

.pyc文件格式学习笔记

简介

.pyc文件是.py文件通过python编译生成的一种更易于执行的文件格式，它可以通过python虚拟机执行。由于.pyc是python对.py源码进行一定符号处理之后得到的文件，相比于直接解释执行.py文件，执行.pyc文件的效率将会更高

.pyc文件的生成

可以通过以下代码将.py文件转换成一个.pyc文件

```
python -m py_compile [filename]
```

.pyc文件的结构

结构	字节数	解释
Magic number	4 B	用于标识生成.pyc文件的python版本信息
Unix timestamp	4 B	unix时间戳数值上等于1970.0.0 00:00到生成文件时刻的秒数
PyCodeObject	--	代码编译后的主要内容，为序列化的字节流

PyCodeObject 的内容

以下是code.h头文件中关于PyCodeObject的部分

```
/* code.h */
/* Bytecode object */
typedef struct {
    PyObject_HEAD
    int co_argcount;          /* #arguments, except *args */
    int co_nlocals;           /* #local variables */
    int co_stacksize;         /* #entries needed for evaluation */
    int co_flags;             /* CO_..., see below */
    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) */
    /* The rest doesn't count for hash/cmp */
    PyObject *co_filename;    /* string (where it was loaded) */
    PyObject *co_name;        /* string (name, for reference) */
    int co_firstlineno;       /* first source line number */
    PyObject *co_lnotab;      /* string (encoding addr<->lineno mapping) */
    void *co_zombieframe;     /* for optimization only (see Notes) */
    PyObject *co_weakreflist; /* to support weakrefs to code objects */
} PyCodeObject;
```

解释如下

- co_nlocals : Code Block中局部变量个数，包括其位置参数个数
- co_stacksize : 执行该段Code Block需要的栈空间
- co_code : Code Block编译得到的字节码指令序列，以PyStringObject的形式存在
- co_consts: PyTupleObject，保存Code Block中的所有常量
- co_names: PyTupleObject，保存Code Block中的所有符号
- co_varnames: Code Block中的局部变量名集合
- co_freevars : Python实现闭包存储内容

- `co_cellvars` : Code Block中内部嵌套函数所引用的局部变量名集合
- `co_filename` : Code Block对应的.py文件的完整路径
- `co_name` : Code Block的名字, 通常是函数名或类名

.pyc文件的反编译

虽然.pyc看似隐藏了python的代码, 实际上.pyc文件的反编译是十分容易的。

我们可以通过一个叫做uncompyle2的工具很容易实现.pyc的反编译

以下(linux环境下需使用root身份执行)使用pip安装这个工具:

```
pip install uncompyle2
```

以下测试uncompyle2:

先写一个简单的python程序:

```
# test.py
import base64
key1 = 'hello123 '
key2 = 'uncompile me'
key3 = base64.b64encode(key1+key2)
print (key3)
```

然后生成.pyc文件

```
python -m py_compile test.py
```

之后我们再通过uncompyle2工具反编译它

```
uncompyle6 test.pyc
```

运行结果与我们所写的程序是一样的

除了使用uncompyle2反编译.pyc文件，我们还可以使用marshal和dis两个模块来查看.pyc文件字节流对应的内容

我们通过以下程序来反编译test.pyc程序

```
# dis_py.py
import dis, marshal, struct, sys, time, types

def show_file(fname):
    f = open(fname, "rb")
    magic = f.read(4)
    moddate = f.read(4)
    modtime = time.asctime(time.localtime(struct.unpack('L',
    print "magic %s" % (magic.encode('hex'))
    print "moddate %s (%s)" % (moddate.encode('hex'), modtime)
    code = marshal.load(f)
    show_code(code)

def show_code(code, indent=''):
    print "%scode" % indent
    indent += '    '
    print "%sargcount %d" % (indent, code.co_argcount)
    print "%snlocals %d" % (indent, code.co_nlocals)
    print "%sstacksize %d" % (indent, code.co_stacksize)
    print "%sflags %04x" % (indent, code.co_flags)
    show_hex("code", code.co_code, indent=indent)
    dis.disassemble(code)
    print "%sconsts" % indent
    for const in code.co_consts:
        if type(const) == types.CodeType:
            show_code(const, indent+'    ')
        else:
```

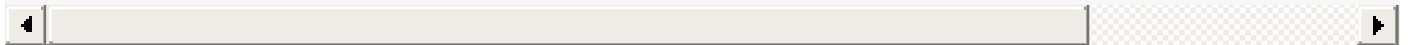
```

        print "    %s\r" % (indent, const)
    print "%snames %r" % (indent, code.co_names)
    print "%svarnames %r" % (indent, code.co_varnames)
    print "%sfreevars %r" % (indent, code.co_freevars)
    print "%scellvars %r" % (indent, code.co_cellvars)
    print "%sfilename %r" % (indent, code.co_filename)
    print "%sname %r" % (indent, code.co_name)
    print "%sfirstlineno %d" % (indent, code.co_firstlineno)
    show_hex("\n\t\t", code.co_lnotab, indent=indent)

def show_hex(label, h, indent):
    h = h.encode('hex')
    if len(h) < 60:
        print "%s%s %s" % (indent, label, h)
    else:
        print "%s%s" % (indent, label)
        for i in range(0, len(h), 60):
            print "%s    %s" % (indent, h[i:i+60])

show_file(sys.argv[1])

```



命令行下执行

```
python dis_py test.pyc
```

得到如下结果

```

magic 03f30d0a
moddate 125bd758 (Sun Mar 26 14:09:22 2017)
code
    argcount 0
    nlocals 0
    stacksize 3
    flags 0040
    code

```

```
6400006401006c00005a00006402005a01006403005a0200650000
650100650200178301005a0400650400474864010053
```

1	0	LOAD_CONST	0	(-1)
	3	LOAD_CONST	1	(None)
	6	IMPORT_NAME	0	(base64)
	9	STORE_NAME	0	(base64)
2	12	LOAD_CONST	2	('hello123 ')
	15	STORE_NAME	1	(key1)
3	18	LOAD_CONST	3	('uncompile me')
	21	STORE_NAME	2	(key2)
4	24	LOAD_NAME	0	(base64)
	27	LOAD_ATTR	3	(b64encode)
	30	LOAD_NAME	1	(key1)
	33	LOAD_NAME	2	(key2)
	36	BINARY_ADD		
	37	CALL_FUNCTION	1	
	40	STORE_NAME	4	(key3)
5	43	LOAD_NAME	4	(key3)
	46	PRINT_ITEM		
	47	PRINT_NEWLINE		
	48	LOAD_CONST	1	(None)
	51	RETURN_VALUE		

consts

-1

None

'hello123 '

'uncompile me'

names ('base64', 'key1', 'key2', 'b64encode', 'key3')

varnames ()

freevars ()

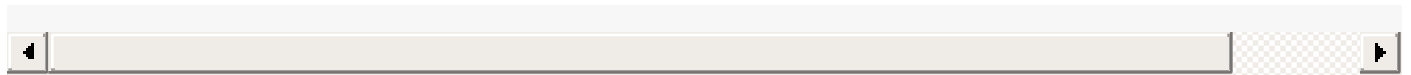
cellvars ()

filename 'test.py'

name '<module>'

firstlineno 1

lnotab 0c01060106011301



其中的LOAD_CONST这些指令实际上是python内部与opcode所对应的指令，通过结果我们可以大致还原出程序的原貌

资料

以下是一道用到.pyc文件结构的逆向题的题目链接:

http://dl.0ops.net/py_d5764c66f02cccdb356c532d60d4d079.zip

以下是这道题的writeup:

<https://0xd13a.github.io/ctfs/0ctf2017/py/>