Hissss

9th 1 23 / Document No. D22.102.99

Prepared By: clubby789

Challenge Author: MinatoTW

Difficulty: Easy

Classification: Official

Synopsis

Hisss is an easy Reversing challenge.

Skills Required

- A

Skills Learned

- A

Solution

We're given an ELF file which is unusually large in size.

```
$ file auth
auth: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=3507aa01d32c34dc8e8c6462b764adb90a82768d, stripped

$ du -hs auth
7.0M    auth
```

This could be a hint that this is packed or has additional resources in it.

```
$ string auth

Py_DontWriteBytecodeFlag
Py_FileSystemDefaultEncoding
Py_FrozenFlag
Py_IgnoreEnvironmentFlag
Py_NoSiteFlag
```

Running strings on it reveals some python library functions and variables. It's possible that this is a packed python script. The pyinstxtractor wiki shows how we can unpack such binaries.

```
$ objcopy --dump-section pydata=pydata.dump ../auth
$ python pyinstxtractor.py pydata.dump

[+] Processing pydata.dump
[+] Pyinstaller version: 2.1+
[+] Python version: 38
[+] Length of package: 7196547 bytes
[+] Found 68 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_multiprocessing.pyc
[+] Possible entry point: auth.pyc
[+] Found 211 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
```

We find compiled python bytecode in the `pydata.dump_extracted/auth.pyc` file. Let's use decompyle3 to reconstruct the source code.

```
decompyle3 auth.pyc
```

The resulting source code is as follows:

```
import sys
password = input('Enter password> ')
if len(password) != 12:
    print("Sorry! You've entered the wrong password.")
    sys.exit(0)

if ord(password[0]) != 48 or password[11] != '!' or ord(password[7]) !=
ord(password[5]) or 143 - ord(password[0]) != ord(password[4]) or
ord(password[1]) ^ ord(password[3]) != 30 or ord(password[2]) *
ord(password[3]) != 5610 or password[1] != 'p' or ord(password[6]) -
ord(password[8]) != -46 or ord(password[6]) ^ ord(password[7]) != 64 or
ord(password[10]) + ord(password[5]) != 166 or ord('n') - ord(password[9]) != 1
or password[10] != str(3):
    print('Sorry, the password is incorrect.')
else:
    print(f"Well Done! HTB{{{password}}}")
```

The input is subjected to a few constrains which we need to solve.

```
password[0] = chr(48) = '0'
password[1] = 'p'
password[3] = 'p' ^ 30 = 'n'
password[2] = 5610 / 'n' = '3'
password[4] = 143 - '0' = '_'
password[10] = '3'
password[5] = 166 - '3' = 's'
password[7] = 's'
password[6] = 's' ^ 64 = '3'
password[8] = '3' + 46 = 'a'
password[9] = 'n' - 1 = 'm'
password[11] = '!'
```

This brings the final password to `0p3n_s3sam3!`. Entering this input as the binary's password causes it to print out the flag.