

# Extracting Python Source Code from Executable

---

Author: Sage

Origin: Nullcon's HackIM 2015

Challenge: Reversing 100

Website: <http://www.thealmightypaper.com>

The first thing I did was run the file command. Subsequently, I ran the strings commands on the executable. The file command's output indicated that the file needed to be run on Mac OS X. The strings command provided no output. By default strings only checks the data section of the executable; so I reran the command this time looking in all sections and found that it was packed with UPX. The file is packed so there are a lot of useless strings such as "IJKLMNOPQ:TUVWXYZ%\$~" in the file. I have included only the UPX related strings for brevity.

```
$ file boo
boo: Mach-O 64-bit executable x86_64
```

*Figure A.1. The file type is a 64-bit Mach-O file*

```
$ strings -a boo | fgrep -i upx
UPX!
UPX!
$Info: This file is packed with the UPX executable packer http://upx.sf.net $
$Id: UPX 3.91 Copyright (C) 1996-2013 the UPX Team. All Rights Reserved. $
UPX!u
UPX!
```

*Figure A.2. UPX packer related strings exist outside the data section*

Aside from UPX another odd string that was pulled out was the word, vpython. I have included the relevant output in the below (See figure B.1). Search engine results reveal that vpython is a

3D programming library for the Python programming language. There are two possibilities that can be concluded from this:

1. The word vpython could be an artifact placed in the executable to send us off track.
2. Perhaps the strings utility pulled out the word vpython simply because the neighboring binary data's ASCII equivalent happened to be a letter v so it says vpython rather than python.

In either case we will need to dive deeper into this python business because one thing is for certain, the substring python in the word vpython was placed in the executable purposely. We will revisit this later.

```
$ strings -a boo | fgrep python  
  
vpython
```

*Figure B.1. Shows the string vpython present in the strings output*

First things first we definitely know it is packed with UPX version 3.91 so I used the Homebrew Package Manager to install UPX. I also confirmed that the version of UPX was equal to or greater than the version used to pack the executable; in this case it was equal too (See figure C.1).

UPX is common packer and it comes with a built-in unpacking capability as well. The command-line utility can sometimes unpack a packed executable; but when that is not possible it has to be unpacked manually. In this case I was unable to unpack the executable (See figure C.2).

```
$ upx --version  
upx 3.91  
UCL data compression library 1.03  
LZMA SDK version 4.65  
Copyright (C) 1996-2013 Markus Franz Xavier Johannes Oberhumer  
Copyright (C) 1996-2013 Laszlo Molnar  
Copyright (C) 2000-2013 John F. Reiser  
Copyright (C) 2002-2013 Jens Medoch  
Copyright (C) 1999-2006 Igor Pavlov  
UPX comes with ABSOLUTELY NO WARRANTY; for details type 'upx -L'.
```

*Figure C.1. UPX version equal to the one that packed the executable*

```

$ upx -d boo

Ultimate Packer for eXecutables

Copyright (C) 1996 - 2013

UPX 3.91      Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

File size      Ratio      Format      Name
-----
upx: boo: NotPackedException: not packed by UPX

Unpacked 0 files.

```

*Figure C.2. UPX unable to unpack executable*

At this point I knew the labor intensive art of manual unpacking was required. I read the program into a demo version of Hopper and did a search for a popad expecting to see a jmp opcode following it, but that didn't pan out because there were no popad opcodes in the packer code. I decided to hold off before pursuing this avenue any further; because static code analysis for finding the real program's OEP (Original Entry Point) is time-consuming.

Following up on the vpython lead from earlier I wanted to corroborate the findings of the strings utility. I was able to do this by using xxd and fgrep in tandem (See figure D.1).

```

$ xxd boo | fgrep vpython

000ac40: b776 2507 9076 7079 7468 6f6e 002f 7072  .v%..vpython./pr

```

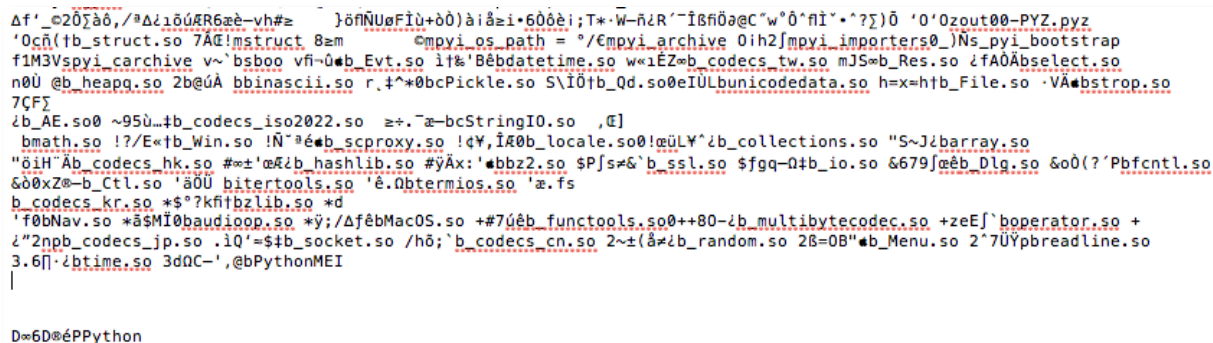
*Figure D.1. UPX unable to unpack executable*

If that is there then that begs the question, "What else might be in the file?" In my experience strings does not always find all strings reliably so I opened the program up in Mac OS X's TextEdit utility; a notepad-like program. Much of the program is incomprehensible but the useful information I did find is as follows:

- The file ends with the word Python.
- A little before the ending there is mention of pickle.so (which is used by Python when doing object serialization).

- Even further back you can see the string, out00-PYZ.pyz. (The file extension .pyz are python zipped executables).
- Between the pickle.so and .pyz find lie a slew of strings reminiscent of Python namely mpyi\_os\_path, mpyi\_archive, mpyi\_importers, pyi\_bootstrap, and pyi\_carchive.

Again, the letter m that precedes many of these strings could be the coincidental ASCII equivalent of binary data once again. However, coupled with our other supporting data in bullet points it is safe to say their is weight in the argument that this executable is a Python executable. Python has always had the ability to create pure machine code executables just like other popular languages such as Perl. To see what I saw look at the figure E.1 below.



```

Δf' _020jāō, /ΔΔ1ōūER6æè-vh#æ }ōñNūgFiū+ōō)āiāæi+6ōōèi;T*·W-ñzR'~īBñōā@C"w"ō^ñī*^?Σ)ō '0'Ozout00-PYZ.pyz
'0cñ(fb_struct.so 7Aē!mstruct 8æm 0mpyi_os_path = °/ēmpyi_archive 0ih2jmpyi_importers0_)Ns_pyi_bootstrap
f1M3Vspyi_carchive v~'bsboo vfñ~ū#b_Evt.so if%Bēbdatetime.so w«1ēZ«b_codecs_tw.so mJS«b_Res.so ifA0Abselect.so
n0ū @b_heapq.so 2b@ūā bbinascii.so r,†*#0bcPickle.so S\I0tb_Qd.so0eIūLbunicodedata.so h=x=htb_File.so ·VÅ#bstrop.so
7ÇFj
zb_AE.so0 ~95ū..tb_codecs_iso2022.so æ+.~æ-bcStringIO.so ,@]
bmash.so !/?E«tb_Win.so !N~*ē#b_scproxy.so !dY,īR0b_locale.so0!æūLY'zb_collections.so "S~Jzibarray.so
"ōih"Ab_codecs hk.so #«±'æRzb_hashlib.so #yÅx:'#bbz2.so $Pjs«&'b_ssl.so $fgq-Qtb_io.so &679fæb_Dlg.so &00(?)'Pbfentl.so
&0xZ«-b_Ctl.so 'ā0ū bitertools.so 'ē.Qbtermios.so 'æ.fs
b_codecs_kr.so *$"?kñtbzlib.so *d
'f0bNav.so *ā$Mī0baudioop.so *y;/ΔfēbMacOS.so +#7ūēb_func tools.so0++80-zb_multibytecodec.so +zeEj`boperator.so +
z'2npb_codecs_jp.so .iQ'=$zb_socket.so /hō;`b_codecs_cn.so 2~±{ā#zb_random.so 2B=0B"#b_Menu.so 2'7UŸpheadline.so
3.6[]·zbtime.so 3dQc-',@bPythonMEI
|
D«6D«ēPPython

```

Figure E.1. The end of the boo executable file

My Google search results for mpyi\_os\_path seems to indicate that Pyinstaller uses this string. So now we know the following:

- We are reversing a Python script that has been converted into an executable by PyInstaller.
- It uses the UPX 3.91 packer.

The PyInstaller manual says UPX output is an optional feature. After further reading through the manual I see that the original Python script is compiled to byte code and embedded into the PyInstaller executable. So the solution is simple, the code simply needs to be extracted. I attempted to use PyInstaller Extractor and an error appeared but it still seemed to get the extraction job done nonetheless (See figure F.1).

```
$ python pyinstxtractor.py boo

Traceback (most recent call last):

  File "pyinstxtractor.py", line 89, in <module>

    (thisTOCLen,)=struct.unpack('!i',exeFile.read(4))

struct.error: unpack requires a string argument of length 4

$ ls
_pyi_bootstrap  boo  out00-PYZ.pyz_extracted  pyi_carchive  pyi_os_path  struct
_struct.so  out00-PYZ.pyz  pyi_archive  pyi_importers  pyinstxtractor.py

$ cat boo

import sys

if len(sys.argv) == 13:

    print "Great: flag{g3771ng_st4rt3d_2015}"

else:

    print "."
```

*Figure F.1. PyInstaller Extractor extracting the python script and cat of boo's source code*

So as you can see from figure F.1 the pyz file was extracted and with it the decompiled original source code, or at least a very close facsimile.

Now that we have the code we can see the most expedient way to get the flag would have been to simply pass the boo program a series of arguments until the argument list was 13 arguments long (See figure G.1).

```
$ ./boo 1 2 3 4 5 6 7 8 9 10 11 12  
Great: flag{g3771ng_st4rt3d_2015}
```

*Figure G.1. Demonstrating the easy way to get the flag*

We did it the hard way, oh well...at least we got the flag.

I included some links below in case you're interested in further reading about these things. If you see a mistake, have a comment, or want to leave some feedback contact me at [srrqonpx@gurnyztuglcncre.pbz](mailto:srrqonpx@gurnyztuglcncre.pbz); you will need to rot13 this email address first. The email address is obfuscated to avoid email harvesters.

### **Useful Links & Further Reading**

<http://www.thegeekstuff.com/2010/11/strings-command-examples/>

<https://github.com/pyinstaller/pyinstaller/wiki>

<http://pythonhosted.org/PyInstaller/>

[http://nullege.com/codes/show/src%40p%40y%40PyInstaller-2.1%40PyInstaller%40loader%40pyi\\_importers.py/25/pyi\\_archive.ZlibArchive/python](http://nullege.com/codes/show/src%40p%40y%40PyInstaller-2.1%40PyInstaller%40loader%40pyi_importers.py/25/pyi_archive.ZlibArchive/python)

<http://bugs.python.org/issue14447>