
  4           0 LOAD_CONST               1 ((0, 1))
              2 UNPACK_SEQUENCE          2
              4 STORE_FAST [illegible]     (a)
              6 STORE_FAST [illegible]     (b)

  5           8 SETUP_LOOP              38 (to 48)
        >>   10 LOAD_FAST                1 (a)
             12 LOAD_FAST                0 (n)
             14 COMPARE_OP               0 (<)
             16 POP_JUMP_IF_FALSE       46

  6          18 LOAD_GLOBAL [illegible]    (print)
             20 LOAD_FAST [illegible]      (a)
             22 LOAD_CONST               2 (' ')
             24 LOAD_CONST               3 (('end',))
             26 CALL_FUNCTION_KW         2
             28 POP_TOP
             ...
```

加密（2）— 函数加密

```
      0 LOAD_GLOBALS     10 '__armor_enter__'
      2 CALL_FUNCTION    0
      4 POP_TOP
      6 SETUP_FINALLY    280

         加密后的代码
         ...

=>      280 LOAD_GLOBALS     1 '__armor_exit_
        282 CALL_FUNCTION    0
        284 POP_TOP
        286 END_FINALLY
```

```
__armor_enter__()

try:

    加密后的代码
    ...

finally:

    __armor_exit__()
```

加密（2）— 函数加密

```
static void
obfucate_code_object( PyCodeObject *co)
{
  // 1. 加密 co_code
  obfuscate_co_code( co->co_code );

  // 2. 添加自定义函数名称
  patch_co_names( co->co_names );

  // 3. 递归加密 Code Object
  for ( i = 0; i < PyTuple_Size( co->co_consts ); i++ ) {
    PyObject *pobj = PyTuple_GetItem( co->co_consts, i );
    if ( PyObject_TypeCheck( pobj, PPyCode_Type ) )
      obfuscate_co_code( pobj )
  }
}
```

加密（2）— 函数加密

```
>>> co.co_names
('print',)
```

加密（2）— 函数加密

```
>>> co.co_names
('print', '__armor_enter__', '__armor_exit__')
```

加密（2）— 函数加密

```c
static void
obfucate_code_object( PyCodeObject *co)
{
  // 1. 加密 co_code
  obfuscate_co_code( co->co_code );

  // 2. 添加自定义函数名称
  patch_co_names( co->co_names );

  // 3. 递归加密 Code Object
  for ( i = 0; i < PyTuple_Size( co->co_consts ); i++ ) {
    PyObject *pobj = PyTuple_GetItem( co->co_consts, i );
    if ( PyObject_TypeCheck( pobj, PPyCode_Type ) )
      obfuscate_co_code( pobj )
  }
}
```

加密（3）— 模块加密

```
char *string_code = marshal.dumps( co );
char *obfuscated_code = obfuscate_algorithm( string_code  );
```

加密（4）— 输出

```
sprintf( buf, "from pytransfrom import pyarmor\n"
              "pyarmor(__name__, __file__, b'%s', 2)", obfuscated_code );
save_file( "dist/foo.py", buf );
```

加密（4）— 输出

```
from pytransform import pyarmor
pyarmor(__name__, __file__, b'\x50\x59\x41\x52\x4d\x4f\x52\x00\
\x41\xf9\xa3\x0d\xb3\x55\x80\x05\x2c\x17\xc4\x40\xf8', 2)
```

运行加密脚本

```
python dist/foo.py
```

运行（1）

```
from pytransform import pyarmor
pyarmor(__name__, __file__, b'\x50\x59\x41\x52\x4d\x4f\x52\x00\
\x41\xf9\xa3\x0d\xb3\x55\x80\x05\x2c\x17\xc4\x40\xf8', 2)
```

- 检查加密脚本许可证
- 增加内置函数 __armor_enter, __armor_exit__

内置函数

```
static PyObject * __armor_enter__(PyObject *self, PyObject *args);
static PyObject * __armor_exit__(PyObject *self, PyObject *args);

static PyMethodDef enter_method =
  {
   "__armor_enter__",
   __armor_enter__,
   METH_NOARGS,
   NULL
  };

PyObject *cfunc = PyCFunction_NewEx( &enter_method, NULL, NULL )
PyObject *builtins = PyEval_GetBuiltins();
PyDict_SetItem( builtins, "__armor_enter__", cfunc );
```

运行（1）

```
from pytransform import pyarmor
pyarmor(__name__, __file__, b'\x50\x59\x41\x52\x4d\x4f\x52\x00\
\x41\xf9\xa3\x0d\xb3\x55\x80\x05\x2c\x17\xc4\x40\xf8', 2)
```

运行（2）

```
static PyObject *
pyarmor(char *name, char *pathname, unsigned char *obfuscated_code)
{
    char *string_code = restore_obfuscated_code( obfuscated_code );
    PyCodeObject *co = marshal.loads( string_code );
    return PyImport_ExecCodeModuleEx( name, co, pathname );
}
```

加密函数

```
    0 LOAD_GLOBALS     10 '__armor_enter__'
    2 CALL_FUNCTION    0
    4 POP_TOP
    6 SETUP_FINALLY    280

    try:

        加密后的代码
        ...

    finally:

=>   280 LOAD_GLOBALS     1 '__armor_exit__'
     282 CALL_FUNCTION    0
     284 POP_TOP
     286 END_FINALLY
```

运行（3）

```
static PyObject *
__armor_enter__(PyObject *self, PyObject *args)
{
    // Got code object
    PyFrameObject *frame = PyEval_GetFrame();
    PyCodeObject *f_code = frame->f_code;

    // Restore byte code if it's obfuscated
    if (IS_OBFUSCATED(f_code->co_flags)) {
        restore_byte_code(f_code->co_code);
        clear_obfuscated_flag(f_code);
    }

    Py_RETURN_NONE;
}
```

加密函数

```
      0 LOAD_GLOBALS     10 '__armor_enter__'
      2 CALL_FUNCTION    0
      4 POP_TOP
      6 SETUP_FINALLY    280

      try:

          加密后的代码
          ...

      finally:

=>       280 LOAD_GLOBALS     1 '__armor_exit__'
         282 CALL_FUNCTION    0
         284 POP_TOP
         286 END_FINALLY
```

已经被恢复了！！！

运行（4）

```c
static PyObject *
__armor_exit__(PyObject *self, PyObject *args)
{
    // Got code object
    PyFrameObject *frame = PyEval_GetFrame();
    PyCodeObject *f_code = frame->f_code;

    // Obfuscate byte code again
    obfuscate_byte_code(f_code->co_code);
    set_obfuscated_flag(f_code);

    // Clear f_locals in this frame
    clear_frame_locals(frame);

    Py_RETURN_NONE;
}
```

加密函数

```
        0 LOAD_GLOBALS     10 '__armor_enter__'
        2 CALL_FUNCTION    0
        4 POP_TOP
        6 SETUP_FINALLY    280

        try:

            加密后的代码   重新被加密了！！！
            ...

        finally:

=>       280 LOAD_GLOBALS     1 '__armor_exit__'
         282 CALL_FUNCTION    0
         284 POP_TOP
         286 END_FINALLY
```

加密脚本性能

函数大小	调用次数	未加密	加密	差值（毫秒）
10 K	1	0.053000	0.119000	0.066
10 K	1,000	32.067000	42.164000	10.097
10 K	10,000	307.478000	407.585000	100.007

PyArmor 安全性

Python 层面的安全

- 使用 dis/inspect 等反编译模块
- 使用 pdb，sys.settrace 等动态跟踪
- 使用 Python C API
- 异常和 traceback 安全

动态库 层面的安全

- 反调试
- JIT 动态代码和虚拟指令 VM
- 交叉保护
- 不定期更新加密算法

兵无常势，水无常形

终极安全

?

终极安全

Python For Good
PyCon China 2020 | PYTHON 中国开发者大会 2020

公开加密算法

PyCon China 2020

THANK YOU

PyArmor 公众号