py2exe原理剖析

安全研究部 陈庆

py2exe 可以将 Python 代码打包成独立可执行的 EXE，有助于向未安装 Python 运行环境的用户分发某些小工具。某些恶意软件用 Python 开发，然后用 py2exe 打包成 EXE 并传播。本文从逆向工程的角度介绍 py2exe 的实现原理，剖析其生成的 EXE 格式。

一、解析 EXE

假设有 rom0scan.py，经 py2exe 处理得到 rom0scan.exe。

py2exe 生成的 EXE 主要由三部分组成。

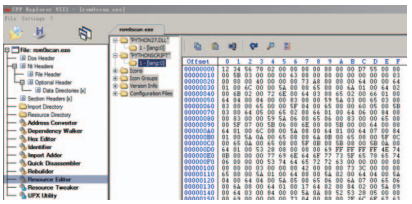
第一部分是名为 "PYTHON27.DLL" 的资源，这里存放的就是 python27.dll。我们只考虑 py2exe 参数 bundle_files 设为 1 的情形，此时 Python 解释器内嵌在 EXE 中。

第二部分是名为 "PYTHONSCRIPT" 的资源，这里存放序列化之后的 rom0scan.pyc 或 pyo。

第三部分是名为 library.zip 的数据，它简单地附加在 EXE 尾部 (overlay)。这里面包含 rom0scan.py 所依赖的各种库文件。py2exe 参数 zipfile 可以更改这个行为，如果不为 None，将在文件系统中生成指定名字的压缩包，此时没有 overlay, 可以在 zipfile 中指定相对路径。

反编译 py2exe 生成的 EXE，基本不关心第一部分、第三部分，主要关心第二部分。

用你喜欢的资源编辑工具打开 rom0scan.exe，即可看到名为 "PYTHONSCRIPT" 的资源，将它析取保存成名为 "PYTHONSCRIPT" 的文件。



二、解析 PYTHONSCRIPT

PYTHONSCRIPT 是个带格式的二进制文件，不直接对应 .pyc、.pyo。它有一个不定长的首部。

```
$ xxd -l 64 -g 1 PYTHONSCRIPT
0000000: 12 34 56 78 02 00 00 00 00 00 00 00 cf 55 00 00  .4Vx.........U..
0000010: 00 5b 03 00 00 00 63 00 00 00 00 00 00 00 00 03  .[....c.........
0000020: 00 00 00 40 00 00 00 73 a8 00 00 00 64 00 00 64  ...@...s....d..d
0000030: 01 00 6c 00 00 5a 00 00 65 00 00 6a 01 00 64 02  ..l..Z..e..j..d.
```

首部数据结构如下：

```
struct Header
{
    /*
    * 0x78563412
    */
    unsigned int    tag;
    /*
    * 2
    *
    * 对应 py2exe 参数 optimize
    */
    unsigned int    optimize;
    /*
    * 0
    *
    * 不知啥意思
    */
    unsigned int    unbuffered;
    /*
    * 首部之后的数据长度 (little-endian)，0x55CF
    *
    * 实际查看 PYTHONSCRIPT，首部长度加上 data_bytes 还没到尾，最后还有两个
    * 0x00，不知啥情况。
    */
    unsigned int    data_bytes;
    /*
    * 对应 py2exe 参数 zipfile，以 NUL 字符结尾。如果 zipfile 为 None，此处只有一
    * 个 \0，此时 PYTHONSCRIPT 占 0x11(17) 字节。
    */
    unsigned char   zippath[VARIABLE_SIZE]
};
```

PYTHONSCRIPT 首部之后是各个序列化过后的 .pyc、.pyo。

三、GetPyc.py

编辑 GetPyc.py 如下：

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-
#
# Note that you have to run the script in the same version of python which
# was used to generate the exe. Otherwise unmarshalling will fail.
#
import marshal, imp
```

```
f   = open( 'PYTHONSCRIPT', 'rb' )
#
# struct Header
# {
#     unsigned int    tag;
#     unsigned int    optimize;
#     unsigned int    unbuffered;
#     unsigned int    data_bytes;
#     unsigned char   zippath[VARIABLE_SIZE]
# };
#
# Skip the header, you have to know the header size beforehand.
#
f.seek( 0x11 )
ob  = marshal.load( f )
for i in xrange( 0, len( ob ) ) :
    open( str( i ) + '.pyc', 'wb' ).write( imp.get_magic() + '\0' * 4 +
marshal.dumps( ob[i] ) )
f.close()
```

GetPyc.py从PYTHONSCRIPT中析取、反序列化出各个.pyc、.pyo。

```
$ xxd -l 16 -g 1 rom0scan.pyc
0000000: 03 f3 0d 0a 09 14 37 55 63 00 00 00 00 00 00 00
.....7Uc.....
```

.pyc文件前面一般有8字节的首部：

```
03 f3 0d 0a // +0x00 magic
09 14 37 55 // +0x04 timestamp, little-endian
```

前4字节是用于识别.pyc文件的magic数据，后4字节是时间戳。py2exe在序列化.pyc文件时，将这8字节首部剥掉了。GetPyc.py在反序列化时补回8字节首部，前4字节通过imp.get_magic()获得，后4字节简单地用\0填充。

四、exe2py2.py

GetPyc.py还是太简单，exe2pyc.py要方便很多。

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-
import os, sys, os, struct, time, marshal, imp, dis
import win32api
def exe2pyc ( exefile ) :
    handle  = win32api.LoadLibrary( exefile )
    data = win32api.LoadResource( handle, "PYTHONSCRIPT",
1, 0 )
    tag, optimize, unbuffered, data_bytes   \
    = struct.unpack( "<IIII", data[:4*4] )
    data      = data[4*4:]
    zippath, data   \
```

```
            = data.split( "\0", 1 )
print                       \
"tag        : 0x%08X\n"    \
"optimize   : %u\n"        \
"unbuffered  : %u\n"       \
 "data_bytes  : 0x%08X\n"    \
"zippath    : [%s]"        \
%                          \
 (
tag,
optimize,
unbuffered,
data_bytes,
zippath
)
codeobj = marshal.loads( data )
i       = 0
for co in codeobj :
    #
    # co.co_filename 有可能带路径
    #
    fname       = os.path.basename( co.co_filename )
    #
```

```
    # The resource contains a sequence of marshaled code
objects, not
    # all of them come from a .py file, some are constructed
from
    # source at build time.
    #
    if not os.path.splitext( fname )[1] :
      fname   = "on_the_fly_%u.py" % i
    if optimize :
      fname   = fname + 'o'
    else :
      fname   = fname + 'c'
    #
    # little-endian
    #
    timestamp   = struct.pack( "<I", int( time.time() ) )
    #
    # 在 .pyc、pyo 首部写入时间戳没有实际意义, 完全可以用 \0
填充。
    #
    open( fname, "wb" ).write( imp.get_magic() + timestamp
+ marshal.dumps( co ) )
    print "[%u] - [%s] => [%s]" % ( i, co.co_filename, fname )
```

```
            if "on_the_fly" in fname :
            dis.dis( co )
             i     += 1
        #
        # end of for
        #
if "__main__" == __name__ :
        exe2pyc( sys.argv[1] )
```

```
        $ exe2pyc.py rom0scan.exe
```

```
tag         : 0x78563412
optimize    : 2
unbuffered  : 0
data_bytes  : 0x000055CF
zippath     : []
[0] - [C:\Python27\lib\site-packages\py2exe\boot_common.py]
=> [boot_common.pyo]
[1] - [<install zipextimporter>] => [on_the_fly_1.pyo]
        1           0 LOAD_CONST               0 (-1)
                    3 LOAD_CONST               1 (None)
                    6 IMPORT_NAME              0 (zipextimporter)
                    9 STORE_NAME               0 (zipextimporter)
                   12 LOAD_NAME                0 (zipextimporter)
                   15 LOAD_ATTR                1 (install)
                   18 CALL_FUNCTION            0
                   21 POP_TOP
                   22 LOAD_CONST               1 (None)
                   25 RETURN_VALUE
[2] - [rom0scan.py] => [rom0scan.pyo]
```

就 rom0scan.exe 而言，exe2pyc.py 总共析取、反序列化出 3 个 .pyo。

第一个是 boot_common.pyo，来自：

C:\Python27\lib\site-packages\py2exe\boot_common.py

第二个是 on_the_fly_1.pyo，来自 "<install zipextimporter>"，它是 py2exe 的中间生成物：

```
import zipextimporter
zipextimporter.install()
```

对应：

C:\Python27\lib\site-packages\zipextimporter.py

这里允许从 .zip 文件中直接加载 Python 模块，而不必将 .zip 解压缩到文件系统。zipextimporter.install() 完成 "import hook"。

第三个是 rom0scan.pyo，来自 rom0scan.py。

得到 .pyc、.pyo 之后，用你喜欢的 Python 反编译工具处理它，得到 Python 源代码。在不考虑 Python 源代码混淆技术介入的前提下，uncompyle2 是个不错的 Python 反编译工具。