SECTALKS
CTF 0x05 Solution

booto

June 25, 2014

Start

We have two executable files, each the same thing just built for different platforms. A quick look at `strings(1)` output for either shows a heap of references to functions from Python's native API.

Unbundle

There are bundlers that let you distribute a Python application as a native executable without any dependencies. A bit of googling turns up a few, the most likely of which is PyInstaller – it can target both Windows and Mac OS X. PyInstaller actually ships with an archive viewing and extraction tool, here's example output of running it against the provided Windows binary:

```
pos, length, uncompressed, iscompressed, type, name
[(0, 1529172, 1529172, 0, 'z', 'out00-PYZ.pyz'),
 (1529172, 169, 234, 1, 'm', 'struct'),
 (1529341, 1138, 2578, 1, 'm', 'pyi_os_path'),
 (1530479, 4779, 12596, 1, 'm', 'pyi_archive'),
 (1535258, 3960, 13376, 1, 'm', 'pyi_importers'),
 (1539218, 1800, 4228, 1, 's', '_pyi_bootstrap'),
 (1541018, 4173, 13142, 1, 's', 'pyi_carchive'),
 (1545191, 4545, 8984, 1, 's', 'balls'),
 (1549736, 605, 1859, 1, 'b', 'Microsoft.VC90.CRT.manifest'),
 (1550341, 286837, 627200, 1, 'b', 'msvcr90.dll'),
 (1837178, 218176, 851456, 1, 'b', 'msvcp90.dll'),
 (2055354, 70667, 245248, 1, 'b', 'msvcm90.dll'),
 (2126021, 1255863, 3004416, 1, 'b', 'python27.dll'),
 (3381884, 11424, 27648, 1, 'b', 'win32pipe.pyd'),
 (3393308, 5052, 10752, 1, 'b', 'select.pyd'),
 (3398360, 257865, 689664, 1, 'b', 'unicodedata.pyd'),
 (3656225, 12128, 29696, 1, 'b', 'win32wnet.pyd'),
 (3668353, 204238, 474624, 1, 'b', '_hashlib.pyd'),
 (3872591, 38925, 80896, 1, 'b', 'bz2.pyd'),
 (3911516, 534296, 1211904, 1, 'b', '_ssl.pyd'),
 (4445812, 45361, 111616, 1, 'b', '_ctypes.pyd'),
 (4491173, 64446, 164352, 1, 'b', 'pyexpat.pyd'),
 (4555619, 48855, 130048, 1, 'b', 'win32api.pyd'),
 (4604474, 20351, 47616, 1, 'b', '_socket.pyd'),
 (4624825, 49391, 137728, 1, 'b', 'pywintypes27.dll'),
 (4674216, 263, 476, 1, 'b', 'balls.exe.manifest')]
```

We see a whole heap of Python libraries, Windows DLLs and other junk... and one strange file called *balls*.

After extracting this file and inspecting it, you can see that it's a Python script:

```
#!/usr/bin/env python

# WICKED IMPORTS
from collections import OrderedDict
import random
import os
import sys
import requests
import time
import hashlib
import base64

# SWEET BASE64

data = 'ZnJvbSBjb2xsZWN0aW9ucyBpbXB ...
# EXEC THE MOTHERFUCKER
exec(base64.decodestring(data))
```

Use whatever method you like to decode the Base64 string, and we get another Python script.

Main Application

The main application simulates a connection to a mainframe. There are a few tricks to prevent the user from getting to the main interesting part of the app, but because we have the source, we can just skip it.

It calculates the MD5 hash of the calling executable. It then iterates through the current directory looking for a file with the same MD5 hash as the calling executable, but with a different filename. If it finds a file that fits the criteria, the two files get sent via HTTP to a remote machine, and the response from that machine is presented to the user.

We can just extract the submission routine into a new Python file and use that rather than going through the main application each time. The submission doesn't actually care what the filenames are, so we can just tell it to send the original binary as both files.

We get the response:

```
Error: SHA1 does match
```

If we try two completely different files, we get:

```
Error: MD5s do not match
```

So we want two files with the same MD5 hash but different SHA1 hashes: i.e. a MD5 hash collision. Google is accommodating and can find some. Upon submitting these, we get:

```
Error: Files do not have expected prefix.. you are so close right
now...
```

Ok, so the files can't just be arbitrary, they need a specific prefix. Given how the script operates, the prefix is probably the 4.5MB or so executable. A bit of further research about the construction of MD5 collisions with a chosen prefix leads to a website on Marc Stevens' HashClash project which provides source and executables that will generate two different files with the same MD5, and both of which have a shared (provided) prefix. Submitting the files generated by this utility, with the original executable as the prefix, results in the flag.