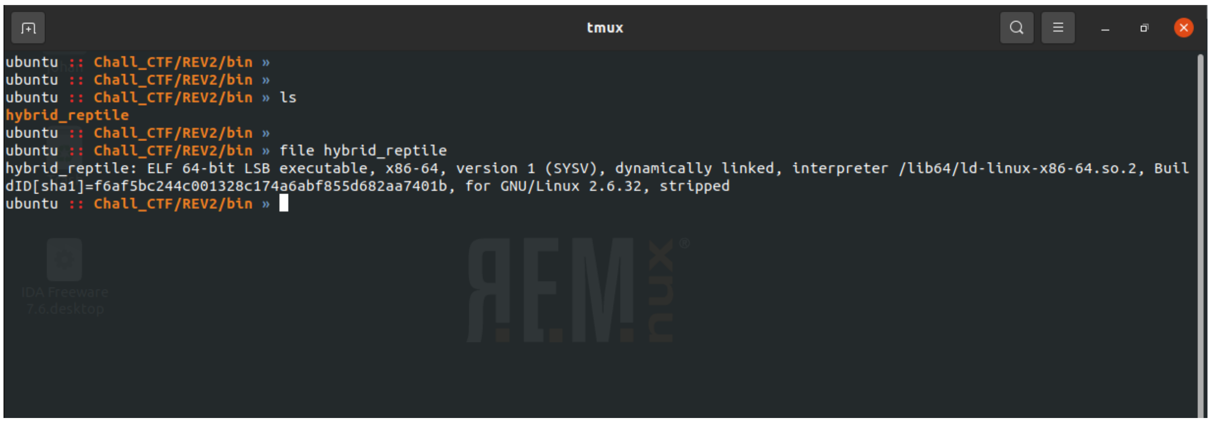
Hybrid Reptile

As always let's start by seeing what we're dealing with.

We got an ELF file and it is stripped.

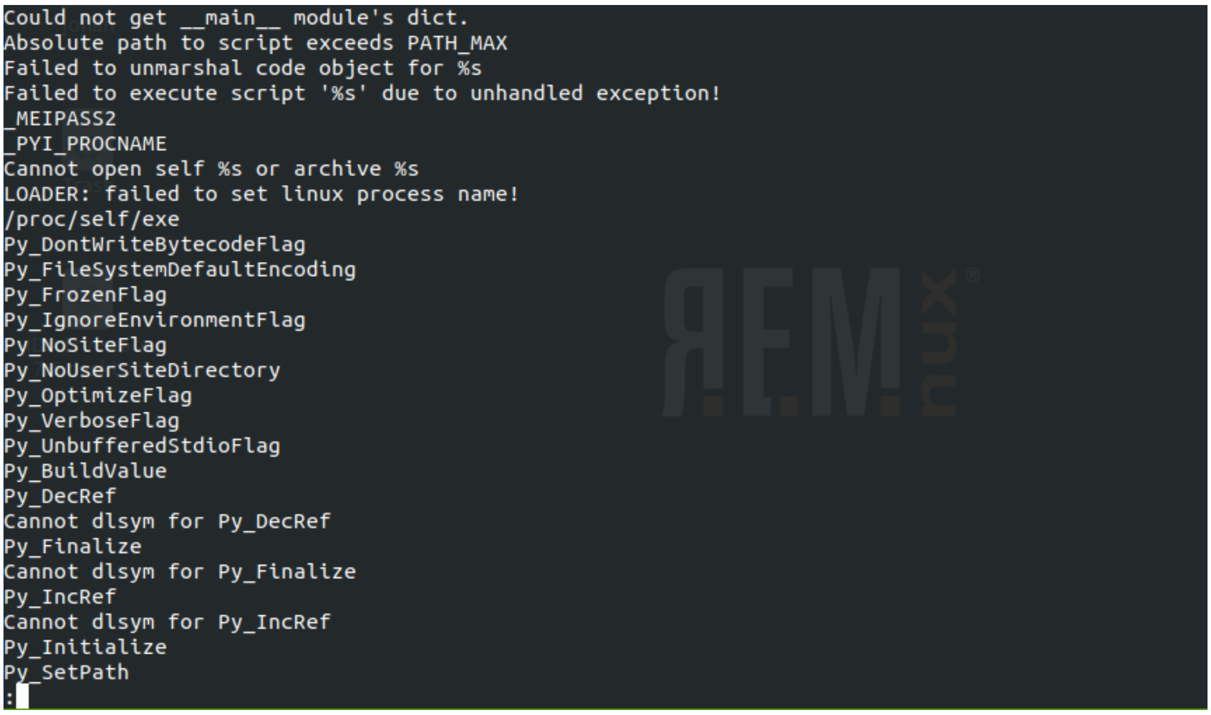


```
ubuntu :: Chall_CTF/REV2/bin »
ubuntu :: Chall_CTF/REV2/bin »
ubuntu :: Chall_CTF/REV2/bin » ls
hybrid_reptile
ubuntu :: Chall_CTF/REV2/bin »
ubuntu :: Chall_CTF/REV2/bin » file hybrid_reptile
hybrid_reptile: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, Buil
dID[sha1]=f6af5bc244c001328c174a6abf855d682aa7401b, for GNU/Linux 2.6.32, stripped
ubuntu :: Chall_CTF/REV2/bin »
```

Let's see if there is any hard-coded text in the binary.

strings give us a lot of insight about this file.

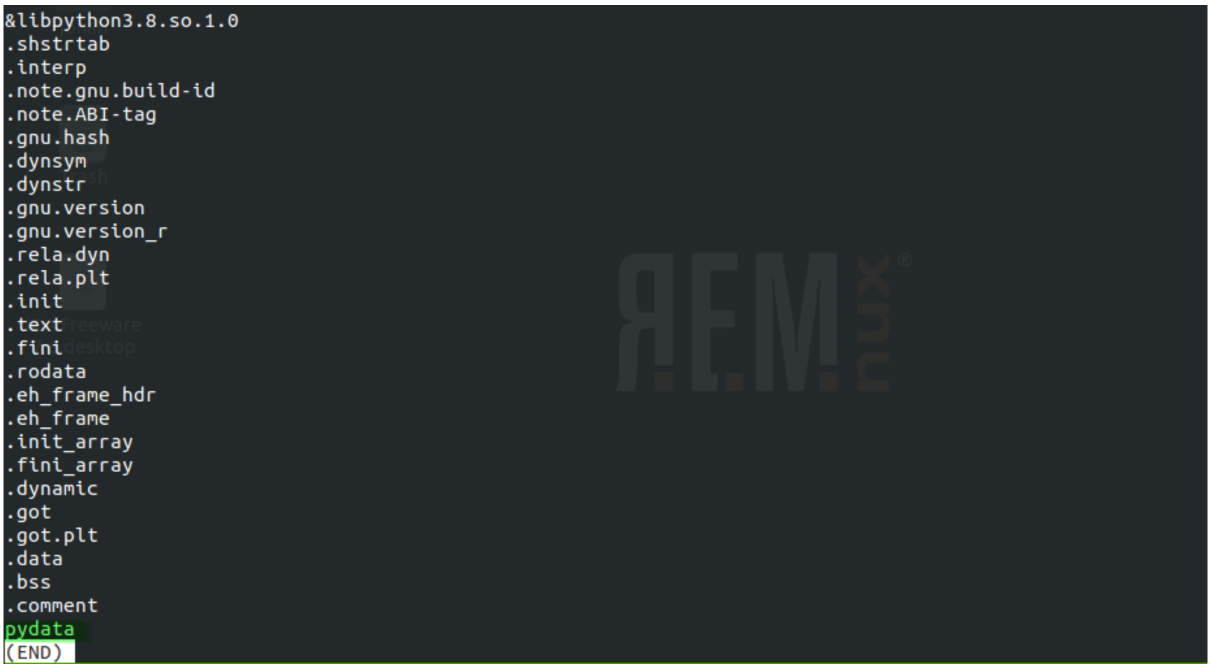
We're seeing a lot of python related strings.



```
Could not get __main__ module's dict.
Absolute path to script exceeds PATH_MAX
Failed to unmarshal code object for %s
Failed to execute script '%s' due to unhandled exception!
_MEIPASS2
_PYI_PROCNAME
Cannot open self %s or archive %s
LOADER: failed to set linux process name!
/proc/self/exe
Py_DontWriteBytecodeFlag
Py_FileSystemDefaultEncoding
Py_FrozenFlag
Py_IgnoreEnvironmentFlag
Py_NoSiteFlag
Py_NoUserSiteDirectory
Py_OptimizeFlag
Py_VerboseFlag
Py_UnbufferedStdioFlag
Py_BuildValue
Py_DecRef
Cannot dlsym for Py_DecRef
Py_Finalize
Cannot dlsym for Py_Finalize
Py_IncRef
Cannot dlsym for Py_IncRef
Py_Initialize
Py_SetPath
:
```

We see one interesting piece of string called `pydata` here. Why this is important? cuz this section usually exist in the binaries which were compiled from python script.

Mostly compiled using pyinstaller.

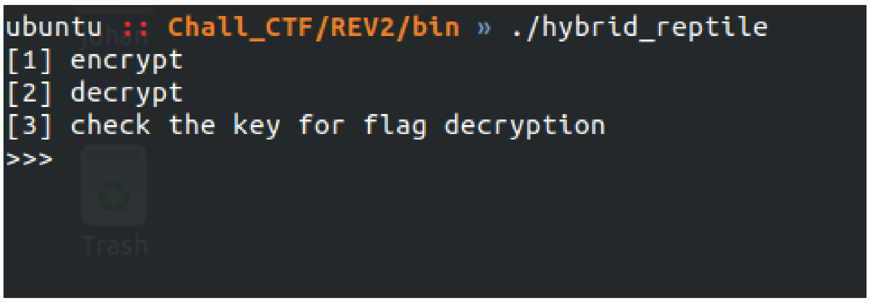


```
&libpython3.8.so.1.0
.shstrtab
.interp
.note.gnu.build-id
.note.ABI-tag
.gnu.hash
.dynsym
.dynstr
.gnu.version
.gnu.version_r
.rela.dyn
.rela.plt
.init
.text
.fini
.rodata
.eh_frame_hdr
.eh_frame
.init_array
.fini_array
.dynamic
.got
.got.plt
.data
.bss
.comment
pydata
(END)
```

Now let's see what this binary does.

This binary starts with giving us some options to work with.

1. we can encrypt a text
2. decrypt a text
3. check the key for the flag.



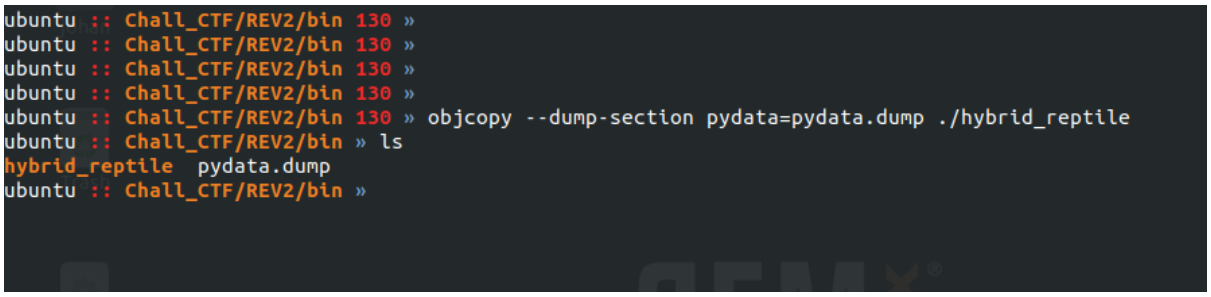
```
ubuntu :: Chall_CTF/REV2/bin » ./hybrid_reptile
[1] encrypt
[2] decrypt
[3] check the key for flag decryption
>>>
```

The encrypt option will take plain text as input encrypt and give encrypted string and the key for the encryption. The decrypt fucntion will take the encrypted text as input and ask for the key and give plain text as result.

The interesting function here is the 3rd one. It only gets the key as input and will tell us whether or not the key is correct one for decrypting the flag.

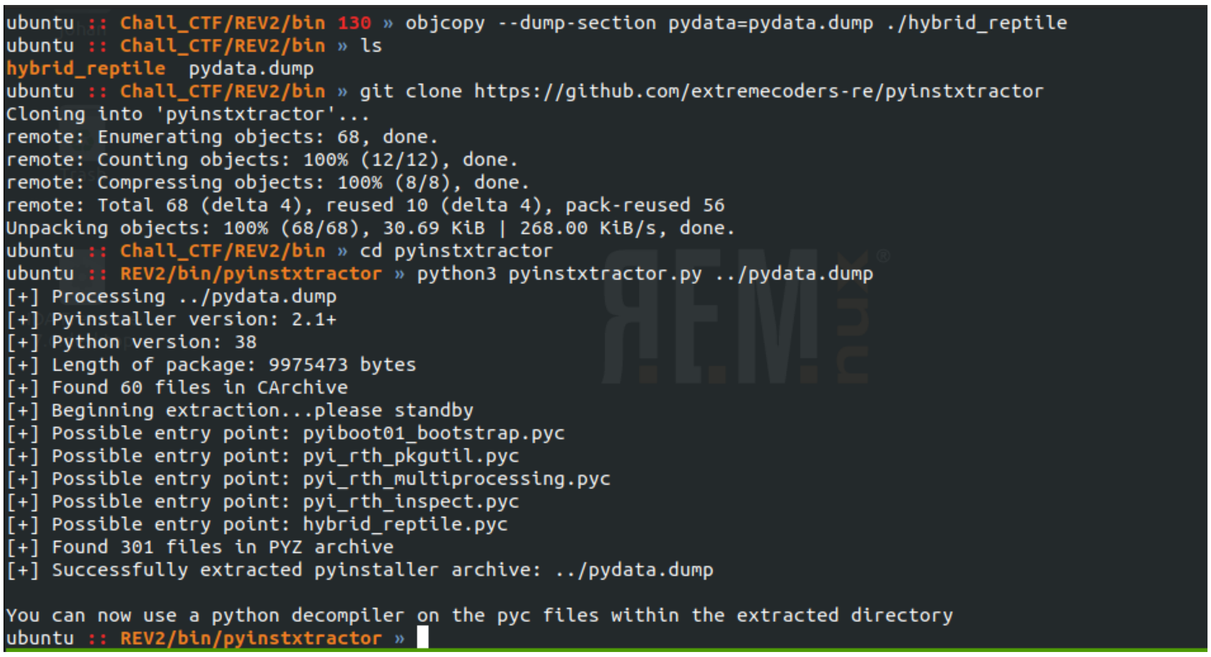
One awesome thing about python compiled binary is that we can easily obtain the source code of the program.

With `objcopy` utility we can dump sections of a binary. In our case we want pydata section as it holds all the information that we need.



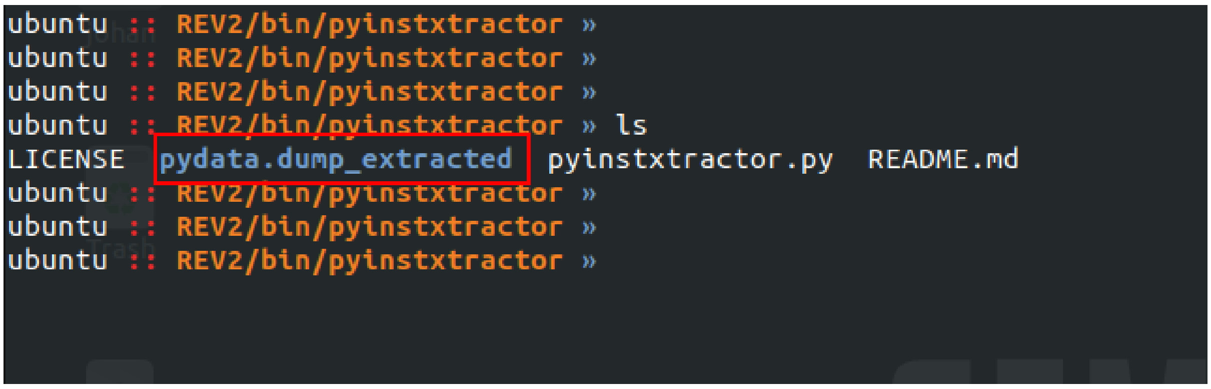
```
ubuntu :: Chall_CTF/REV2/bin 130 »
ubuntu :: Chall_CTF/REV2/bin 130 »
ubuntu :: Chall_CTF/REV2/bin 130 »
ubuntu :: Chall_CTF/REV2/bin 130 »
ubuntu :: Chall_CTF/REV2/bin 130 » objcopy --dump-section pydata=pydata.dump ./hybrid_reptile
ubuntu :: Chall_CTF/REV2/bin » ls
hybrid_reptile  pydata.dump
ubuntu :: Chall_CTF/REV2/bin »
```

And with the help of Pyinsextractor we can extract the scripts and dependencies from the pydata dump.



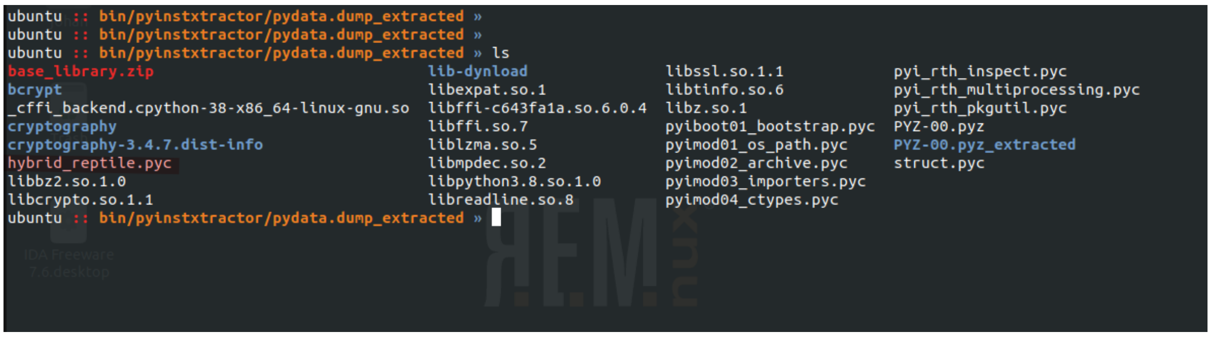
```
ubuntu :: Chall_CTF/REV2/bin 130 » objcopy --dump-section pydata=pydata.dump ./hybrid_reptile
ubuntu :: Chall_CTF/REV2/bin » ls
hybrid_reptile  pydata.dump
ubuntu :: Chall_CTF/REV2/bin » git clone https://github.com/extremecoders-re/pyinstxtractor
Cloning into 'pyinstxtractor'...
remote: Enumerating objects: 68, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 68 (delta 4), reused 10 (delta 4), pack-reused 56
Unpacking objects: 100% (68/68), 30.69 KiB | 268.00 KiB/s, done.
ubuntu :: Chall_CTF/REV2/bin » cd pyinstxtractor
ubuntu :: REV2/bin/pyinstxtractor » python3 pyinstxtractor.py ../pydata.dump
[+] Processing ../pydata.dump
[+] Pyinstaller version: 2.1+
[+] Python version: 38
[+] Length of package: 9975473 bytes
[+] Found 60 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_pkgutil.pyc
[+] Possible entry point: pyi_rth_multiprocessing.pyc
[+] Possible entry point: pyi_rth_inspect.pyc
[+] Possible entry point: hybrid_reptile.pyc
[+] Found 301 files in PYZ archive
[+] Successfully extracted pyinstaller archive: ../pydata.dump

You can now use a python decompiler on the pyc files within the extracted directory
ubuntu :: REV2/bin/pyinstxtractor »
```

Now we have a new directory under the current directory.



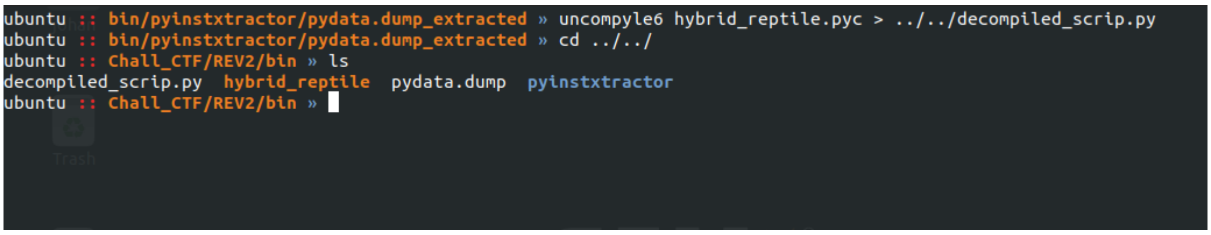
```
ubuntu :: REV2/bin/pyinstxtractor »
ubuntu :: REV2/bin/pyinstxtractor »
ubuntu :: REV2/bin/pyinstxtractor »
ubuntu :: REV2/bin/pyinstxtractor » ls
LICENSE  pydata.dump_extracted  pyinstxtractor.py  README.md
ubuntu :: REV2/bin/pyinstxtractor »
ubuntu :: REV2/bin/pyinstxtractor »
ubuntu :: REV2/bin/pyinstxtractor »
```

inside that directory we can see compiled python code



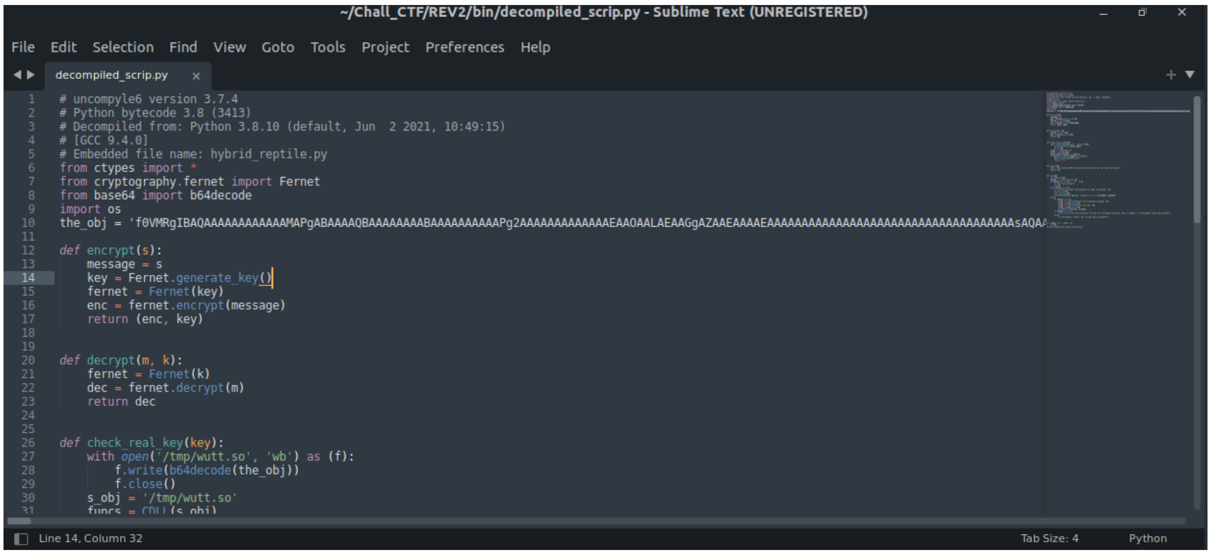
```
ubuntu :: bin/pyinstxtractor/pydata.dump_extracted »
ubuntu :: bin/pyinstxtractor/pydata.dump_extracted »
ubuntu :: bin/pyinstxtractor/pydata.dump_extracted » ls
base_library.zip                                lib-dynload               libssl.so.1.1              pyi_rth_inspect.pyc
bcrypt                                          libexpat.so.1             libtinfo.so.6              pyi_rth_multiprocessing.pyc
_cffi_backend.cpython-38-x86_64-linux-gnu.so    libffi-c643fa1a.so.6.0.4  libz.so.1                  pyi_rth_pkgutil.pyc
cryptography                                    libffi.so.7               pyiboot01_bootstrap.pyc    PYZ-00.pyz
cryptography-3.4.7.dist-info                    liblzma.so.5              pyimod01_os_path.pyc       PYZ-00.pyz_extracted
hybrid_reptile.pyc                              libmpdec.so.2             pyimod02_archive.pyc       struct.pyc
libbz2.so.1.0                                   libpython3.8.so.1.0       pyimod03_importers.pyc
libcrypto.so.1.1                                libreadline.so.8          pyimod04_ctypes.pyc
ubuntu :: bin/pyinstxtractor/pydata.dump_extracted »
```

so now with frameworks like `uncompyle6` we can easily revert this file into python source code.



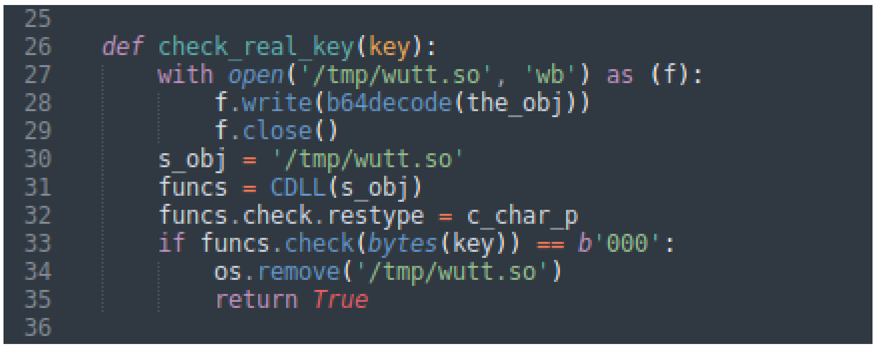
```
ubuntu :: bin/pyinstxtractor/pydata.dump_extracted » uncompyle6 hybrid_reptile.pyc > ../../decompiled_scrip.py
ubuntu :: bin/pyinstxtractor/pydata.dump_extracted » cd ../../
ubuntu :: Chall_CTF/REV2/bin » ls
decompiled_scrip.py  hybrid_reptile  pydata.dump  pyinstxtractor
ubuntu :: Chall_CTF/REV2/bin »
```

Let's take a look at the decompiled scrip now.


~/Chall_CTF/REV2/bin/decompiled_scrip.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
decompiled_scrip.py
Line 14, Column 32
Tab Size: 4
Python

```python
# uncompyle6 version 3.7.4
# Python bytecode 3.8 (3413)
# Decompiled from: Python 3.8.10 (default, Jun  2 2021, 10:49:15)
# [GCC 9.4.0]
# Embedded file name: hybrid_reptile.py
from ctypes import *
from cryptography.fernet import Fernet
from base64 import b64decode
import os
the_obj = 'f0VMRgIBAQAAAAAAAAAAAAAMAPgABAAAAQBAAAAAAAABAAAAAAAAAAAPg2AAAAAAAAAAAAAEAAOAALAEAAGgAZAAEAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAsAQAA

def encrypt(s):
    message = s
    key = Fernet.generate_key()
    fernet = Fernet(key)
    enc = fernet.encrypt(message)
    return (enc, key)


def decrypt(m, k):
    fernet = Fernet(k)
    dec = fernet.decrypt(m)
    return dec


def check_real_key(key):
    with open('/tmp/wutt.so', 'wb') as (f):
        f.write(b64decode(the_obj))
        f.close()
    s_obj = '/tmp/wutt.so'
    funcs = CDLL(s_obj)
```

let's ignore the `encrypt()` and `decrypt()` functions for now. Take a look at the `check_real_key()` function.

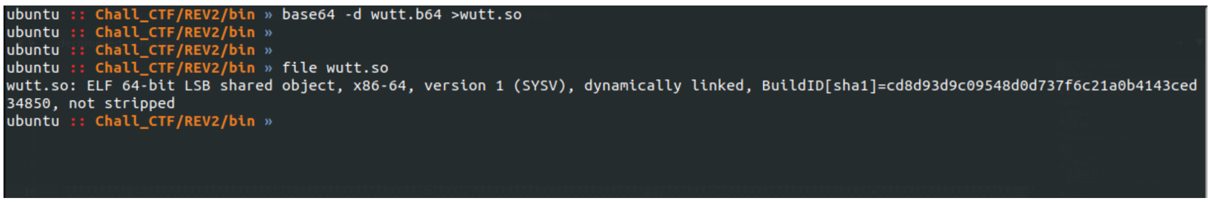


```python

def check_real_key(key):
    with open('/tmp/wutt.so', 'wb') as (f):
        f.write(b64decode(the_obj))
        f.close()
    s_obj = '/tmp/wutt.so'
    funcs = CDLL(s_obj)
    funcs.check.restype = c_char_p
    if funcs.check(bytes(key)) == b'000':
        os.remove('/tmp/wutt.so')
        return True

```

what this function does is it decodes `the_obj` from base64 and writes the results to /tmp/wutt.so(shared object). And uses python ctype to access the shared functions from that object file. From the looks of it there is a function called `check` in that shared object and it takes string input. And after taking the key if the output is equal to `b'000'` then it decides that the function is valid. Finally it deletes the `wutt.so` from `/tmp`.

```
def check_real_key(key):
  with open('/tmp/wutt.so', 'wb') as (f): #opening file descryptor
    # writes base64decoded result on "/tmp/wutt.so"
    f.write(b64decode(the_obj))
    f.close()
  s_obj = '/tmp/wutt.so'
  funcs = CDLL(s_obj)
  funcs.check.restype = c_char_p # defining the ctype func input as char(str)
  if funcs.check(bytes(key)) == b'000':
    os.remove('/tmp/wutt.so') # removes after execution
  return True
```

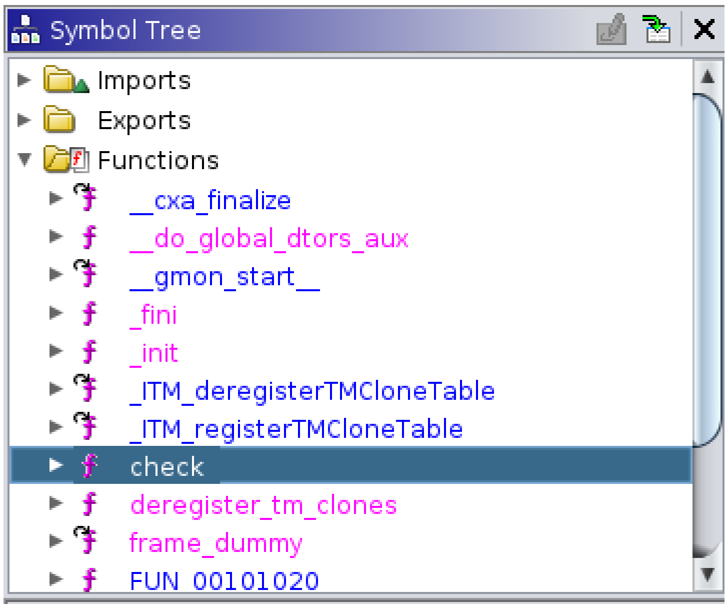
Let's manually decode the base64 encoded string and analyse `wutt.so`.



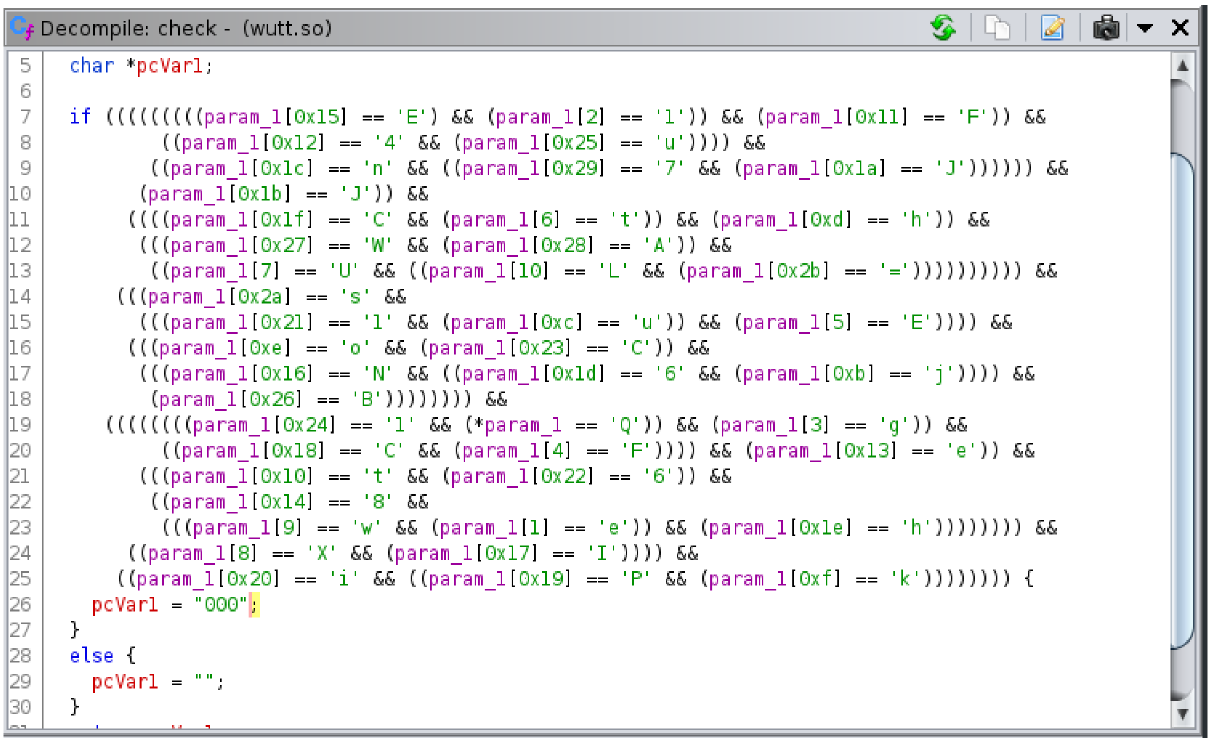
```
ubuntu :: Chall_CTF/REV2/bin » base64 -d wutt.b64 >wutt.so
ubuntu :: Chall_CTF/REV2/bin »
ubuntu :: Chall_CTF/REV2/bin »
ubuntu :: Chall_CTF/REV2/bin » file wutt.so
wutt.so: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, BuildID[sha1]=cd8d93d9c09548d0d737f6c21a0b4143ced
34850, not stripped
ubuntu :: Chall_CTF/REV2/bin »
```

We can analyse this file with Ghidra.

The shared object only have one user defined fucntion.



When seeing decompiled version of this function.. we see some mangled array. And it is returning two things based on the condition. After changing the data type to char we can clearly see what it return. So based on what we have if our input matches with this conditions it returns `"000"` and if we remember from the `check_real_key()` function if the return type matches with `b"000"` it gives success message.



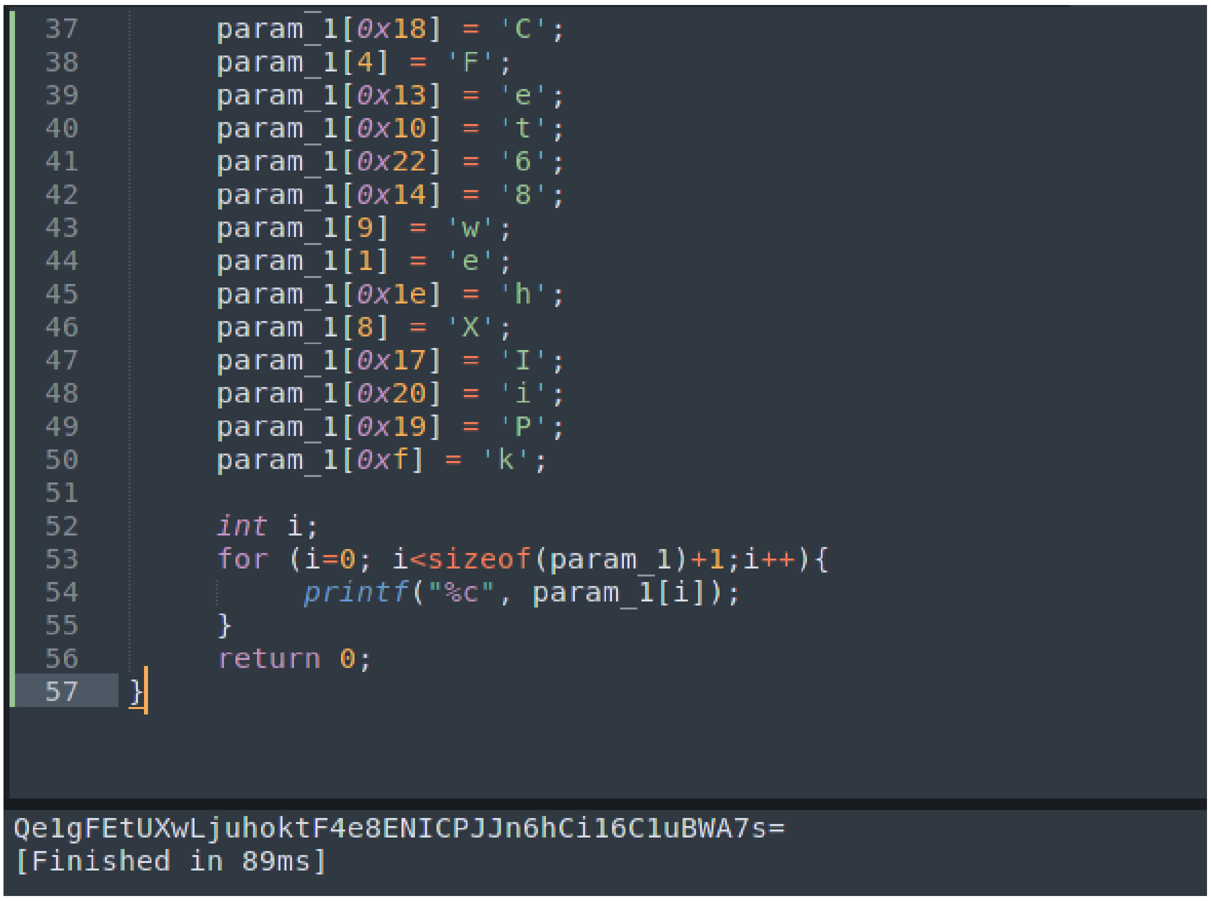
```
  char *pcVar1;

  if ((((((((((param_1[0x15] == 'E') && (param_1[2] == '1')) && (param_1[0x11] == 'F')) &&
          ((param_1[0x12] == '4' && (param_1[0x25] == 'u')))) &&
         ((param_1[0x1c] == 'n' && ((param_1[0x29] == '7' && (param_1[0x1a] == 'J')))))) &&
        (param_1[0x1b] == 'J')) &&
       ((((param_1[0x1f] == 'C' && (param_1[6] == 't')) && (param_1[0xd] == 'h')) &&
        (((param_1[0x27] == 'W' && (param_1[0x28] == 'A')) &&
         ((param_1[7] == 'U' && ((param_1[10] == 'L' && (param_1[0x2b] == '=')))))))))) &&
      (((param_1[0x2a] == 's' &&
        (((param_1[0x21] == '1' && (param_1[0xc] == 'u')) && (param_1[5] == 'E')))) &&
       (((param_1[0xe] == 'o' && (param_1[0x23] == 'C')) &&
        (((param_1[0x16] == 'N' && ((param_1[0x1d] == '6' && (param_1[0xb] == 'j')))) &&
         (param_1[0x26] == 'B')))))))) &&
     (((((((param_1[0x24] == '1' && (*param_1 == 'Q')) && (param_1[3] == 'g')) &&
          ((param_1[0x18] == 'C' && (param_1[4] == 'F')))) && (param_1[0x13] == 'e')) &&
        (((param_1[0x10] == 't' && (param_1[0x22] == '6')) &&
         ((param_1[0x14] == '8' &&
          (((param_1[9] == 'w' && (param_1[1] == 'e')) && (param_1[0x1e] == 'h')))))))) &&
       ((param_1[8] == 'X' && (param_1[0x17] == 'I')))) &&
      ((param_1[0x20] == 'i' && ((param_1[0x19] == 'P' && (param_1[0xf] == 'k')))))))) {
    pcVar1 = "000";
  }
  else {
    pcVar1 = "";
  }
```

So with reordering this array we can get our key for flag decryption.

With the following C script we can get the key easily.

```
#include <stdio.h>
#include <string.h>

int main(){
     char param_1[43];

     param_1[0x15] = 'E';
     param_1[2] = '1';
     param_1[0x11] = 'F';
     param_1[0x12] = '4';
     param_1[0x25] = 'u';
     param_1[0x1c] = 'n';
     param_1[0x29] = '7';
     param_1[0x1a] = 'J';
     param_1[0x1b] = 'J';
     param_1[0x1f] = 'C';
     param_1[6] = 't';
     param_1[0xd] = 'h';
     param_1[0x27] = 'W';
     param_1[0x28] = 'A';
     param_1[7] = 'U';
     param_1[10] = 'L';
     param_1[0x2b] = '=';
     param_1[0x2a] = 's';
     param_1[0x21] = '1';
     param_1[0xc] = 'u';
```

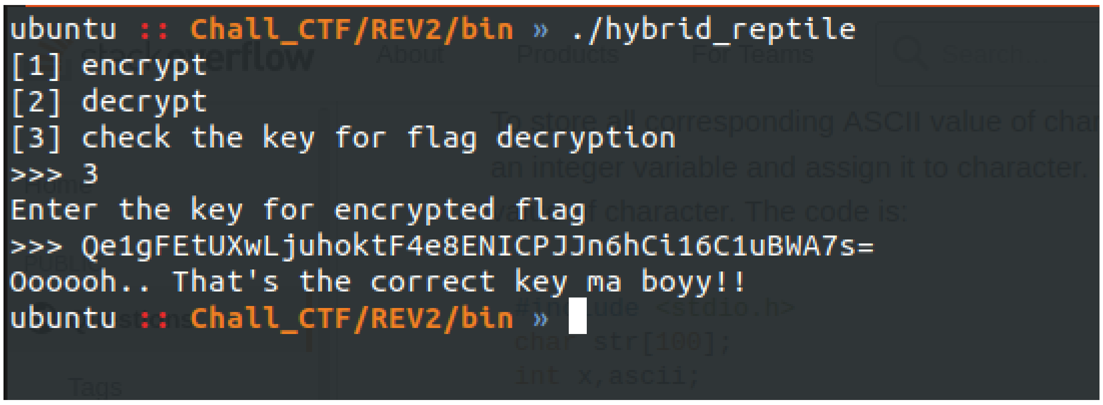
```
       param_1[5] = 'E';
       param_1[0xe] = 'o';
       param_1[0x23] = 'C';
       param_1[0x16] = 'N';
       param_1[0x1d] = '6';
       param_1[0xb] = 'j';
       param_1[0x26] = 'B';
       param_1[0x24] = '1';
       param_1[0] = 'Q';
       param_1[3] = 'g';
       param_1[0x18] = 'C';
       param_1[4] = 'F';
       param_1[0x13] = 'e';
       param_1[0x10] = 't';
       param_1[0x22] = '6';
       param_1[0x14] = '8';
       param_1[9] = 'w';
       param_1[1] = 'e';
       param_1[0x1e] = 'h';
       param_1[8] = 'X';
       param_1[0x17] = 'I';
       param_1[0x20] = 'i';
       param_1[0x19] = 'P';
       param_1[0xf] = 'k';

       int i;
       for (i=0; i<sizeof(param_1)+1;i++){
            printf("%c", param_1[i]);
       }
       return 0;
  }
```



```
        param_1[0x18] = 'C';
        param_1[4] = 'F';
        param_1[0x13] = 'e';
        param_1[0x10] = 't';
        param_1[0x22] = '6';
        param_1[0x14] = '8';
        param_1[9] = 'w';
        param_1[1] = 'e';
        param_1[0x1e] = 'h';
        param_1[8] = 'X';
        param_1[0x17] = 'I';
        param_1[0x20] = 'i';
        param_1[0x19] = 'P';
        param_1[0xf] = 'k';

        int i;
        for (i=0; i<sizeof(param_1)+1;i++){
            printf("%c", param_1[i]);
        }
        return 0;
}
```

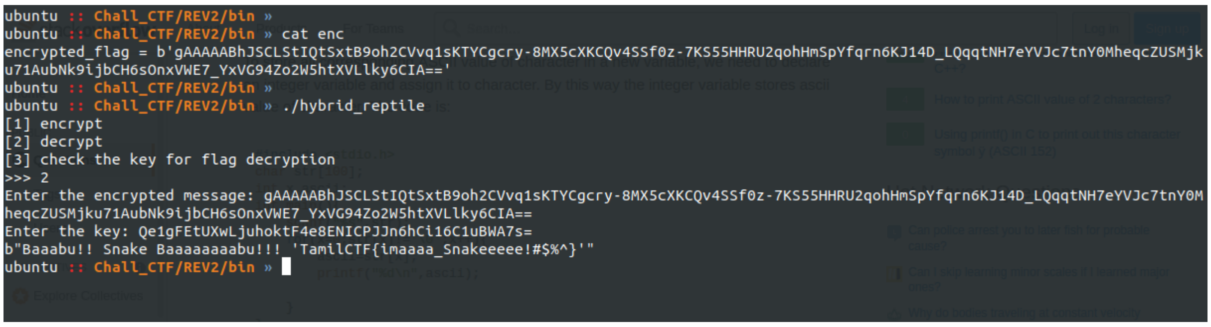
```
Qe1gFEtUXwLjuhoktF4e8ENICPJJn6hCi16C1uBWA7s=
[Finished in 89ms]
```

Looks like a valid key. We can check if it is the correct key with the binary we have. And yep.. that's the correct key.



```
ubuntu :: Chall_CTF/REV2/bin » ./hybrid_reptile
[1] encrypt
[2] decrypt
[3] check the key for flag decryption
>>> 3
Enter the key for encrypted flag
>>> Qe1gFEtUXwLjuhoktF4e8ENICPJJn6hCi16C1uBWA7s=
Oooooh.. That's the correct key ma boyy!!
ubuntu :: Chall_CTF/REV2/bin »
```

We can decrypt the encrypted message now.



```
ubuntu :: Chall_CTF/REV2/bin »
ubuntu :: Chall_CTF/REV2/bin » cat enc
encrypted_flag = b'gAAAAABhJSCLStIQtSxtB9oh2CVvq1sKTYCgcry-8MX5cXKCQv4SSf0z-7KS55HHRU2qohHmSpYfqrn6KJ14D_LQqqtNH7eYVJc7tnY0MheqcZUSMjk
u71AubNk9ijbCH6sOnxVWE7_YxVG94Zo2W5htXVLlky6CIA=='
ubuntu :: Chall_CTF/REV2/bin »
ubuntu :: Chall_CTF/REV2/bin » ./hybrid_reptile
[1] encrypt
[2] decrypt
[3] check the key for flag decryption
>>> 2
Enter the encrypted message: gAAAAABhJSCLStIQtSxtB9oh2CVvq1sKTYCgcry-8MX5cXKCQv4SSf0z-7KS55HHRU2qohHmSpYfqrn6KJ14D_LQqqtNH7eYVJc7tnY0M
heqcZUSMjku71AubNk9ijbCH6sOnxVWE7_YxVG94Zo2W5htXVLlky6CIA==
Enter the key: Qe1gFEtUXwLjuhoktF4e8ENICPJJn6hCi16C1uBWA7s=
b"Baaabu!! Snake Baaaaaaaabu!!! 'TamilCTF{imaaaa_Snakeeeee!#$%^}'"
ubuntu :: Chall_CTF/REV2/bin »
```

`Flag : TamilCTF{imaaaa_Snakeeeee!#$%^}`