

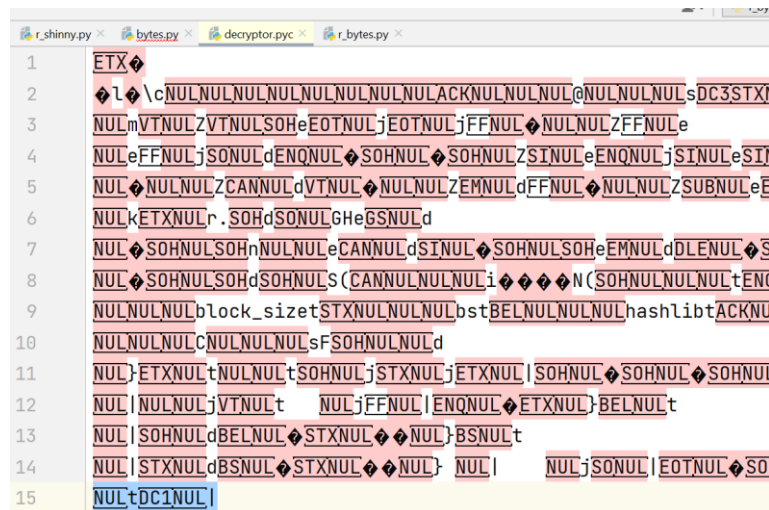
## Using Uncompyle6

### A Tutorial on Decompiling a Compiled Python Source Code File

#### A Git Hub Repo

<https://github.com/extremecoders-re/uncompyle6-builds>

Scenario: We have received, or discovered a pyc (a Python Compiled) file, which, when we open it in its native 'binary' form is unreadable or cannot otherwise be deciphered. If we were to use PyCharm or a text editor to open a .pyc file it would look something like this:



Not a lot of help here, and even if we parse the file for text (i.e., Control + F) we are unlikely to discover any real information. This is because this is bytecode (or code which is intended to be understood by an operating system, not a programmer).

In addition, we might try to just 'run' the file (which is what we would normally do with a .pyc file), but when we do, we get an odd error message at the command line.

Trying to run the code with a python 3 editor simply fails with a bad "magic number" reference

```
PS C:\Users\quincy\Documents\jjc\cyberwolves\pv2> python .\decryptor.pyc
RuntimeError: Bad magic number in .pyc file
```

A bit of experience might suggest to us that using Python Version 3 may not be able to 'run' this file, so we can try an older version of Python v 2.

```
PS C:\Users\quincy\Documents\jjc\cyberwolves\pv2> python27 .\decryptor.pyc
Traceback (most recent call last):
  File ".\decryptor.py", line 10, in <module>
    from crypto.Cipher import AES
ImportError: No module named Crypto
```

This time we get a different error message but still not running program. We can see, the source code file was named 'decryptor.py', and on line 10 there is a problem.

Solution:

The process of decompiling or returning a source code file from a binary file allows us to generate a readable text file with the original code. You might also hear this called reverse engineering, or simply decompiling a binary. So how do we 'reverse engineer' this file...? We will need a tool.

Enter: uncompyle6

You will find, and hopefully you don't code this way, most software in the industry has amazing promises and claims to do everything under the sun, but in reality, come with a minimal set of instructions, and even fewer examples of how to use the tool, and software which is intended to decompile

In addition, while modern versions of Windows have Python installed, this tends to be the minimal base version without any add-ins (other people's libraries). The traditional way to install addons for python is to use PIP, but the pip installer does not install these libraries as 'apps' or software on a Windows system. PIP will only install a software (lib) into a Python environment.

For this demonstration however, we cannot assume that a Python environment (a project) is available already in the Windows system. In other words, we want a stand-alone tool which we can use directly inside of Windows, without having to spin a virtual Python environment.

The best tool of this type (to decompile a python file in a Windows environment) is Uncompyle. The versions of this tool, 2, 3, 4... is added to the name, so for this demo we will use uncompyle6, or the sixth version.

As mentioned before, we will run this on a Windows system, which means that we are looking for an executable file (a file with a .exe at the end of it), and we can find this executable by visiting the GitHub Repo (see link at the top). We will need to: 1) Download the resource; 2) Decompress/unzip the resource; 3) Move the executable to the folder where the .pyc file is currently at; 4) run the executable there, pointing it at the .pyc file.

1 & 2) The Repo above provides source code as 7zip and in other binary and source code formats. We download the file and use 7zip to decompress (extract) the .exe. Once the exe file is exposed, then you need to move it into the same folder where the .pyc file is also found.

```
PS C:\Users\quincy\Documents\jjc\cyberwolves\pv2> dir

Directory: C:\Users\quincy\Documents\jjc\cyberwolves\pv2

Mode                LastWriteTime         Length Name
----                -
d-----          3/21/2023  12:07 AM              .idea
-a----          3/20/2023  11:20 PM           1119 bytes.py
-a----          3/21/2023  12:04 AM           3119 decryptor.py
-a----          3/19/2023  11:04 PM           3687 decryptor.pyc
-a----          3/20/2023  10:54 PM           4047 r_bytes.py
-a----          3/20/2023  10:08 PM           3994 r_shinny.py
-a----          3/19/2023  11:04 PM            696 shinny.py
-a----          12/27/2022   4:11 AM        46386688 uncompyle6.exe
```

### 3) Make sure the executable file is saved into the same folder as the .pyc file

> quincy > Documents > jjc > cyberwolves > pv2			Search pv2
<input type="checkbox"/> Name	Date modified	Type	
.idea	3/21/2023 12:07 AM	File folder	
bytes.py	3/20/2023 11:20 PM	Python source file	
decryptor.pyc	3/19/2023 11:04 PM	Compiled Python File	
r_bytes.py	3/20/2023 10:54 PM	Python source file	
r_shinny.py	3/20/2023 10:08 PM	Python source file	
shinny.py	3/19/2023 11:04 PM	Python source file	
<input checked="" type="checkbox"/> uncompile6.exe	12/27/2022 4:11 AM	Application	

- 4) The suggestion (see the official docs) says that you can simply run uncompile6 against a file in the directory as follows: So, using PowerShell, navigate to the folder where both the .exe and the .pyc file are saved. Then run:

```
>> uncompile6.exe -o . decryptor.pyc
```

This does throw an error because PowerShell does not 'know about the program', or in other words, it is not properly installed as an app on the system. This is okay. If you continue to read the error message all the way to the end however, PS does report that it did find an executable file (by that name) in the folder. In addition, the error message provides instructions on how to do a work around, which explicitly allows us to use the .exe from the command line.

```
PS C:\Users\quincy\Documents\jjc\cyberwolves\pv2> uncompile6.exe -o . decryptor.pyc
uncompile6.exe : The term 'uncompile6.exe' is not recognized as the name of a cmdlet, function, script file,
or operable program. Check the spelling of the name, or if a path was included, verify that the path is
correct and try again.
At line:1 char:1
+ uncompile6.exe -o . decryptor.pyc
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (uncompile6.exe:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

Suggestion [3,General]: The command uncompile6.exe was not found, but does exist in the current location. Wind
ows PowerShell does not load commands from the current location by default. If you trust this command, instead
type: ".\uncompile6.exe". See "get-help about_Command_Precedence" for more details.
PS C:\Users\quincy\Documents\jjc\cyberwolves\pv2>
```

The updated command to run in PowerShell is:

```
>> .\uncompile6.exe -o . decryptor.pyc
```

All you are doing here is adding the .\ before the exe file name, which is an indicator that you know that the file is in this directory, AND YES, I want to use THIS ONE. When you give that a go it delivers the decrypted (Python .py) file:

```
PS C:\Users\quincy\Documents\jjc\cyberwolves\pv2> .\uncompile6.exe -o . decryptor.pyc
decryptor.pyc --
# Successfully decompiled file
PS C:\Users\quincy\Documents\jjc\cyberwolves\pv2>
```

We can now view the source code using any editor (my suggestion is to use PyCharm)

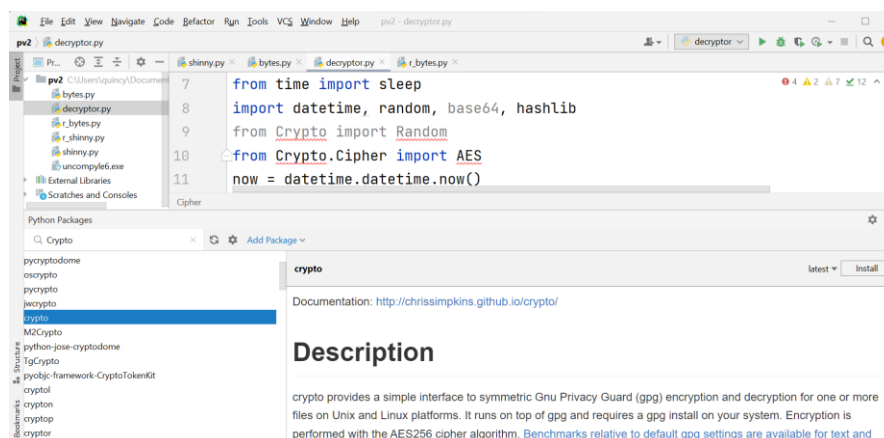
```
# uncomple6 version 3.9.0
# Python bytecode version base 2.7 (62211)
# Decompiled from: Python 3.10.8 (main, Nov 6 2022, 02:23:43) [MSC v.1929 64
# Embedded file name: ./decryptor.py
# Compiled at: 2019-04-07 16:34:37
import os, sys
from time import sleep
import datetime, random, base64, hashlib
from Crypto import Random
from Crypto.Cipher import AES
now = datetime.datetime.now()
seed = int(now.strftime('%Y%m%d'))
random.seed(seed)
cryptoKey = str(random.randint(0, sys.maxint))
password = 'goeeyflubber'
```

As you can tell the uncomple6 added some header data on the top of the source code file. We can also see that the reverse engineering effort produced plain text, python code, which is what we wanted. Do you notice the string 'password' variable?

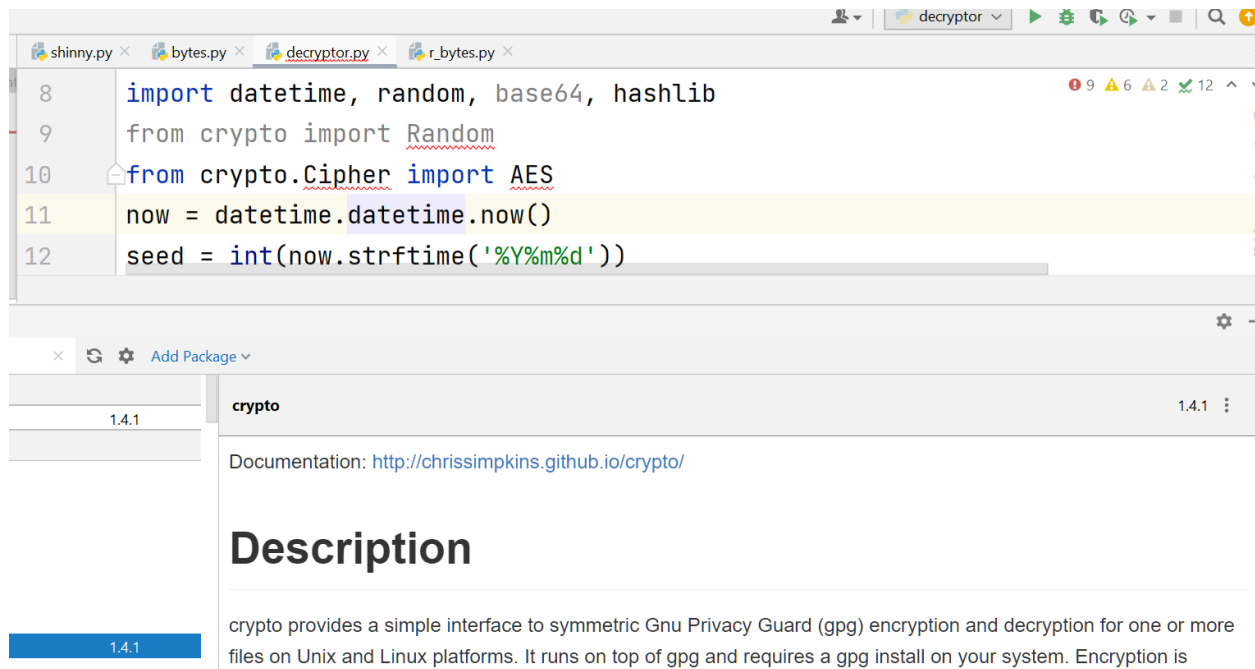
The file, which is decrypted here, also takes the name decryptor.py (which is the name we expected it would get after trying to run this file in Python v2). That same V2 error message also suggested that we would find a problem on Line 10.

On line 10 we do in fact see a red squiggle indicating there is an error in the source code. Errors in a source code file is a valid reason as to why we might want to 'decompile' a broken source code file in the first place. Because we can now read the file (text file) we can start to make out some of the issues which were causing the program to fail to run in the first place. Notice 'Crypto' which is the name of a module which either does not exist anymore or may have been a proprietary library.

PyCharm can be used to search for packages/modules, and Crypto is not found (with a capital letter C, but there is a package named crypto, so we will install it and see if we can resolve some of the errors.



Once crypto is installed, we also will need to update the import statement to use the proper capitalization (name it with in lower case crypto).



The screenshot shows a code editor with four tabs: `shinny.py`, `bytes.py`, `decryptor.py`, and `r_bytes.py`. The `decryptor.py` tab is active, displaying the following Python code:

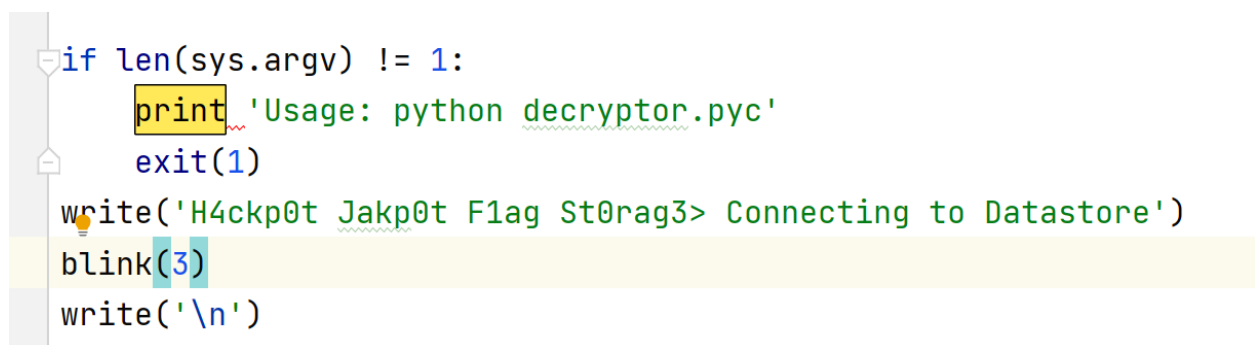
```
8 import datetime, random, base64, hashlib
9 from crypto import Random
10 from crypto.Cipher import AES
11 now = datetime.datetime.now()
12 seed = int(now.strftime('%Y%m%d'))
```

Below the code editor is a package manager interface. It shows a search for the package `crypto`, version `1.4.1`. The interface includes a "Description" section with the following text:

crypto provides a simple interface to symmetric Gnu Privacy Guard (gpg) encryption and decryption for one or more files on Unix and Linux platforms. It runs on top of gpg and requires a gpg install on your system. Encryption is

Even with `crypto` installed and updating the code from `Crypto` to `crypto` you can tell that the modules inside the package are not resolving properly. We see on line 10 above `crypto` (resolves properly), but `crypto.Cipher` (as well as `crypto.cypher`) are not modules. Also `Random` is not included in `crypto`, nor is `AES`. It's a shame but sometimes modules just go away and leave your code broken.

In addition, because we can now read the code, we can also discover that this source code file is Ancient, as the syntax being used is from nearly ten years ago when Python was in version 2.



The screenshot shows a snippet of Python code with several syntax errors highlighted by yellow boxes:

```
if len(sys.argv) != 1:
    print 'Usage: python decryptor.pyc'
    exit(1)
write('H4ckp0t Jakp0t Flag St0rag3> Connecting to Datastore')
blink(3)
write('\n')
```

Summary: When the need arises and we must find a way to 'read the source code' from a binary (compiled file) we have options. While this demo showed the Windows way the same library can be used on any system (MACOS, Linux) with similar results.