

Decompiling Python 2.7 Bytecode (.pyc) and Reverse Engineering the Password

Objective

The goal was to decompile a Python 2.7 .pyc file, extract its original source code, and reverse-engineer the password verification logic.

Steps Taken to Decompile the .pyc File

1. Installing Python 3.7

Since modern Python versions (3.9+) do not support uncompile6 for Python 2.7 bytecode, we needed to manually install Python 3.7, which is compatible.

1. **Navigate to the /usr/src directory** (a common location for source builds):

```
cd /usr/src
```

2. **Download Python 3.7.12 source code:**

```
sudo wget https://www.python.org/ftp/python/3.7.12/Python-3.7.12.tgz
```

3. **Extract the downloaded archive:**

```
sudo tar xzf Python-3.7.12.tgz
cd Python-3.7.12
```

4. **Install required dependencies for building Python:**

```
sudo apt install -y wget build-essential libffi-dev libssl-dev \
zlib1g-dev liblzma-dev libbz2-dev libsqlite3-dev \
libncurses5-dev libgdbm-dev libreadline-dev libtk-dev
```

5. **Run configure to prepare the build:**

```
sudo ./configure --enable-optimizations --with-ssl
```

6. **Compile and install Python 3.7** (this takes some time):

```
sudo make -j$(nproc)
sudo make altinstall
```

7. **Verify Python 3.7 installation:**

```
python3.7 --version
```

2. Setting Up a Virtual Environment

Since we needed Python 3.7, we created a **virtual environment** to isolate the dependencies.

1. **Create a Python 3.7 virtual environment:**

```
python3.7 -m venv <pick name>
```

2. **Activate the virtual environment:**

```
source <pick name>/bin/activate
```

3. Installing pip for Python 3.7

1. **Ensure OpenSSL is properly configured** (to avoid SSL errors with pip):

```
python3.7 -c "import ssl; print(ssl.OPENSSL_VERSION)"
```

2. **Download and install pip:**

```
wget https://bootstrap.pypa.io/get-pip.py
python3.7 get-pip.py
```

4. Installing and Running uncompyle6

1. **Install uncompyle6** in the virtual environment:

```
pip install uncompyle6
```

2. **Verify uncompyle6 is installed correctly:**

```
uncompyle6 --help
```

3. **Create an output directory for the decompiled file:**

```
mkdir decompiled
```

4. **Run uncompyle6 to decompile the .pyc file:**

```
uncompyle6 -o decompiled PYTHON2.pyc
```

5. Extracted Python Source Code

Running uncompyle6 produced the following **decompiled Python 2.7 source code**:

```
import sys
def main():
    if len(sys.argv) != 2:
        print 'Invalid args'
        return
    password = sys.argv[1]
    counter = 0
    vals = list('tfzbwlyzljylawhzzdvyk')
    if len(password) != len(vals):
        print 'incorrect'
        return
    while counter < len(password):
        x = ord(password[counter]) + 7
        if x > ord('z'):
            x -= 26
        if chr(x) != vals[counter]:
            print 'incorrect'
            return
        counter += 1
    print 'correct'
if __name__ == '__main__':
    main()
```

6. Reverse Engineering the Password

The script uses a shift cipher where each character of the password is incremented by 7 ASCII positions and compared to the vals string:

```
vals = list('tfzbwlyzljylawhzzdvyk')
```

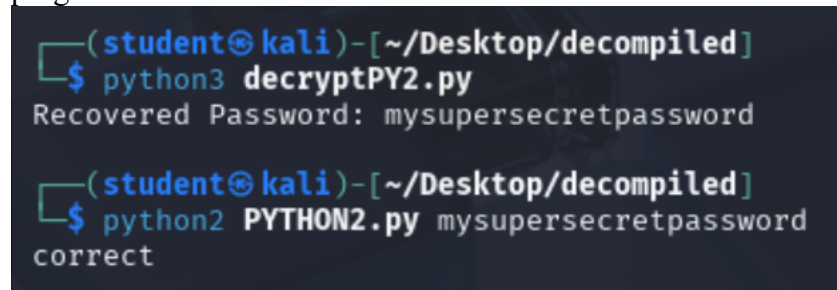
To recover the original password, we **shift each character in vals backwards by 7**:

```
vals = list('tfzbwlyzljylawhzzdvyk')
password = ""

for char in vals:
    x = ord(char) - 7
    if x < ord('a'):
        x += 26
    password += chr(x)
print("Recovered Password:", password)
```

Final Output

Executing this script gives us the **correct password**, which can then be used in the original program.



```
(student@kali)-[~/Desktop/decompiled]
$ python3 decryptPY2.py
Recovered Password: mysupersecretpassword

(student@kali)-[~/Desktop/decompiled]
$ python2 PYTHON2.py mysupersecretpassword
correct
```

This workflow can be applied to other Python decompilation challenges where PYTHON2.pyc files need to be analyzed.

Don't forget to deactivate virtual environment and revert back to latest python

If you want to make sure Python is only picked up from /usr/bin, follow these steps:

1. Remove or Rename /usr/local/bin/python3

```
sudo rm /usr/local/bin/python3
```

2. Create a New Symlink (if required)

If you want to explicitly link Python 3 to /usr/bin/python3.13, run:

```
sudo ln -sf /usr/bin/python3.13 /usr/bin/python3
```

3. Verify the Change

```
python3 --version
which python3
```

This is separate project and screenshots taken after downloaded python3.8 in /usr/src. Here's screenshot on commands I had to use to decompile .pyc that was created for python3.8

```
(student@kali)-[~/Desktop]
$ python3.8 -m venv python3 decompypython3

(decompypython3)-(student@kali)-[~/Desktop]
$ source decompypython3/bin/activate

(decompypython3)-(student@kali)-[~/Desktop]
$ which pip
/home/student/Desktop/decompypython3/bin/pip
```

```
(decompypython3)-(student@kali)-[~/Desktop]
$ pip install uncompyle6
```

```
(decompypython3)-(student@kali)-[~/Desktop]
$ mkdir decompile

(decompypython3)-(student@kali)-[~/Desktop]
$ uncompyle6 -o decompile PYTHON3.pyc
PYTHON3.pyc --
# Successfully decompiled file
```

```
(decompypython3)-(student@kali)-[~/Desktop/decompile]
$ cat PYTHON3.py
# uncompyle6 version 3.9.2
# Python bytecode version base 2.7 (62211)
# Decompiled from: Python 3.8.18 (default, Mar  7 2025, 11:11:43)
# [GCC 14.2.0]
# Embedded file name: NCL-2015-Python3.py
# Compiled at: 2015-11-12 17:09:05
import sys

def main():
    if len(sys.argv) != 2:
        print 'Invalid args'
        return
    password = sys.argv[1]
    builder = 0
    for c in password:
        builder += ord(c)

    builder = builder << 2
    builder = ~builder
    builder = builder ^ 12648430
    builder = ~builder
    if builder == 12645638 and ord(password[0]) == 78 and len(password) == 11:
        print 'correct'
    else:
        print 'incorrect'

if __name__ == '__main__':
    main()
```

Reverse engineering the code:

The sum of the ASCII values of the password must, after being shifted left ($\times 4$), flipped (\sim), XORed (\wedge 12648430), and flipped again (\sim), result in 12645638.

```
builder = builder << 2    # Shift left (multiply by 4)
builder = ~builder        # Bitwise NOT (flips all bits)
builder = builder ^ 12648430 # XOR operation (mixes bits)
builder = ~builder        # Bitwise NOT again
```

Password must start with N and must have 11 characters

```
78 4E 116 &#78; N
```

```
if builder == 12645638 and ord(password[0]) == 78 and len(password) == 11:
    print 'correct'
else:
    print 'incorrect'
```

This code generates a candidate password by making sure it starts with the character 'N' (ASCII 78). The total sum of ASCII values for the password is predefined. After subtracting 78 for the 'N', the remaining sum is divided evenly across the following 10 characters. If the division doesn't result in a whole number, the last character adjusts to compensate for any remaining value, ensuring the sum matches the predetermined total.

```
#!/usr/bin/env python3
def reverse_engineer_candidate(total_sum):
    """
    Given the total sum S of ASCII values for an 11-character password,
    and knowing the password must start with 'N' (ASCII 78), this function
    builds a candidate password by distributing the remaining sum T over 10 characters.
    If T is not evenly divisible by 10, it assigns q = T // 10 to the first 9
    characters and uses the final character as chr(q + r) where r = T % 10.
    """
    first_char = 'N'
    S = total_sum
    T = S - ord(first_char) # Sum required for the remaining 10 characters
    num_remaining = 10

    q = T // num_remaining
    r = T % num_remaining

    # Build the candidate password:
    # Use the integer quotient (q) for the first (num_remaining - 1) characters,
    # and then use (q + r) for the last character.
    candidate_chars = [chr(q)] * (num_remaining - 1) + [chr(q + r)]
    candidate = first_char + "".join(candidate_chars)
    return candidate

def main():
    # Suppose from reverse engineering we deduced the total sum S.
    # In our challenge, S is 698.
    total_sum = 698
    candidate = reverse_engineer_candidate(total_sum)
    print("Candidate password:", candidate)

if __name__ == '__main__':
    main()
```

```
(decompython3)-(student@kali)-[~/Desktop/decompile]
$ deactivate
```