Pyarmor Documentation

Release 8.0.1

Jondy Zhao

Apr 10, 2023

Contents

Version 8.0.1

Homepage https://pyarmor.dashingsoft.com/

Contact pyarmor@163.com

Authors Jondy

CHAPTER 1

How the documentation is organized

Pyarmor has a lot of documentation. A high-level overview of how it's organized will help you know where to look for certain things:

- *Part 1: Tutorials* takes you by the hand through a series of steps to obfuscate *Python* scripts and packages. Start here if you're new to *Pyarmor*. Also look at the *Getting Started*
- *Part 2: How To* guides are recipes. They guide you through the steps involved in addressing key problems and use-cases. They are more advanced than tutorials and assume some knowledge of how *Python* works.
- *Part 3: References* guides contain key concepts, man page, configurations and other aspects of *Pyarmor* machinery.
- *Part 4: Topics* guides insight into key topics and provide useful background information and explanation. They describe how it works and how to use it but assume that you have a basic understanding of key concepts.
- *Part 5: Licneses* describes EULA of *Pyarmor*, the different *Pyarmor* licenses and how to purchase *Pyarmor* license.

CHAPTER 2

Getting help

Having trouble? We'd like to help!

Try the *FAQ* – it's got answers to many common questions.

Looking for specific information? Try the genindex, or *the detailed table of contents*.

Not found anything? See *asking questions in github*.

Report bugs with Pyarmor in issues

CHAPTER 3

Table of Contents

3.1 Tutorials

3.1.1 Getting Started

New to *Pyarmor*? Well, you came to the right place: read this material to quickly get up and running.

Content

What's Pyarmor

Pyarmor is a command line tool used to obfuscate *Python* scripts, bind obfuscated scripts to fixed machine or expire obfuscated scripts.

Key Features:

- The obfuscated scritpt is still a normal `.py` script, in most of cases the original python scripts can be replaced with obfuscated scripts seamlessly.
- Provide many options to obfuscate the scripts to balance security and performance
- Rename functions/methods/classes/variables/arguments, irreversible obfuscation
- Convert part of Python functions to C function, irreversible obfuscation
- Bind obfuscated scripts to fixed machine or expire obfuscted scripts
- Protect obfuscated scripts by Themida (Only for Windows)

Installation from PyPI

Pyarmor packages are published on the PyPI. The preferred tool for installing packages from PyPI is **`pip`**. This tool is provided with all modern versions of Python.

On Linux or MacOS, you should open your terminal and run the following command:

```
$ pip install -U pyarmor
```

On Windows, you should open Command Prompt (`Win-r` and type **`cmd`**) and run the same command:

```
C:\> pip install -U pyarmor
```

After installation, type **`pyarmor --version`** on the command prompt. If everything worked fine, you will see the version number for the Pyarmor package you just installed.

Obfuscating one script

Here it's the simplest command to obfuscate one script `foo.py`:

```
$ pyarmor gen foo.py
```

The command `gen` could be replaced with `g` or `generate`:

```
$ pyarmor g foo.py
$ pyarmor generate foo.py
```

This command generates an obfuscated script `dist/foo.py`, which is a valid Python script, run it by Python interpreter:

```
$ python dist/foo.py
```

Check all generated files in the default output path:

```
$ ls dist/
...    foo.py
...    pyarmor_runtime_000000
```

There is an extra Python package `pyarmor_runtime_000000`, which is required to run the obfuscated script.

Distributing the obfuscated script

Only copy `dist/foo.py` to another machine doesn't work, instead copy all the files in the `dist/`.

Why? It's clear after checking the content of `dist/foo.py`:

```
from pyarmor_runtime_000000 import __pyarmor__
__pyarmor__(__name__, __file__, ...)
```

Actually the obfuscaetd script can be taken as normal Python script with dependent package `pyarmor_runtime`, use it as it's not obfuscated.

Note: The obfuscated scripts could be run by Python interpreter without Pyarmor, DO NOT install Pyarmor in the *Target Device*

Obfuscating one package

Now let's do a package. `-O` is used to set output path `dist2` different from the default:

```
$ pyarmor gen -O dist2 src/mypkg
```

Check the output:

```
$ ls dist2/
...    mypkg
...    pyarmor_runtime_000000

$ ls dist2/mypkg/
...          __init__.py
```

All the obfuscated scripts in the `dist2/mypkg`, test it:

```
$ cd dist2/
$ python -C 'import mypkg'
```

If there are sub-packages, using `-r` to enable recursive mode:

```
$ pyarmor gen -O dist2 -r src/mypkg
```

Distributing the obfuscated package

Also it works to copy the whole path `dist2` to another machine. But it's not convience, the better way is using `-i` to generate all the required files inside package path:

```
$ pyarmor gen -O dist3 -r -i src/mypkg
```

Check the output:

```
$ ls dist3/
...    mypkg

$ ls dist3/mypkg/
```

(continues on next page)

(continued from previous page)

```
...           __init__.py
...           pyarmor_runtime_000000
```

Now everything is in the package path `dist3/mypkg`, just copy the whole path to any target machine.

Note: Comparing current `dist3/mypkg/__init__.py` with above section `dist2/mypkg/__init__.py` to understand more about obfuscated scripts

Expiring obfuscated scripts

It's easy to set expire date for obfuscated scripts by `-e`. For example, generate obfuscated script with the expire date to 30 days:

```
$ pyarmor gen -O dist4 -e 30 foo.py
```

Run the obfuscated scripts `dist4/foo.py` to verify it:

```
$ python dist4/foo.py
```

It checks network time, make sure your machine is connected to internet.

Let's use another form to set past date `2020-12-31`:

```
$ pyarmor gen -O dist4 -e 2020-12-31 foo.py
```

Now `dist4/foo.py` should not work:

```
$ python dist4/foo.py
```

If expire date has a leading `.`, it will check local time other than NTP server. For examples:

```
$ pyarmor gen -O dist4 -e .30 foo.py
$ pyarmor gen -O dist4 -e .2020-12-31 foo.py
```

For this form internet connection is not required in target machine.

Distributing the expired script is same as above, copy the whole directory `dist4/` to target machine.

Binding obfuscated scripts to device

Suppose got target machine hardware informations:

```
IPv4:                       128.16.4.10
Enternet Addr:              00:16:3e:35:19:3d
Hard Disk Serial Number:    HXS2000CN2A
```

Using `-e` to bind hardware information to obfuscated scripts. For example, bind `dist5/foo.py` to enternet address:

```
$ pyarmor gen -O dist5 -b 00:16:3e:35:19:3d foo.py
```

So `dist5/foo.py` only could run in target machine.

It's same to bind IPv4 and serial number of hard disk:

```
$ pyarmor gen -O dist5 -b 128.16.4.10 foo.py
$ pyarmor gen -O dist5 -b HXS2000CN2A foo.py
```

It's possible to combine some of them. For example:

```
$ pyarmor gen -O dist5 -b "00:16:3e:35:19:3d HXS2000CN2A" foo.py
```

Only both enternet address and hard disk are matched machine could run this obfuscated script.

Distributing scripts bind to device is same as above, copy the whole directory `dist5/` to target machine.

Packaging obfuscated scripts

Remeber again, the obfuscated script is normal Python script, use it as it's not obfuscated.

Suppose package `mypkg` structure like this:

```
projects/
└── src/
    └── mypkg/
        ├── __init__.py
        ├── utils.py
        └── config.json
```

First make output path `projects/dist6` for obfuscated package:

```
$ cd projects
$ mkdir dist6
```

Then copy package data files to output path:

```
$ cp -a src/mypkg dist6/
```

Next obfuscate scripts to overwrite all the `.py` files in `dist6/mypkg`:

```
$ pyarmor gen -O dist6 -i src/mypkg
```

The final output:

```
projects/
├── README.md
└── src/
    └── mypkg/
        ├── __init__.py
        ├── utils.py
        └── config.json
└── dist6/
    └── mypkg/
        ├── __init__.py
        ├── utils.py
        ├── config.json
        └── pyarmor_runtime_000000/__init__.py
```

Comparing with `src/mypkg`, the only difference is `dist6/mypkg` has an extra sub-package `pyarmor_runtime_000000`. The last thing is packaging `dist6/mypkg` as your prefer way.

New to Python packaging? Refer to Python Packaging User Guide

Something need to know

There is binary extension module `pyarmor_runtime` in extra sub-package `pyarmor_runtime_000000`, here it's package content:

```
$ ls dist6/mypkg/pyarmor_runtime_000000
...    __init__.py
...    pyarmor_runtime.so
```

Generally using binary extensions means the obfuscated scripts require `pyarmor_runtime` be created for different platforms, so they

- only works for platforms which provides pre-built binaries
- may not be compatible with different builds of CPython interpreter
- often will not work correctly with alternative interpreters such as PyPy, IronPython or Jython

For example, when obfuscating scripts by Python 3.8, they can't be run by Python 3.7, 3.9 etc.

Another disadvantage of relying on binary extensions is that alternative import mechanisms (such as the ability to import modules directly from zipfiles) often won't work for extension modules (as the dynamic loading mechanisms on most platforms can only load libraries from disk).

What to read next

There is a complete *installation* guide that covers all the possibilities:

- install pyarmor by source
- call pyarmor from Python script
- clean uninstallation

Next is *Basic Obfuscation*. It covers

- using more option to obfuscate script and package
- using outer file to store runtime key
- localizing runtime error messages
- packing obfuscated scripts and protect system packages

And then *Advanced Tutorials*, some of them are not available in trial pyarmor

- 2 irreversible obfuscation: RFT mode, BCC mode [pyarmor-pro]
- Customization error handler
- plugin and hooks
- runtime error internationalization
- cross platform, multiple platforms and multiple Python version

Also you may be instersting in this guide *Highest security and performace*

How the documentation is organized

Pyarmor has a lot of documentation. A high-level overview of how it's organized will help you know where to look for certain things:

- *Part 1: Tutorials* now you're reading.
- *Part 2: How To* guides are recipes. They guide you through the steps involved in addressing key problems and use-cases. They are more advanced than tutorials and assume some knowledge of how *Python* works.
- *Part 3: References* guides contain key concepts, man page, configurations and other aspects of *Pyarmor* machinery.
- *Part 4: Topics* guides insight into key topics and provide useful background information and explanation. They describe how it works and how to use it but assume that you have a basic understanding of key concepts.
- *Part 5: Licneses* describes EULA of *Pyarmor*, the different *Pyarmor* licenses and how to purchase *Pyarmor* license.

Looking for specific information? Try the genindex, or *the detailed table of contents*.

3.1.2 Installation

Contents

Installation from PyPI

Pyarmor packages are published on the PyPI. The preferred tool for installing packages from PyPI is **pip**. This tool is provided with all modern versions of Python.

On Linux or MacOS, you should open your terminal and run the following command:

```
$ pip install -U pyarmor
```

On Windows, you should open Command Prompt (`Win-r` and type **cmd**) and run the same command:

```
C:\> pip install -U pyarmor
```

After installation, type **pyarmor --version** on the command prompt. If everything worked fine, you will see the version number for the Pyarmor package you just installed.

Installation from PyPI also allows you to install the latest development release. You will not generally need (or want) to do this, but it can be useful if you see a possible bug in the latest stable release. To do this, use the `--pre` flag:

```
$ pip install -U --pre pyarmor
```

If you need generate obfuscated scripts to run in other platforms, install `pyarmor.runtime`:

```
$ pip install pyarmor.runtime
```

Installed command

- **pyarmor** is the main command to do everything. See *Man Page*.

Start Pyarmor by Python interpreter

pyarmor is same as the following command:

```
$ python -m pyarmor.cli
```

Using virtual environments

When installing Pyarmor using **pip**, use *virtual environments* which could isolate the installed packages from the system packages, thus removing the need to use administrator privileges. To create a virtual environment in the `.venv` directory, use the following command:

```
$ python -m venv .venv
```

You can read more about them in the Python Packaging User Guide.

Installation from source

You can install Pyarmor directly from a clone of the Git repository. This can be done either by cloning the repo and installing from the local clone, on simply installing directly via **git**:

```
$ git clone https://github.com/dashingsoft/pyarmor
$ cd pyarmor
$ pip install .
```

You can also download a snapshot of the Git repo in either tar.gz or zip format. Once downloaded and extracted, these can be installed with **pip** as above.

Run Pyarmor from Python script

Create a script `tool.py`, pass arguments by yourself

```
from pyarmor.cli.__main__ import main_entry

args = ['gen', 'foo.py']
main(args)
```

Run it by Python interpreter:

```
$ python tool.py
```

Clean uninstallation

Run the following commands to make a clean uninstallation:

```
$ pip uninstall pyarmor
$ rm -rf ~/.pyarmor
$ rm -rf ./.pyarmor
```

Note: The path ~ may be different when logging by different user. `$HOME` is home path of current logon user, check the environment variable `HOME` to get the real path.

3.1.3 Basic Obfuscation

Contents

We'll assume you have Pyarmor 8.0+ installed already. You can tell Pyarmor is installed and which version by running the following command in a shell prompt (indicated by the $ prefix):

```
$ pyarmor --version
```

If Pyarmor is installed, you should see the version of your installation. If it isn't, you'll get an error.

This tutorial is written for Pyarmor 8.0+, which supports Python 3.7 and later. If the Pyarmor version doesn't match, you can refer to the tutorial for your version of Pyarmor by using the version switcher at the bottom right corner of this page, or update Pyarmor to the newest version.

Throughout this tutorial, assume run **`pyarmor`** in project path which includes:

```
project/
    ├── foo.py
    ├── queens.py
    └── joker/
        ├── __init__.py
        ├── queens.py
        └── config.json
```

Pyarmor uses *pyarmor gen* with rich options to obfuscate scripts to meet the needs of different applications.

Here only introduces common options in a short, using any combination of them as needed. About usage of each option in details please refer to *pyarmor gen*

More options to protect script

For scripts, use these options to get more security:

```
$ pyarmor gen --enable-jit --mix-str --assert-call foo.py
```

Using *--enable-jit* tells Pyarmor processes some sentensive data by `c` function generated in runtime.

Using *--mix-str*[1] could mix the string constant (length > 4) in the scripts.

Using *--assert-call* makes sure function is obfuscated, to prevent called function from being replaced by special ways

For example,

```
data = "abcxyz"

def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b

if __name__ == '__main__':
    fib(n)
```

String constant `abcxyz` and function `fib` will be protected like this

```
data = __mix_str__(b"******")

def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b

if __name__ == '__main__':
    __assert_call__(fib)(n)
```

If function `fib` is obfuscated, `__assert_call__(fib)` returns original function `fib`. Otherwise it will raise protection exception.

More options to protect package

For package, append 2 extra options:

```
$ pyarmor gen --enable-jit --mix-str --assert-call --assert-import --restrict joker/
```

Using *--assert-import* prevents obfsucated modules from being replaced with plain script. It checks each import statement to make sure the modules are obfuscated.

Using *--restrict* makes sure the obfuscated module is only available inside package. It couldn't be imported from any plain script, also not be run by Python interpreter.

[1] *--mix-str* is not available in trial version

By default `__init__.py` is not restricted, in order to let others use your package functions, just import them in the `__init__.py`, then others could get exported functions in the public `__init__.py`.

In this test package, `joker/__init__.py` is an empty file, so module `joker.queens` is not exported. Let's check this, first create a script `dist/a.py`

```
import joker
print('import joker OK')
from joker import queens
print('import joker.queens OK')
```

Then run it:

```
$ cd dist
$ python a.py
... import joker OK
... RuntimeError: unauthorized use of script
```

In order to export `joker.queens`, edit `joker/__init__.py`, add one line

```
from joker import queens
```

Then do above test again, now it should work:

```
$ cd dist/
$ python a.py
... import joker OK
... import joker.queens OK
```

Checking runtime key periodically

Checking runtime key every hour:

```
$ pyarmor gen --period 1 foo.py
```

Binding to many machines

Using *-b* many times to bind obfuscated scripts to many machines.

For example, machine A and B, the ethernet addresses are `66:77:88:9a:cc:fa` and `f8:ff:c2:27:00:7f` respectively. The obfuscated script could run in both of machine A and B by this command

```
$ pyarmor gen -b "66:77:88:9a:cc:fa" -b "f8:ff:c2:27:00:7f" foo.py
```

Using outer file to store runtime key

First obfuscating script with *--outer*:

```
$ pyarmor gen --outer foo.py
```

In this case, it could not be run at this time:

```
$ python dist/foo.py
```

Let generate an outer runtime key valid for 3 days by this command:

```
$ pyarmor gen key -e 3
```

It generates a file `dist/pyarmor.rkey`, copy it to runtime package:

```
$ cp dist/pyarmor.rkey dist/pyarmor_runtime_000000/
```

Now run `dist/foo.py` again:

```
$ python dist/foo.py
```

Let's generate another license valid for 10 days:

```
$ pyarmor gen key -O dist/key2 -e 10

$ ls dist/key2/pyarmor.rkey
```

Copy it to runtime package to replace the original one:

```
$ cp dist/key2/pyarmor.rkey dist/pyarmor_runtime_000000/
```

The outer runtime key file also could be saved to other paths, but the file name must be `pyarmor.rkey`, here list the search order:

1. First search runtime package
2. Next search path *PYARMOR_RKEY*
3. Next search path `sys._MEIPASS`
4. Next search current path

If no found in these paths, raise runtime error and exits.

Localization runtime error

Some of runtime error messages could be customized. When something is wrong with the obfuscated scripts, it prints your own messages.

First create `messages.cfg` in the path `.pyarmor`:

```
$ mkdir .pyarmor
$ vi .pyarmor/message.cfg
```

Then edit it. It's a `.ini` format file, change the error messages as needed

```
[runtime.message]

  error_1 = this license key is expired
  error_2 = this license key is not for this machine
  error_3 = missing license key to run the script
  error_4 = unauthorized use of script
```

Now obfuscate the script in the current path to use customized messages:

```
$ pyarmor gen foo.py
```

If we want to show same message for all of license errors, edit it like this

```
[runtime.message]

  error_1 = invalid license key
  error_2 = invalid license key
  error_3 = invalid license key
```

Here no `error_4`, it means this error uses the default message.

And then obfuscate the scripts again.

Packing obfuscated scripts

Pyarmor need PyInstaller to pack scripts first, then replace plain scripts with obfuscated ones in bundle.

Packing to one file

First packing script to one file by PyInstaller with option `-F`:

```
$ pyinstaller -F foo.py
```

It generates one bundle file `dist/foo`, pass this to pyarmor:

```
$ pyarmor gen -O obfdist --pack dist/foo foo.py
```

This command will obfuscate `foo.py` first, then repack `dist/foo`, replace the original `foo.py` with `obfdist/foo.py`, and append all the runtime files to bundle.

The final output is still `dist/foo`:

```
$ dist/foo
```

Packing to one folder

First packing script to one foler by PyInstaller:

```
$ pyinstaller foo.py
```

It generates one bundle folder `dist/foo`, and an executable file `dist/foo/foo`, pass this executable to pyarmor:

```
$ pyarmor gen -O obfdist --pack dist/foo/foo foo.py
```

Like above section, `dist/foo/foo` will be repacked with obfuscated scripts.

Now run it:

```
$ dist/foo/foo
```

3.1.4 Advanced Tutorials

Contents

Using rftmode [pro]

RFT mode could rename most of builints, functions, classes, local variables. It equals rewritting scripts in source level.

For example, the following Python script

```
import sys

def sum2(a, b):
    return a + b

def main(msg):
    a = 2
    b = 6
    c = sum2(a, b)
    print('%s + %s = %d' % (a, b, c))

if __name__ == '__main__':
    main('pass: %s' % data)
```

will be reformed to

```
pyarmor__17 = __assert_armored__(b'\x83\xda\x03sys')

def pyarmor__22(a, b):
    return a + b

def pyarmor__16(msg):
    pyarmor__23 = 2
    pyarmor__24 = 6
    pyarmor__25 = pyarmor__22(pyarmor__23, pyarmor__24)
    pyarmor__14('%s + %s = %d' % (pyarmor__23, pyarmor__24, pyarmor__25))

if __name__ == '__main__':
    pyarmor__16('pass: %s' % pyarmor__20)
```

Using `--enable-rft` to enable RTF mode:

```
$ pyarmor gen --enable-rft foo.py
```

This feature is only available for *Pyarmor Pro*.

Using bccmode pro

BCC mode could convert most of functions and methods in the scripts to equivalent C functions, those c functions will be comipled to machine instructions directly, then called by obfuscated scripts.

Note that the code in model level is not converted to C function.

Using `--enable-bcc` to enable BCC mode:

```
$ pyarmor gen --enable-bcc foo.py
```

This feature is only available for *Pyarmor Pro*.

Customization error handler

By default when something is wrong with obfuscated scripts, a RuntimeError with error message is raised.

If prefer to show error message only:

```
$ pyarmor cfg on_error=1
```

If prefer to quit directly without any message:

```
$ pyarmor cfg on_error=2
```

Restore the default handler:

```
$ pyarmor cfg on_error=0
```

Or reset this option:

```
$ pyarmor cfg --reset on_error
```

After the option is changed, obfuscating the script again to make it effects.

Patching source by plugin marker

Before obfuscating a script, Pyarmor scans each line, remove plugin marker plus the following one whitespace, leave the rest as it is.

The default plugin marker is `# pyarmor:`, any comment line with this prefix will be as a plugin marker.

For example, these lines

```
print('start ...')

# pyarmor: print('this is plugin code')
# pyarmor: check_something()
```

will be changed to

```
print('start ...')

print('this is plugin code')
check_something()
```

One real case: protecting hidden imported modules

By default *--assert-import* could only protect modules imported by statement `import`, it doesn't handle modules imported by other methods.

For example,

```
m = __import__('abc')
```

In obfuscated script, there is a builtin function `__assert_armored__` could be used to check `m` is obfuscated. In order to make sure `m` could not be replaced by others, check it manually:

```
m = __import__('abc')
__assert_armored__(m)
```

But this results in a problem, The plain script could not be run because `__assert_armored__` is only available in the obfuscated script.

The plugin marker is right solution for this case. Let's make a little change

```
m = __import__('abc')
# pyarmor: __assert_armored__(m)
```

By plugin marker, both the plain script and the obfsucated script work as expected.

Using hooks

New in version 8.1: This feature is not implemented in 8.0

Hooks is used to do some extra checks when running obfuscated scripts.

A hook is a Python script called in any of

- boot: when importing the runtime package `pyarmor_runtime`
- period: only called when runtime key is in period mode
- import: when imporing an obfuscated module

An example of hook script `hook.py`

```
{
   'boot': '''def boot_hook(*args):
   print('hello, boot hook')''',

   'import': '''def import_hook(*args):
   print('hello, import hook')''',

   'period': '''def period_hook(*args):
   print('hello, period hook')''',
}
```

Save it to global or local configuration path

Internationalization runtime error message

Create `messages.cfg` in the path `.pyarmor`:

```
$ mkdir .pyarmor
$ vi .pyarmor/message.cfg
```

It's a `.ini` format file, add a section `runtime.message` with option `languages`. The language code is same as environment variable `LANG`, assume we plan to support 2 languages, and only customize 2 errors:

- error_1: license is expired
- error_2: license is not for this machine

```
[runtime.message]

languages = zh_CN zh_TW

error_1 = invalid license
error_2 = invalid license
```

error_1 and error_2 is default message for any non-matched language.

Now add 2 extra sections `runtime.message.zh_CN` and `runtime.message.zh_TW`

```
[runtime.message.zh_CN]

error_1 =
error_2 =

[runtime.message.zh_TW]

error_1 =
error_2 =
```

Then obfuscate script again to make it works.

PYARMOR_LANG could be used to set runtime language. If it's set, the obfuscated scripts ignore *LANG*.

Generating cross platform scripts

New in version 8.1: This feature is not implemented in 8.0

Use *--platform*

Obfuscating scripts for multiple Pythons

New in version 8.1: This feature is not implemented in 8.0

Use helper script *merge.py*

3.2 How To

3.2.1 Highest security and performace

Contents

3.2.2 Packaging data files

3.2.3 Obfuscating django app

3.2.4 Building obfuscated wheel

3.2.5 Packing with outer key

3.2.6 Protecting system packages

New in version 8.1: This feature is not implemented in 8.0

When packing the scripts, Pyarmor could also obfuscate system packages in the bundle.

3.2.7 Advanced Usage

Contents

Fix encoding error

Set script encoding:

```
$ pyarmor cfg encoding=utf-8
```

When customize runtime error message, set encoding of `messages.cfg`:

```
$ pyarmor cfg messages=messages.cfg:gbk
```

3.2.8 Register Pyarmor

Contents

Initial Registration

First read *Pyarmor License* to purchase one Pyarmor License.

An activation file like `pyarmor-regcode-xxxx.txt` will be sent to you by email. This file is used to initial registration.

At the first time to register Pyarmor, `-p` (product name) should be set. If not set, this Pyarmor license is bind to `TBD`, and could not be used for commercial product.

It need internet connection for intial registration.

For non-profits usage

For internal use or any non-profits use, run this command:

```
$ pyarmor reg pyarmor-regcode-xxxx.txt
```

For commercial usage

Assume this license is used to protect your product `Robot Studio`, initial registration by this command:

```
$ pyarmor reg -p "Robot Studio" pyarmor-regcode-xxxx.txt
```

Pyarmor will show registration information and ask your confirmation. If everything is fine, type `yes` and `Enter` to continue.

If initial registration is successful, it prints final license information in the console. And a registration file named `pyarmor-regfile-xxxx.zip` for this license is generated in the current path at the sametime. This file is used for next registration in other machines.

Activation file `pyarmor-keycode-xxxx.txt` can be uses only 10 times, after that it doesn't work. So once initial registration is successful, using registration file `pyarmor-regfile-xxxx.zip` for next registration.

Please keep this registration file carefully. If lost, Pyarmor is not responsible for keeping this license. In this case, if continue to use Pyarmor, needs purchase new one.

Once register successfully, product name can't be changed.

Product name is not decided

When product is in developing, and product name is not decide. Initial registration with product `TBD`. For example:

```
$ pyarmor reg -p "TBD" pyarmor-regcode-xxxx.txt
```

It can be changed once later, before product starts selling, the real name must be set by this command:

```
$ pyarmor reg -p "Robot Studio" pyarmor-regcode-xxxx.txt
```

Registeration in other machines

Once initial registeration successfully, it generates registration file named `pyarmor-regfile-xxxx.zip` at the same time.

Copy this file to other machines, then run the following command:

```
$ pyarmor reg pyarmor-regfile-xxxx.zip
```

It need not internet connection.

Check the registration information:

```
$ pyarmor -v
```

Upgrade Pyarmor from prior to 8.0

Refer to *upgrade old license*

3.3 References

3.3.1 Concepts

Activation File A text file used to initial registration *Pyarmor License*

When purchasing any *Pyarmor License*, an activation file is be sent to registration email after payment is completed.

BCC Mode An obfuscation method of Pyarmor by converting Python functions to C functions

Registration File A zip file generated after initial registration is successful. It's used to register *Pyarmor License* excpet initial registration.

Pyarmor Pyarmor is product domain, the goal is to provide functions and services to obfuscate Python scripts in high security and high performance. The mission of Pyarmor is let Python use easily in commercial product.

Pyarmor is composed of

- *Pyarmor Home*
- *pyarmor package*

Pyarmor Basic A *Pyarmor License* type

Pyarmor Group A *Pyarmor License* type

Pyarmor Home Host in github: https://github.com/dashingsoft/pyarmor/

It serves open source part of Pyarmor, issues and documentations.

Pyarmor License Issued by Pyarmor Team to unlock some limitations in Pyarmor trial version.

Refer to *Pyarmor License Types*

Pyarmor Package A *Python Package*, it includes

- *pyarmor*
- *pyarmor.cli*
- *pyarmor.cli.core*
- *pyarmor.cli.runtime*

Pyarmor Pro A *Pyarmor License* type

Python A program language.

Python Script A file that serves as an organizational unit of Python code.

Refer to https://docs.python.org/3.11/glossary.html#term-module

Python Package Refer to https://docs.python.org/3.11/glossary.html#term-package

RFT Mode An obfuscation method of Pyarmor by renaming function/class in the scripts

Runtime Files All the files required to run the obfuscated scripts.

Generally it equals *Runtime Package*. If *outer key* is used, plus this outer key file.

Runtime Key The settings of obfuscated scripts. It may include the expired date, device information of bind to obfuscated scripts. Also include all the flags to control the behaviours of obfuscated scripts.

Generally it's embedded into *Runtime Package*, but it also could be stored to a independent file *outer key*

Runtime Package A *Python Package* generally named `pyarmor_runtime_000000`.

When obfuscating the scripts, it's be generated at the same time.

It's required to run the obfuscated scripts.

Outer Key A file generally named `pyarmor.rkey` to store *Runtime Key*

The outer key file must be located in one of path

- *Runtime package*
- *PYARMOR_RKEY*
- `sys._MEIPASS`
- Current path

Home Path Store Pyarmor registration file, global configuration, other data file generated by **`pyarmor`**, the default path is user home path `~/.pyarmor`

Global Configuration Path Store Pyarmor global configuration file, default is `config/global` in the *Home Path*

Local Configuration Path Store Pyarmor local configuration file, default is `.pyarmor` in the current path

Registration File Path Store registration information of Pyarmor License, default is same as *Home Path*

Build Machine The device in which to install pyarmor, and to run pyarmor to generate obfuscated scripts.

Pyarmor Users Developers or organizations who use Pyarmor to obfuscate their Python scripts

Target Device In which run the obfuscated scripts distributed by *Pyarmor Users*, generally it's in customer side

Platform The standard platform name defined by Pyarmor. It's composed of os.arch.

Supported platforms list:

- **Windows**
  - windows.x86_64
  - windows.x86

- **Many Linuxs**
  - linux.x86_64
  - linux.x86
  - linux.aarch64
  - linux.armv7
- **Apple Intel and Silicon**
  - darwin.x86_64
  - darwin.aarch64

JIT Abbr. JUST-IN-TIME, just generating machine instructions in run time.

extension module A module written in C or C++, using Python's C API to interact with the core and with user code.

3.3.2 Man Page

Contents

Pyarmor is a powerful tool to obfuscate Python scripts with rich option set that provides both high-level operations and full access to internals.

pyarmor

Syntax

pyarmor [options] <command> ...

Options

-h, --help	show available command set then quit
-v, --version	show version information then quit
-q, --silent	suppress all normal output ...
-d, --debug	show more information in the console ...
--home PATH	set Pyarmor HOME path ...

These options can be used after **pyarmor** but before command, here are available commands:

gen	Obfuscate scripts
gen key	Generate outer runtime key
cfg	Show and configure environments
reg	Register Pyarmor

See **pyarmor <command> -h** for more information on a specific command.

Description

-q, --silent
Suppress all normal output.

For example:

```
pyarmor -q gen foo.py
```

-d, --debug
Show more information in the console

When something is wrong, print more debug informations in the console. For example:

```
pyarmor -d gen foo.py
```

--home PATH[,GLOBAL[,LOCAL[,REG]]]
Set Pyarmor *Home Path*, *Global Configuration Path*, *Local Configuration Path* and *Registration File Path*

The default paths

- *Home Path* is `~/.pyarmor`
- *Global Configuration Path* is `~/.pyarmor/config`, it's always relative to *Home Path*
- *Local Configuration Path* is `.pyarmor`
- *Registration File Path* is same as *Home Path*

All of them could be changed by this option. For example, change home path to `~/.pyarmor2`:

```
$ pyarmor --home ~/.pyarmor2 ...
```

Then

- *Global Configuration Path* is `~/.pyarmor2/config`
- *Registration File Path* is `~/.pyarmor2`
- *Local Configuration Path* still is `.pyarmor`

Another example, keep all others but change global path only:

```
$ pyarmor --home ,config2 ...
```

This command sets *Global Configuration Path* to `~/.pyarmor/config2`

Another example, keep all others but change local path only:

```
$ pyarmor --home ,,/var/myproject/ ...
```

This command sets *Local Configuration Path* to `/var/myproject`

Another example, set *Registration File Path* to `/opt/pyarmor/`:

```
$ pyarmor --home ,,,/opt/pyarmor ...
```

It's useful when may use **sudo** to run **pyarmor** occassionally. This makes sure the registration file could be found even switch to another user.

When there are many Pyarmor Licenses registerred in one machine, set each license to different *Registration File Path*

There are 2 solutions

The first solution, one license one home:

```
$ pyarmor --home ~/.pyarmor1 reg pyarmor-regfile-2051.zip
$ pyarmor --home ~/.pyarmor1 gen project1/foo.py

$ pyarmor --home ~/.pyarmor2 reg pyarmor-regfile-2052.zip
$ pyarmor --home ~/.pyarmor2 gen project2/foo.py
```

The second solution, same home, one license one path:

```
$ pyarmor --home ,,,pyarmor1 reg pyarmor-regfile-2051.zip
$ pyarmor --home ,,,pyarmor1 gen project1/foo.py

$ pyarmor --home ,,,pyarmor2 reg pyarmor-regfile-2052.zip
$ pyarmor --home ,,,pyarmor2 gen project2/foo.py
```

Start pyarmor with clean configuration by setting *Global Configuration Path* and *Local Configuration Path* to any non-exists path `x`:

```
$ pyarmor --home ,x,x, gen foo.py
```

See also:

PYARMOR_HOME

pyarmor gen

Generate obfuscated scripts and all the required runtime files.

Syntax

pyarmor gen <options> <SCRIPT or PATH>

Options

-h, --help show option list and help information then quit

-O PATH, --output PATH output path . . .

-r, --recursive search scripts in recursive mode . . .

-e DATE, --expired DATE set expired date . . .

-b DEV, --bind-device DEV bind obfuscated scripts to device . . .

--period N check runtime key periodically . . .

--outer enable outer runtime key . . .

--platform NAME cross platform obfuscation . . .

-i store runtime files inside package . . .

--prefix PREFIX import runtime package with PREFIX . . .

--obf-module <0,1> obfuscate whole module (default is 1) . . .

--obf-code <0,1> obfuscate each function (default is 1) . . .

--no-wrap disable wrap mode . . .

--enable <jit,rft,bcc,themida> enable different obfuscation features . . .

--mix-str protect string constant . . .

--private enable private mode for script . . .

--restrict enable restrict mode for package . . .

--assert-import assert module is obfuscated . . .

--assert-call assert function is obfuscated . . .

--pack BUNDLE repack bundle with obfuscated scripts . . .

Description

This command is used to obfuscate all the scripts and packages listed in the command line. For example:

```
pyarmor gen foo.py
pyarmor gen src/mypkg
pyarmor gen -r src/mypkg
pyarmor gen -r src/pkg1 src/pkg2 libs/dbpkg
pyarmor gen -r main.py src/*.py libs/utils.py libs/dbpkg
```

-O PATH, **--output** PATH

Set the output path for all the generated files, default is `dist`

-r, --recursive

When obfuscating package, search all scripts recursively. No this option, only the scripts in package path are obfuscated.

-i

When obfuscating package, store the runtime files inside package. For example:

```
$ pyarmor gen -r -i mypkg
```

The *runtime package* will be stored inside package `dist/mypkg`:

```
$ ls dist/
...      mypkg/

$ ls dist/mypkg/
...            pyarmor_runtime_000000/
```

Without this option, the output path is like this:

```
$ ls dist/
...      mypkg/
...      pyarmor_runtime_000000/
```

This option can't be used to obfuscate script.

--prefix PREFIX

Only used when obfuscating many packages at the same time and still store the runtime package inside package.

In this case, use this option to specify which package is used to store runtime package. For example:

```
$ pyarmor gen --prefix mypkg src/mypkg mypkg1 mypkg2
```

This command tells pyarmor to store runtime package inside `dist/mypkg`, and make `dist/mypkg1` and `dist/mypkg2` to import runtime package from `mypkg`.

Checking the content of `.py` files in output path to make it clear.

As a comparison, obfuscating 3 packages without this option:

```
$ pyarmor gen -O dist2 src/mypkg mypkg1 mypkg2
```

And check `.py` files in the path `dist2`.

-e DATE, **--expired** DATE
: Expired date of obfuscated scripts.

It supports 4 forms:

- A number stands for valid days
- A date with iso format `YYYY-MM-DD`
- A leading `.` with above 2 forms

Without leading dot, the obfuscated scripts checks NTP server time. For example:

```
$ pyarmor gen -e 30 foo.py
$ pyarmor gen -e 2022-12-31 foo.py
```

With leading dot, it checks local time. For example:

```
$ pyarmor gen -e .30 foo.py
$ pyarmor gen -e .2022-12-31 foo.py
```

-b DEV, **--bind-device** DEV
: Use this option multiple times to bind multiple machines

Bind obfuscated script to specified device. Now only harddisk serial number, ethernet address and IPv4 address are available.

For example:

```
$ pyarmor gen -b 128.16.4.10 foo.py
$ pyarmor gen -b 52:38:6a:f2:c2:ff foo.py
$ pyarmor gen -b HXS2000CN2A foo.py
```

Also set 30 valid days for this device:

```
$ pyarmor gen -e 30 -b 128.16.4.10 foo.py
```

Check all of hardware informations in this device:

```
$ pyarmor gen -b "128.16.4.10 52:38:6a:f2:c2:ff HXS2000CN2A" foo.py
```

Using this options multiple times means binding many machines. For example, the following command makes the obfuscated scripts could run 2 machiens:

```
$ pyarmor gen -b "52:38:6a:f2:c2:ff" -b "f8:ff:c2:27:00:7f" foo.py
```

In case there are more network cards, binding anyone by this form:

```
$ pyarmor gen -b "<2a:33:50:46:8f>" foo.py
```

Bind all network cards by this form:

```
$ pyarmor gen -b "<2a:33:50:46:8f,f0:28:69:c0:24:3a>" foo.py
```

In Linux, it's possible to bind named ethernet card:

```
$ pyarmor gen -b "eth1/fa:33:50:46:8f:3d" foo.py
```

If there are many harddisks. In Windows, binding anyone by sequence no:

```
$ pyarmor gen -b "/0:FV994730S6LLF07AY" foo.py
$ pyarmor gen -b "/1:KDX3298FS6P5AX380" foo.py
```

In Linux, binding to specify name:

```
$ pyarmor gen -b "/dev/vda2:KDX3298FS6P5AX380" foo.py
```

--period N

Check *Runtime Key* periodically.

Support units:

- s
- m
- h

The default unit is hour, for example, the following examples are equivalent:

```
$ pyarmor gen --period 1 foo.py
$ pyarmor gen --period 3600s foo.py
$ pyarmor gen --period 60m foo.py
$ pyarmor gen --period 1h foo.py
```

Note: If the obfuscated script enters an infinite loop without call any obfuscated function, it doesn't trigger periodic check.

--outer
Enable *outer key*

It tells the obfuscated scripts find *runtime key* in outer file.

Once this option is specified, *pyarmor gen key* must be used to generate an outer key file and copy to the corresponding path in *target device*. Otherwise the obfuscated scripts will complain of `missing license key to run the script`

The default name of outer key is `pyarmor.rkey`, it can be changed by this command:

```
$ pyarmor cfg outer_keyname=".pyarmor.key"
```

By this command the name of outer key is set to `.pyarmor.key`.

--platform NAME
Specify target platform to run obfuscated scripts.

The name must be one of standard *platform* defined by Pyarmor.

It requires *pyarmor.cli.runtime* to get prebuilt binary libraries of other platforms.

--private
Enable private mode for scripts.

When private mode is enabled, the function name is empty in traceback. And the obfuscated scripts could not be imported by plain script or Python interpreter.

It can't be used with *--restrict*, the latter enables private mode implicitly.

--restrict
Enable restirct mode for package, do not use it to obfuscate scripts.

It enables *--private* implicitly, and has all the features of private mode.

When restrict mode is enabled, all the modules excpet `__init__.py` in the package could not be imported by plain scripts.

For example, obfuscate a restrict package to `dist/joker`:

```
$ pyarmor gen -i --restrict joker
$ ls dist/
...    joker/
```

Then create a plaint script `dist/foo.py`

```
import joker
print('import joker should be OK')
from joker import queens
print('import joker.queens should fail')
```

Run it to verify:

```
$ cd dist
$ python foo.py
... import joker should be OK
... RuntimeError: unauthorized use of script
```

If there are extra modules need to be exported, list all the modules in this command:

```
$ pyarmor cfg exclude_restrict_modules="__init__ queens"
```

Then obfuscate the package again.

--obf-module <0,1>
Enable the whole module (default is 1)

--obf-code <0,1>
Enable each function in module (default is 1)

--no-wrap
Disable wrap mode

If wrap mode is enabled, when enter a function, it's restored. but when exit, this function will be obfuscated again.

If wrap mode is disabled, once the function is restored, it's never be obfuscated again.

If *--obf-code* is 0, this option is meaningless.

--enable <jit,rft,bcc,themida>
Enable different obfuscation features.

--enable-jit

Use *JIT* to process some sentensive data to improve security.

--enable-rft
Enable *RFT Mode* to obfuscate the script [pro]

--enable-bcc
Enable *BCC Mode* to obfuscate the script [pro]

--enable-themida
Use Themida to protect extension module in *runtime package*

Only works for Windows platform.

--mix-str
Mix the string constant in scripts [basic]

--assert-call
Assert function is obfuscated

If this option is enabled, Pyarmor scans each function call in the scripts. If the called function is in the obfuscated scripts, protect it as below, and leave others as it is. For example,

```
def fib(n):
    a, b = 0, 1
    return a, b

print('hello')
fib(n)
```

will be changed to

```
def fib(n):
    a, b = 0, 1

print('hello')
__assert_armored__(fib)(n)
```

The function `__assert_armored__` is a builtin function in obfuscated script. It checks the argument, if it's an obfuscated function, then returns this function, otherwise raises protection exception.

In this example, `fib` is protected, `print` is not.

--assert-import
Assert module is obfuscated

If this option is enabled, Pyarmor scans each `import` statement in the scripts. If the imported module is obfuscated, protect it as below, and leave others as it is. For example,

```
import sys
import foo
```

will be changed to

```
import sys
import foo
__assert_armored__(foo)
```

The function `__assert_armored__` is a builtin function in obfuscated script. It checks the argument, if it's an obfuscated module, then return this module, otherwise raises protection exception.

This option neither touchs statement `from import`, nor the module imported by function `__import__`.

`--pack` `BUNDLE`
Repack bundle with obfuscated scripts

Here `BUNDLE` is an executable file generated by PyInstaller

Pyarmor just obfuscates the script first.

Then unpack the bundle.

Next replace all the `.pyc` in the bundle with obfuscated scripts, and append all the *runtime files* to the bundle.

Finally repack the bundle and overwrite the original `BUNDLE`.

pyarmor gen key

Generate *outer key* for obfuscated scripts.

Syntax
pyarmor gen key <options>

Options

-O PATH, --output PATH output path

-e DATE, --expired DATE set expired date

--period N check runtime key periodically

-b DEV, --bind-device DEV bind obfuscated scripts to device

Description

This command is used to generate *outer key*, the options in this command have same meaning as in the *pyarmor gen*.

There must be at least one of option `-e` or `-b` for *outer key*.

It's invalid that outer key is neither expired nor binding to a device. For this case, don't use outer key.

By default the outer key is saved to `dist/pyarmor.rkey`. For example:

```
$ pyarmor gen key -e 30
$ ls dist/pyarmor.rkey
```

Save outer key to other path by this way:

```
$ pyarmor gen key -O dist/mykey2 -e 10
$ ls dist/mykey2/pyarmor.rkey
```

By default the outer key name is `pyarmor.rkey`, use the following command to change outer key name to any others. For example, `sky.lic`:

```
$ pyarmor cfg outer_keyname=sky.lic
$ pyarmor gen key -e 30
$ ls dist/sky.lic
```

pyarmor cfg

Configure or show Pyarmor environments

Syntax

pyarmor cfg <options> [OPT[=VALUE]] ...

Options

-h, --help	show this help message and exit
-p NAME	private settings for special module or package
-g, --global	do everything in global settings, otherwise local settings
-r, --reset	reset option to default value
--encoding ENCODING	specify encoding to read configuration file

Description

Run this command without arguments to show all available options:

```
$ pyarmor cfg
```

Show one exact option `obf_module`:

```
$ pyarmor cfg obf_module
```

Show all options which start with `obf`:

```
$ pyarmor cfg obf*
```

Set option to new value:

```
$ pyarmor cfg obf_module=0
```

Reset option to default:

```
$ pyarmor cfg -r obf_module
```

Change option `excludes` in the section `finder` by this form:

```
$ pyarmor cfg finder:excludes=ast
```

If no prefix `finder`, for example:

```
$ pyarmor cfg excludes=ast
```

Not only option `excludes` in section `finder`, but also in other sections `assert.call`, `mix.str` etc. are changed.

-p NAME
Private settings for special module or package

All the settings is only used for specified module *NAME*.

-g, --global
Do everything in global settings

Without this option, all the changed settings are soted in *Local Configuration Path*, generally it's `.pyarmor` in the current path. By this option, everything is stored in *Global Configuration Path*, generally it's `~/.pyarmor/config/global`

-r, --reset
Reset option to default value

pyarmor reg

Register Pyarmor or upgrade Pyarmor license

Syntax

pyarmor reg [OPTIONS] [FILENAME]

Options

-h, --help	show this help message and exit
-p NAME, --product NAME	license to this product
-u, --upgrade	upgrade Pyarmor license
-y, --confirm	register Pyarmor without asking for confirmation

Arguments

The `FILENAME` must be one of these forms:

- `pyarmor-regcode-xxxx.txt` got by purchasing Pyarmor license
- `pyarmor-regfile-xxxx.zip` got by initial registration with above file

Description

Check the registration information:

```
$ pyarmor -v
```

Show verbose information:

```
$ pyarmor reg
```

-p `NAME`**, --product** `NAME`
Set product name bind to license

When initial registration, use this option to set proudct name bind to license.

If no this option, the product name is set to `non-profits`.

It's meanless to use this option after initial registration.

`TBD` is a special product name. If product name is `TBD` at initial registration, the product name can be changed later.

For any other product name, it can't be changed any more.

-y, --confirm
In initial registration, without asking for confirmation

-u, --upgrade
Upgrade old license to Pyarmor 8.0 Licese

Important: Once initial registration successfully, `pyarmor-regcode-xxxx.txt` may not work again. Using registration file `pyarmor-regfile-xxxx.zip` for next registration instead.

PLEASE BACKUP registration file `pyarmor-regfile-xxxx.zip` carefully, Pyarmor doesn't provide lost-found service

---

Using registration file `pyarmor-regfile-xxxx.zip` to register Pyarmor in other machine.

Copy it to target device, then run this command:

```
$ pyarmor reg pyarmor-regfile-xxxx.zip
```

Environment Variables

The following environment variables only used in *Build Machine* when generating the obfuscated scripts, not in *Target Device*.

PYARMOR_HOME
: Same as `pyarmor --home`

It mainly used in the shell scrits to change Pyarmor settings. If `pyarmor --home` is set, this environment var is ignored.

PYARMOR_PLATFORM
: Set the right *Platform* to run **pyarmor**

It's mainly used in some platforms Pyarmor could not tell right but still works.

PYARMOR_CC
: Specify C compiler for bccmode

PYARMOR_CLI
: Only for compatible with old Pyarmor, ignore this if you don't use old command prior to 8.0

If you do not use new commands in Pyarmor 8.0, and prefer to only use old commands, set it to `7`, for example:

```
# In Linux
export PYARMOR_CLI=7
pyarmor -h

# Or
PYARMOR_CLI=7 pyarmor -h

# In Windows
set PYARMOR_CLI=7
pyarmor -h
```

It forces command **pyarmor** to use old cli directly.

Without it, **pyarmor** first try new cli, if the command line couldn't be parsed by new cli, fallback to old cli.

This only works for command **pyarmor**.

3.3.3 Environments

Building Device

Building device is to run **pyarmor** to geneate obfuscated scripts and all the other required files.

Supported Platforms:

- Windows
- Linux
- Darwin

Support Arches:

- x86_64

- aarch64
- i386
- aarch32
- armv7

Supported Pyton versions:

- Python 3.7 ~ Python 3.11

Command line options and environment variables are described in *Man Page*

Configuration files

There are 3 kinds of configuration files

- global: an ini file `~/.pyarmor/config/global`
- local: an ini file `.pyarmor/config`
- private: each module `foo` may has one ini file either `~/.pyarmor/foo.rules` or `.pyarmor/foo.rules`

Target Device

Target device is to run the obfuscated scripts.

Support platforms, arches and Python versions are same as *Building device*

`sys._MEIPASS`

Borrowed from PyInstaller, set search path for *outer key*.

`sys._PARLANG`

It's used to set language for runtime error message.

If it's set, *LANG* is ignored.

LANG
OS environment variable, used to select language for runtime error message.

PYARMOR_LANG
It's used to set language for runtime error message.

If it's set, both *LANG* and `sys._PARLANG` are ignored.

PYARMOR_RKEY
Set search path for *outer key*

3.3.4 Error Messages

Here list all the errors when running **pyarmor** or obfuscated scripts.

If something is wrong, search error message here to find the reason.

If no exact error message found, most likely it's not caused by Pyarmor, search it in google or any other search engine to find the solution.

For example, someone reports error `Operation did not complete successfully because the file contains a virus or is potentially unwanted software question`

It's caused by Windows Defender, not Pyarmor. I'm sure Pyarmor is safe, but it uses some technics which let anti-virtus tools makes wrong decision.

In most of case, the outer error is out of my control, for this example, the solutions what I thought of

1. Check documentation of Windows Defender
2. Ask question in MSDN
3. Google this error message

Building Errors

Here list all the errors when run **`pyarmor`** in building machine

- out of license

  Using any feature is not avaiable in trial version or current Pyarmor License.

  Refer to *License Types*

- not machine id

  This machine is not registered, or the hardware information is changed.

  Try to register Pyarmor again to fix it.

- query machine id failed

  Pyarmor need query harddisk serial number or mac address, if it could not get hardware information, it complains of this.

- unknown license type OLD

  You purchase old license for Pyarmor 7.x, here are *the latest licenses*

  If you prefer to use Pyarmor 7.x, please use `pyarmor-7` or downgrade pyarmor to 7.7.4

  If you prefer to use Pyarmor 8.0+, please refund this order if it's still not activated:

  - Email to Ordersupport@mycommerce.com with order information and ask for refund.
  - Or click FindMyOrder page to submit refund request

Runtime Errors

Here list all the errors when run the obfuscated scripts

- error code out of range
- this license key is expired
- this license key is not for this machine
- missing license key to run the script
- unauthorized use of script
- this Python version is not supported
- the script doesn't work in this system

- the format of obfuscated script is incorrect

  may caused by

  - the obfuscated script is made by other Pyarmor version
  - can not get the path of runtime package
- the format of obfuscated function is incorrect

3.4 Topics

3.4.1 Insight Into Obfuscation

3.4.2 Insight Into Obfuscated Script

3.4.3 Changed features by obfuscated scripts

3.4.4 Localization and Internationalization

3.4.5 Insight Into Pack Command

Pyarmor 8.0 has no command pack, but *--pack*. It could specify an executable file generated by PyInstaller:

```
pyinstaller foo.py
pyarmor gen --pack dist/foo/foo foo.py
```

If no this option, pyarmor only obfuscates the scripts.

If this option is set, pyarmor first obfuscates the scripts, then does extra work:

- Unpacking this executable to a temporary folder
- Replacing the scripts in bundle with obfuscated ones
- Appedning runtime files to the bundle in this temporary folder
- Repacking this temporary folder to an executable file and overwite the old

Packing obfuscated scripts manually

If something is wrong with *--pack*, or the final bundle doesn't work, try to pack the obfuscated scripts manually.

You need know how to using PyInstaller and using spec file, if not, learn it by yourself.

Here is an example to pack script `foo.py` in the path `/path/to/src`

- First obfuscating the script by Pyarmor[1]:

  ```
  cd /path/to/src
  pyarmor gen -O obfdist -a foo.py
  ```

- Moving runtime package to current path[2]:

[1] Do not use `-i` and `--prefix` to obfuscate the scripts

[2] Just let PyInstaller could find runtime package without extra pypath

```
mv obfdist/pyarmor_runtime_000000 ./
```

- Already have `foo.spec`, appending runtime package to `hiddenimports`

```
a = Analysis(
    ...
    hiddenimports=['pyarmor_runtime_000000'],
    ...
)
```

- Otherwise generating `foo.spec` by PyInstaller[3]:

```
pyi-makespec --hidden-import pyarmor_runtime_000000 foo.py
```

- Patching `foo.spec` by inserting extra code after `a = Analysis`

```
a = Analysis(
    ...
)

# Patched by PyArmor
_src = r'/path/to/src'
_obf = r'/path/to/src/obfdist'

_count = 0
for i in range(len(a.scripts)):
    if a.scripts[i][1].startswith(_src):
        x = a.scripts[i][1].replace(_src, _obf)
        if os.path.exists(x):
            a.scripts[i] = a.scripts[i][0], x, a.scripts[i][2]
            _count += 1
if _count == 0:
    raise RuntimeError('No obfuscated script found')

for i in range(len(a.pure)):
    if a.pure[i][1].startswith(_src):
        x = a.pure[i][1].replace(_src, _obf)
        if os.path.exists(x):
            if hasattr(a.pure, '_code_cache'):
                with open(x) as f:
                    a.pure._code_cache[a.pure[i][0]] = compile(f.read(), a.pure[i][1],
↪ 'exec')
            a.pure[i] = a.pure[i][0], x, a.pure[i][2]
# Patch end.

pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)
```

- Generating final bundle by this patched `foo.spec`:

```
pyinstaller foo.spec
```

If following this example, please

- Replacing all the `/path/to/src` with actual path
- Replacing all the `pyarmor_runtime_000000` with actual name

[3] Most of other PyInstaller options could be used here

notes

3.4.6 Insight Into RFT Mode

3.4.7 Insight Into BCC Mode

3.4.8 Performance

3.5 License Types

Contents

3.5.1 Introduction

This documentation is only apply to Pyarmor 8.0 plus.

Pyarmor is published as shareware, free trial version never expires, but there are some limitations:

a. Can not obfuscate big scritps[1]

b. Can not use feature mix-str[2] to obfuscate string constant in scripts

c. Can not use RFT Mode[3], BCC Mode[4]

d. Can not be used for any commercial product without permission

e. Can not be used to provide obfuscation service in any form

These limitations can be unlocked by different **'License Types'_** except last one.

3.5.2 License types

Pyarmor has 3 kind of licenses:

Pyarmor Basic Basic license could unlock big script[1] and mix-str[2] feature.

It requires internet connection to verify license

[1] Big Script means file size exceeds a cerntain value.
[2] Mix Str: obfscating string constant in script
[3] RFT Mode: renaming function/class/method/variable in Python scripts
[4] BCC Mode: Transforming some Python functions in scripts to c functions, compile them to machine instructions directly

Pyarmor Pro Pro license could unlock big script[1] and mix-str[2] feature.

Pro license also unlocks BCC Mode[4] and RFT Mode[3]

It requires internet connection to verify license

Pyarmor Group Group license unlocks all limitions and doesn't require internet.

Internet connection is only used to verify Pyarmor License in the build machine to generate the obfuscated scripts.

For the obfuscated scripts run in the customer's device, Pyarmor has no any limitions, it's totally controlled by users. Pyarmor only cares about build machine.

Each license has an unique number, the format is `pyarmor-vax-xxxxxx`, which x stands for a digital.

Each product requires one License No. So any product in global also has an unique number in Pyarmor world.

If user has many products, many license are required.

One product in Pyarmor world means a product name and everything that makes up this name.

It includes all the devices to develop, build, debug, test product.

It also includes product current version, history versions and all the future versions.

One product may has several variants, each variant name is composed of product name plus feature name. As long as the proportion of the variable part is far less than that of the common part, they're considered as "one product".

Pyarmor License could be installed in many machines and devices which belong to licensed product. But there is limitation to be used at the same time.

In 24 hours only less than 100 devices can use one same Pyarmor License. Pyarmor License be used means use any feature of Pyarmor in one machine. Running obfuscated scripts generated by Pyarmor is not considered as Pyarmor License be used.

In details read EULA of Pyarmor

License features

Table 1: Table-1. License Features

Features	Trial	Basic	Pro	Group	Remark
Basic Obfuscation	Y	Y	Y	Y	[5]
Expired Script	Y	Y	Y	Y	[6]
Bind Device	Y	Y	Y	Y	[7]
JIT Protection	Y	Y	Y	Y	[8]
Assert Protection	Y	Y	Y	Y	[9]
Themedia Protection	Y	Y	Y	Y	[10]
Big Script	No	Y	Y	Y	
Mix Str	No	Y	Y	Y	
RFT MODE	No	No	Y	Y	
BCC MODE	No	No	Y	Y	

[5] Basic Obfuscation: obfuscating the scripts by default options
[6] Expired Script: obfuscated scripts has expired date
[7] Bind Device: obfuscated scripts only run in specified devices
[8] JIT Protection: processing some sentensive data by runtime generated binary code
[9] Assert Protection: preventing others from hacking obfuscated scripts
[10] Themedia Protection: using Themedia to protect Widnows dlls

notes

3.5.3 Purchasing license

Open shopping cart in any web browser:

https://order.mycommerce.com/product?vendorid=200089125&productid=301044051

If you have Pyarmor 8.0+ installed, this command also could open shopping cart:

```
$ pyarmor reg --buy
```

In the shopping cart, select License Type and complete the payment online.

Please fill regname with personal or company name when placing order.

Table 2: Table-2. License Prices

License Type	Net Price($)	Remark
Basic	52	
Pro	89	
Group	158	

An activation file named `pyarmor-regcode-xxxx.txt` will be sent by email immediately after payment is completed successfully.

Following the guide in activation file to take the purchased license effects.

There are no additional license fees, apart from the cost of the license. And it only needs to be paid once, not periodically

Refund policy

If activation file isn't used, and purchasing date is in six months, refund is accepted. Please send request to pyarmor@163.com, Pyarmor will refund the order in a week. Out of six monthes, or activation file has been used to activate the license, refund request is not accepted.

Why no refund even if my PayPal account is hacked and someone else bought Pyarmor by this PayPal account?

Imaging you lost cash €100, someone else got it and buys a cloth, I don't think the shopper should refund money to you. It's same for money in PayPal, you lost money by yourself, the shopper should not bear loss because of your fault.

3.5.4 Upgrading old license

Not all the old license could be upgraded to latest version.

The old license could be upgraded to Pyarmor Basic freely only if it matchs these conditions:

- Following new EULA of Pyarmor
- The license no. starts with `pyarmor-vax-`
- The original activation file `pyarmor-regcode-xxxx.txt` is used not more than 100 times.

If it's not matched, please purchase new license to use Pyarmor latest version.

Upgrading to Pyarmor Pro needs extra fees.

Table 3: Table-3. Upgrade fee from old license

License Type	Upgrading fee($)	Remark
Basic	0	following new EULA and match some conditions
Pro	50	
Group	N/A	

Freely to Pyarmor-Basic

First find the activation file `pyarmor-regcode-xxxx.txt`, which is sent to registration email when purchasing the license.

In any build machine which has old license, first install Pyarmor 8.0+.

If no product name is set when purchasing old license, please decide which product will use this upgraded license. According to new EULA of Pyarmor, each license is only for one product.

Assume this license will be used to obfuscate product `Robot Studio`, run this command:

```
$ pyarmor reg -u -p "Robot Studio" pyarmor-regcode-xxxx.txt
```

If product name has been set when purchasing old license, run this command:

```
$ pyarmor reg -u pyarmor-regcode-xxxx.txt
```

If this license is only for non-profits use, run this command as above, in this case product name will be set to `TBD`:

```
$ pyarmor reg -u pyarmor-regcode-xxxx.txt
```

Check the upgraded license information:

```
$ pyarmor -v
```

If old license is used by many products (mainly old personal license), only one product could be used after upgrading. For the others, it need purchase new license.

With extra fee to Pyarmor-Pro

Open shopping cart in any web browser:

> https://order.mycommerce.com/product?vendorid=200089125&productid=301044051

If you have Pyarmor 8.0+ installed, this command also could open shopping cart:

```
$ pyarmor reg --buy
```

In the shopping cart, select `Pyarmor-upgrade` and complete the payment online.

An activation file named `pyarmor-regcode-to-pro.txt` will be sent by email immediately after payment is completed successfully.

Following the guide in activation file to take the purchased license effects.

3.6 FAQ

3.6.1 Asking Questions In Github

TBD

3.6.2 Purchasing and Registration

- Our company has a suite of products that we offer together or separately to our clients. Do we need a different license for each of them?

  Answer:

```
For a suite of products, if each product is different totally, for example,
a suite "Microsoft Office" includes "Microsoft Excel", "Microsoft Word",
each product need one license.

If a suite of products share most of Python scripts, as long as the
proportion of the variable part of each product is far less than that of the
common part, they're considered as "one product".

If each product in a suite of products is functionally complementary, for
example, product "Editor" for editing the file, product "Viewer" for
view the file, they're considered as "one product"
```

- How to refund my order?

  Answer:

```
If this key of this order isn't activated, you can refund the order
by one of ways
```

  1. Email to Ordersupport@mycommerce.com with order information and ask for refund.
  2. Or click FindMyOrder page to submit refund request

- I want to test obfs with version 8. Of course i want to buy your great product but i want to test if it is applicable with my current project. Is it possible to have 7 days demo?

  Answer:

```
Sorry, Pyarmor is a small tool and only cost small money, there is no demo
license plan.

Most of features could be verified in trial version, other advanced
features, for example, mix-str, bcc mode and rft mode, could be configured
to ignore one function or one script so that all the others could work with
these advanced features.
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

E