Lec3-8-3-15-protectingPython.ppt

Harry Li, Ph.D.

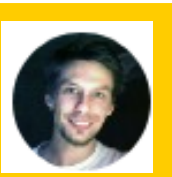
CTI One Corporation
3679 Enochs Street
Santa Clara, CA 95051

Company confidential

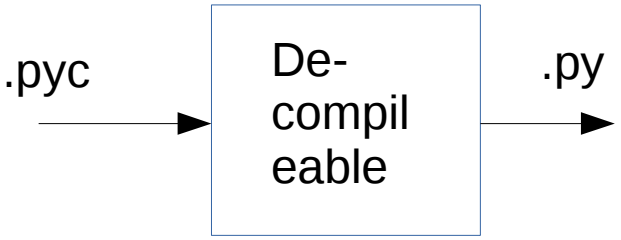
Protecting Python Sources With Cython

https://medium.com/@xpl/protecting-python-sources-using-cython-dcd940bb188e

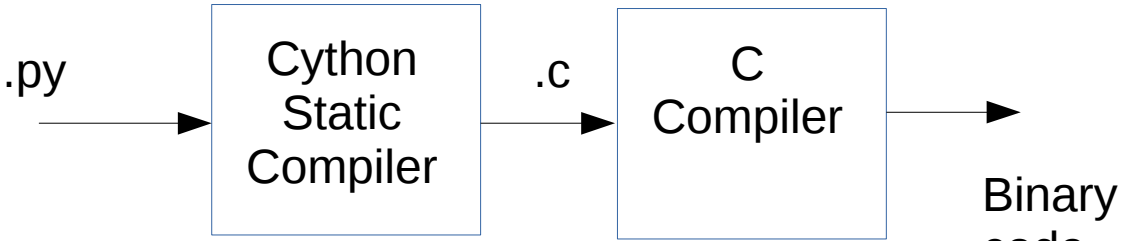
Protecting your Python sources from unwanted readers is easier said than done, because .pyc bytecode is decompileable and the obfuscation is easily reverse-engineered.



Vitaly Gordon



Reverse engineered



Can not be reverse engineered

“Cython, an optimizing static compiler that takes your .py modules and translates them to high-performant C files. Resulting C files can be compiled into native binary libraries with no effort. When the compilation is done there’s no way to reverse compiled libraries back to readable Python source code! Cython supports both Python 2 and 3, including the modern async/await syntax. From my experience, the only thing it couldn’t do is asynchronous generators.”

Steps for Cython

https://medium.com/@xpl/protecting-python-sources-using-cython-dcd940bb188e

1. Install Cython → **2. Add compile.py**

Acted as “makefile”

↓

3. Add main.py ← **4. Run compile.py**

Make the entry point Python file for your application. You will import and launch all the compiled logic from there.

The above command will generate .so and .c files next to your .py source files:

The .c files are intermediate sources used to generate .so files, binary modules to distribute.

Setting Up a Different OS Environment Using VirtualBox and Vagrant

Obviously, the compiled modules are not cross-platform. If you distribute your program to Ubuntu Linux users, compile it on Linux. Compile a platform-specific version of your code for each of your targeted platforms.