PyArmor Documentation

Release 5.9.0

Jondy Zhao

Mar 28, 2020

Contents

1 Installation **3**
- 1.1 Verifying the installation . . . 3
- 1.2 Installed commands . . . 3
- 1.3 Clean uninstallation . . . 4

2 Using PyArmor **5**
- 2.1 Obfuscating Python Scripts . . . 5
- 2.2 Distributing Obfuscated Scripts . . . 6
- 2.3 Generating License For Obfuscated Scripts . . . 6
- 2.4 Extending License Type . . . 7
- 2.5 Obfuscating Single Module . . . 7
- 2.6 Obfuscating Whole Package . . . 8
- 2.7 Packing Obfuscated Scripts . . . 8

3 Advanced Topics **9**
- 3.1 Obfuscating Many Packages . . . 9
- 3.2 Obfuscating Package No Conflict With Others . . . 10
- 3.3 Distributing Obfuscated Scripts To Other Platform . . . 10
- 3.4 Obfuscating Scripts By Other Version Of Python . . . 12
- 3.5 Run bootstrap code in plain scripts . . . 12
- 3.6 Let Python Interpreter Recognize Obfuscated Scripts Automatically . . . 13
- 3.7 Obfuscating Python Scripts In Different Modes . . . 14
- 3.8 Using Plugin to Extend License Type . . . 15
- 3.9 Bundle Obfuscated Scripts To One Executable File . . . 16
- 3.10 Bundle obfuscated scripts with customized spec file . . . 16
- 3.11 Improving The Security By Restrict Mode . . . 17
- 3.12 Checking Imported Function Is Obfuscated . . . 18
- 3.13 Call *pyarmor* From Python Script . . . 19
- 3.14 Check license periodly when the obfuscated script is running . . . 20
- 3.15 Work with Nuitka . . . 20

4 Examples **21**
- 4.1 Obfuscating and Packing PyQt Application . . . 21
- 4.2 Running obfuscated Django site with Apache and mod_wsgi . . . 22

5 Using Project **25**
- 5.1 Managing Obfuscated Scripts With Project . . . 25

5.2 Obfuscating Scripts With Different Modes . . . 26
5.3 Project Configuration File . . . 26

6 Man Page **31**
6.1 obfuscate . . . 32
6.2 licenses . . . 35
6.3 pack . . . 36
6.4 hdinfo . . . 39
6.5 init . . . 39
6.6 config . . . 40
6.7 build . . . 41
6.8 info . . . 42
6.9 check . . . 43
6.10 banchmark . . . 43
6.11 register . . . 44
6.12 download . . . 44
6.13 runtime . . . 45

7 Understanding Obfuscated Scripts **47**
7.1 Global Capsule . . . 47
7.2 Obfuscated Scripts . . . 47
7.3 Bootstrap Code . . . 48
7.4 Runtime Package . . . 48
7.5 The License File for Obfuscated Script . . . 49
7.6 Key Points to Use Obfuscated Scripts . . . 49
7.7 The Differences of Obfuscated Scripts . . . 50
7.8 About Third-Party Interpreter . . . 50

8 How PyArmor Does It **53**
8.1 How to Obfuscate Python Scripts . . . 53
8.2 How to Deal With Plugins . . . 54
8.3 Special Handling of Entry Script . . . 56
8.4 How to Run Obfuscated Script . . . 57
8.5 How To Pack Obfuscated Scripts . . . 59

9 Runtime Module *pytransform* **61**
9.1 Contents . . . 61
9.2 Examples . . . 62

10 Support Platfroms **63**
10.1 Standard Platform Names . . . 64
10.2 Platform Tables . . . 65

11 The Modes of Obfuscated Scripts **67**
11.1 Advanced Mode . . . 67
11.2 Obfuscating Code Mode . . . 68
11.3 Wrap Mode . . . 68
11.4 Obfuscating module Mode . . . 69
11.5 Restrict Mode . . . 69

12 The Performance of Obfuscated Scripts **73**

13 The Security of PyArmor **75**
13.1 Cross Protection for *_pytransform* . . . 75

14 When Things Go Wrong **79**
14.1 Segment fault . . . 80
14.2 Bootstrap Problem . . . 80
14.3 Obfuscating Scripts Problem . . . 81
14.4 Running Obfuscated Scripts Problem . . . 82
14.5 Packing Obfuscated Scripts Problem . . . 84
14.6 PyArmor Registration Problem . . . 85
14.7 Known Issues . . . 85
14.8 Misc. Questions . . . 85

15 License **87**
15.1 Purchase . . . 88

16 Change Logs **89**
16.1 5.9.5 . . . 89
16.2 5.9.4 . . . 89
16.3 5.9.3 . . . 90
16.4 5.9.2 . . . 90
16.5 5.9.1 . . . 90
16.6 5.9.0 . . . 90
16.7 5.8.9 . . . 91
16.8 5.8.8 . . . 91
16.9 5.8.7 . . . 91
16.10 5.8.6 . . . 91
16.11 5.8.5 . . . 91
16.12 5.8.4 . . . 92
16.13 5.8.3 . . . 92
16.14 5.8.2 . . . 92
16.15 5.8.1 . . . 92
16.16 5.8.0 . . . 92
16.17 5.7.10 . . . 92
16.18 5.7.9 . . . 93
16.19 5.7.8 . . . 93
16.20 5.7.7 . . . 93
16.21 5.7.6 . . . 93
16.22 5.7.5 . . . 93
16.23 5.7.4 . . . 93
16.24 5.7.3 . . . 93
16.25 5.7.2 . . . 94
16.26 5.7.1 . . . 94
16.27 5.7.0 . . . 94
16.28 5.6.8 . . . 95
16.29 5.6.7 . . . 95
16.30 5.6.6 . . . 95
16.31 5.6.5 . . . 95
16.32 5.6.4 . . . 96
16.33 5.6.3 . . . 96
16.34 5.6.2 . . . 96
16.35 5.6.1 . . . 96
16.36 5.6.0 . . . 96
16.37 5.5.7 . . . 97
16.38 5.5.6 . . . 97
16.39 5.5.5 . . . 97
16.40 5.5.4 . . . 97

16.41 5.5.3 . . . 97
16.42 5.5.2 . . . 97
16.43 5.5.1 . . . 98
16.44 5.5.0 . . . 98
16.45 5.4.6 . . . 98
16.46 5.4.5 . . . 98
16.47 5.4.4 . . . 99
16.48 5.4.3 . . . 99
16.49 5.4.2 . . . 99
16.50 5.4.1 . . . 99
16.51 5.4.0 . . . 99
16.52 5.3.13 . . . 99
16.53 5.3.12 . . . 99
16.54 5.3.11 . . . 99
16.55 5.3.10 . . . 100
16.56 5.3.9 . . . 100
16.57 5.3.8 . . . 100
16.58 5.3.7 . . . 100
16.59 5.3.6 . . . 100
16.60 5.3.5 . . . 100
16.61 5.3.4 . . . 100
16.62 5.3.3 . . . 100
16.63 5.3.2 . . . 101
16.64 5.3.1 . . . 101
16.65 5.3.0 . . . 101
16.66 5.2.9 . . . 101
16.67 5.2.8 . . . 101
16.68 5.2.7 . . . 101
16.69 5.2.6 . . . 102
16.70 5.2.5 . . . 102
16.71 5.2.4 . . . 102
16.72 5.2.3 . . . 102
16.73 5.2.2 . . . 102
16.74 5.2.1 . . . 102
16.75 5.2.0 . . . 102
16.76 5.1.2 . . . 103
16.77 5.1.1 . . . 103
16.78 5.1.0 . . . 103
16.79 5.0.5 . . . 103
16.80 5.0.4 . . . 104
16.81 5.0.3 . . . 104
16.82 5.0.2 . . . 104
16.83 5.0.1 . . . 104
16.84 4.6.3 . . . 104
16.85 4.6.2 . . . 105
16.86 4.6.1 . . . 105
16.87 4.6.0 . . . 105
16.88 4.5.5 . . . 105
16.89 4.5.4 . . . 105
16.90 4.5.3 . . . 105
16.91 4.5.2 . . . 105
16.92 4.5.1 . . . 105
16.93 4.5.0 . . . 105
16.94 4.4.2 . . . 106

16.95 4.4.2 . . . 106
16.96 4.4.1 . . . 106
16.97 4.4.0 . . . 106
16.98 4.3.4 . . . 106
16.99 4.3.3 . . . 106
16.100 4.3.2 . . . 106
16.101 4.3.1 . . . 107
16.102 4.3.0 . . . 107
16.103 4.2.3 . . . 107
16.104 4.2.2 . . . 107
16.105 4.2.1 . . . 107
16.106 4.1.4 . . . 107
16.107 4.1.3 . . . 108
16.108 4.1.2 . . . 108
16.109 4.1.1 . . . 108
16.110 4.0.3 . . . 108
16.111 4.0.2 . . . 108
16.112 4.0.1 . . . 108
16.113 3.9.9 . . . 108
16.114 3.9.8 . . . 108
16.115 3.9.7 . . . 109
16.116 3.9.6 . . . 109
16.117 3.9.5 . . . 109
16.118 3.9.4 . . . 109
16.119 3.9.3 . . . 109
16.120 3.9.2 . . . 109
16.121 3.9.1 . . . 109
16.122 3.9.0 . . . 109
16.123 3.8.10 . . . 110
16.124 3.8.9 . . . 110
16.125 3.8.8 . . . 110
16.126 3.8.7 . . . 110
16.127 3.8.6 . . . 110
16.128 3.8.5 . . . 110
16.129 3.8.4 . . . 111
16.130 3.8.3 . . . 111
16.131 3.8.2 . . . 111
16.132 3.8.1 . . . 111
16.133 3.8.0 . . . 111
16.134 3.7.5 . . . 111
16.135 3.7.4 . . . 111
16.136 3.7.3 . . . 111
16.137 3.7.2 . . . 111
16.138 3.7.1 . . . 112
16.139 3.7.0 . . . 112
16.140 3.6.2 . . . 112
16.141 3.6.1 . . . 112
16.142 3.6.0 . . . 112
16.143 3.5.1 . . . 112
16.144 3.5.0 . . . 112
16.145 3.4.3 . . . 113
16.146 3.4.2 . . . 113
16.147 3.4.1 . . . 113
16.148 3.4.0 . . . 113

16.149 3.3.1 . . . 113
16.150 3.3.0 . . . 113
16.151 3.2.1 . . . 114
16.152 3.2.0 . . . 114
16.153 3.1.7 . . . 114
16.154 3.1.6 . . . 114
16.155 3.1.5 . . . 114
16.156 3.1.4 . . . 114
16.157 3.1.3 . . . 115
16.158 3.1.2 . . . 115
16.159 3.1.1 . . . 115
16.160 3.0.1 . . . 115
16.161 2.6.1 . . . 116
16.162 2.5.5 . . . 116
16.163 2.5.4 . . . 116
16.164 2.5.3 . . . 116
16.165 2.5.2 . . . 116
16.166 2.5.1 . . . 116
16.167 2.4.1 . . . 116
16.168 2.3.4 . . . 117
16.169 2.3.3 . . . 117
16.170 2.3.2 . . . 117
16.171 2.3.1 . . . 117
16.172 2.2.1 . . . 117
16.173 2.1.2 . . . 117
16.174 2.1.1 . . . 117
16.175 2.0.1 . . . 117
16.176 1.7.7 . . . 118
16.177 1.7.6 . . . 118
16.178 1.7.5 . . . 118
16.179 1.7.4 . . . 118
16.180 1.7.3 . . . 118
16.181 1.7.2 . . . 118
16.182 1.7.1 . . . 118
16.183 1.7.0 . . . 119

17 Indices and tables **121**

Index **123**

Version PyArmor 5.9

Homepage http://pyarmor.dashingsoft.com/

Contact jondy.zhao@gmail.com

Authors Jondy Zhao

Copyright This document has been placed in the public domain.

PyArmor is a command line tool used to obfuscate python scripts, bind obfuscated scripts to fixed machine or expire obfuscated scripts. It protects Python scripts by the following ways:

- Obfuscate code object to protect constants and literal strings.
- Obfuscate co_code of each function (code object) in runtime.
- Clear f_locals of frame as soon as code object completed execution.
- Verify the license file of obfuscated scripts while running it.

PyArmor supports Python 2.6, 2.7 and Python 3.

PyArmor is tested against `Windows`, `Mac OS X`, and `Linux`.

PyArmor has been used successfully with `FreeBSD` and embedded platform such as `Raspberry Pi`, `Banana Pi`, `Orange Pi`, `TS-4600 / TS-7600` etc. but is not fullly tested against them.

Contents:

CHAPTER 1

Installation

PyArmor is a normal Python package. You can download the archive from PyPi, but it is easier to install using pip where is available, for example:

```
pip install pyarmor
```

or upgrade to a newer version:

```
pip install --upgrade pyarmor
```

There is also web ui for pyarmor, install it by this command:

```
pip install pyarmor-webui
```

1.1 Verifying the installation

On all platforms, the command `pyarmor` should now exist on the execution path. To verify this, enter the command:

```
pyarmor --version
```

The result should show `PyArmor Version X.Y.Z` or `PyArmor Trial Version X.Y.Z`.

If the command is not found, make sure the execution path includes the proper directory.

1.2 Installed commands

The complete installation places these commands on the execution path:

- `pyarmor` is the main command. See *Using PyArmor*.
- `pyarmor-webui` is used to open web ui of PyArmor.

If you do not perform a complete installation (installing via `pip`), these commands will not be installed as commands. However, you can still execute all the functions documented below by running Python scripts found in the distribution folder. The equivalent of the `pyarmor` command is *`pyarmor-folder`*`/pyarmor.py`.

`pyarmor-webui` is *`pyarmor-folder`*`/webui/server.py`.

1.3 Clean uninstallation

The following files are created by *pyarmor* after it has been installed:

```
~/.pyarmor_capsule.zip
~/.pyarmor/license.lic (since v5.8.0)
~/.pyarmor/platforms/

{pyarmor-folder}/license.lic (before v5.8.0)
```

Run the following commands to make a clean uninstallation:

```
pip uninstall pyarmor

rm -rf {pyarmor-folder}
rm -rf ~/.pyarmor_capsule.zip
rm -rf ~/.pyarmor
```

CHAPTER 2

Using PyArmor

The syntax of the `pyarmor` command is:

> `pyarmor` [*command*] [*options*]

2.1 Obfuscating Python Scripts

Use command *obfuscate* to obfuscate python scripts. In the most simple case, set the current directory to the location of your program `myscript.py` and execute:

```
pyarmor obfuscate myscript.py
```

PyArmor obfuscates `myscript.py` and all the `*.py` in the same folder:

- Create `.pyarmor_capsule.zip` in the `HOME` folder if it doesn't exists.
- Creates a folder `dist` in the same folder as the script if it does not exist.
- Writes the obfuscated `myscript.py` in the `dist` folder.
- Writes all the obfuscated `*.py` in the same folder as the script in the `dist` folder.
- Copy runtime files used to run obfuscated scripts to the `dist` folder.

In the `dist` folder the obfuscated scripts and all the required files are generated:

```
dist/
    myscript.py

    pytransform/
        __init__.py
        _pytransform.so, or _pytransform.dll in Windows, _pytransform.dylib in MacOS
        pytransform.key
        license.lic
```

The extra folder `pytransform` called *Runtime Package*, it's required to run the obfuscated script.

Normally you name one script on the command line. It's entry script. The content of `myscript.py` would be like this:

```
from pytransform import pyarmor_runtime
pyarmor_runtime()
__pyarmor__(__name__, __file__, b'\x06\x0f...')
```

The first 2 lines called *Bootstrap Code*, are only in the entry script. They must be run before using any obfuscated file. For all the other obfuscated `*.py`, there is only last line:

```
__pyarmor__(__name__, __file__, b'\x0a\x02...')
```

Run the obfuscated script:

```
cd dist
python myscript.py
```

By default, only the `*.py` in the same path as the entry script are obfuscated. To obfuscate all the `*.py` in the sub-folder recursively, execute this command:

```
pyarmor obfuscate --recursive myscript.py
```

2.2 Distributing Obfuscated Scripts

Just copy all the files in the output path *dist* to end users. Note that except the obfuscated scripts, the *Runtime Package* need to be distributed to end users too.

The *Runtime Package* may not with the obfuscated scripts, it could be moved to any Python path, only if *import pytransform* works.

About the security of obfuscated scripts, refer to *The Security of PyArmor*

Note: PyArmor need NOT be installed in the runtime machine

2.3 Generating License For Obfuscated Scripts

Use command *licenses* to generate new `license.lic` for obfuscated scripts.

By default there is `dist/pytransform/license.lic` generated by command *obfuscate*. It allows obfuscated scripts run in any machine and never expired.

Generate an expired license for obfuscated script:

```
pyarmor licenses --expired 2019-01-01 product-001
```

PyArmor generates new license file:

- Read data from `.pyarmor_capsule.zip` in the `HOME` folder
- Create `license.lic` in the `licenses/product-001` folder
- Create `license.lic.txt` in the `licenses/product-001` folder

Overwrite default license with new one:

```
cp licenses/code-001/license.lic dist/pytransform/
```

Run obfuscated script with new license, It will report error after Jan. 1, 2019:

```
cd dist
python myscript.py
```

Generate license to bind obfuscated scripts to fixed machine, first get hardware information:

```
pyarmor hdinfo
```

Then generate new license bind to harddisk serial number and mac address:

```
pyarmor licenses --bind-disk "100304PBN2081SF3NJ5T" --bind-mac "20:c1:d2:2f:a0:96"␣
↪code-002
```

Run obfuscated script with new license:

```
cp licenses/code-002/license.lic dist/pytransform/

cd dist/
python myscript.py
```

Note: Before v5.7.0, the default `license.lic` locates in the path `dist` other than `dist/pytransform`

2.4 Extending License Type

It's easy to extend any other licese type for obfuscated scripts: **just add authentication code in the entry script**. The script can't be changed any more after it is obfuscated, so do whatever you want in your script. In this case the *Runtime Module pytransform* would be useful.

The prefer way is *Using Plugin to Extend License Type*. The advantage is that your scripts needn't be changed at all. Just write authentication code in a separated script, and inject it in the obfuscated scripts as obfuscating. For more information, refer to *How to Deal With Plugins*

Here are some plugin examples

https://github.com/dashingsoft/pyarmor/tree/master/plugins

2.5 Obfuscating Single Module

To obfuscate one module exactly, use option `--exact`:

```
pyarmor obfuscate --exact foo.py
```

Only `foo.py` is obfuscated, now import this obfuscated module:

```
cd dist
python -c "import foo"
```

2.6 Obfuscating Whole Package

Run the following command to obfuscate a package:

```
pyarmor obfuscate --recursive --output dist/mypkg mykpg/__init__.py
```

To import the obfuscated package:

```
cd dist
python -c "import mypkg"
```

2.7 Packing Obfuscated Scripts

Use command `pack` to pack obfuscated scripts into the bundle.

First install *PyInstaller*:

```
pip install pyinstaller
```

Set the current directory to the location of your program `myscript.py` and execute:

```
pyarmor pack myscript.py
```

PyArmor packs `myscript.py`:

- Execute `pyarmor obfuscate` to obfuscate `myscript.py`
- Execute `pyinstaller myscipt.py` to create `myscript.spec`
- Update `myscript.spec`, replace original scripts with obfuscated ones
- Execute `pyinstaller myscript.spec` to bundle the obfuscated scripts

In the `dist/myscript` folder you find the bundled app you distribute to your users.

Run the final executeable file:

```
dist/myscript/myscript
```

Check the scripts have been obfuscated. It should return error:

```
rm dist/myscript/license.lic
dist/myscript/myscript
```

Generate an expired license for the bundle:

```
pyarmor licenses --expired 2019-01-01 code-003
cp licenses/code-003/license.lic dist/myscript

dist/myscript/myscript
```

For complicated cases, refer to command *pack* and *How To Pack Obfuscated Scripts*.

CHAPTER 3

Advanced Topics

3.1 Obfuscating Many Packages

There are 3 packages: *pkg1*, *pkg2*, *pkg2*. All of them will be obfuscated, and use shared runtime files.

First change to work path, create 3 projects:

```
mkdir build
cd build

pyarmor init --src /path/to/pkg1 --entry __init__.py pkg1
pyarmor init --src /path/to/pkg2 --entry __init__.py pkg2
pyarmor init --src /path/to/pkg3 --entry __init__.py pkg3
```

Then make the *Runtime Package*, save it in the path *dist*:

```
pyarmor build --output dist --only-runtime pkg1
```

Next obfuscate 3 packages, save them in the *dist*:

```
pyarmor build --output dist --no-runtime pkg1
pyarmor build --output dist --no-runtime pkg2
pyarmor build --output dist --no-runtime pkg3
```

Check all the output and test these obfuscated packages:

```
ls dist/

cd dist
python -c 'import pkg1
import pkg2
import pkg3'
```

Note: The runtime package `pytransform` in the output path *dist* also could be move to any other Python path, only if it could be imported.

From v5.7.2, the *Runtime Package* also could be generate by command *runtime* separately:

```
pyarmor runtime
```

3.2 Obfuscating Package No Conflict With Others

Note: New in v5.8.7

Suppose there are 2 packages obfuscated by different developers, could they be imported in the same Python interpreter?

If both of them are obfuscated by trial version of pyarmor, no problem, the answer is yes. But if anyone is obfuscated by registerred version, the answer is no.

Since v5.8.7, the scripts could be obfuscated with option `--enable-suffix` to generate the *Runtime Package* with an unique suffix, other than fixed name `pytransform`. For example:

```
pyarmor obfuscate --enable-suffix foo.py
```

The output would be like this:

```
dist/
    foo.py
    pytransform_vax_000001/
        __init__.py
        ...
```

The suffix `_vax_000001` is based on the registration code of PyArmor.

For project, set `enable-suffix` by command *config*:

```
pyarmor config --enable-suffix 0
pyarmor build -B
```

Or disable it by this way:

```
pyarmor config --enable-suffix 1
pyarmor build -B
```

3.3 Distributing Obfuscated Scripts To Other Platform

First list all the avaliable platform names by command *download*:

```
pyarmor download
pyarmor download --help-platform
```

Display the detials with option `--list`:

```
pyarmor download --list
pyarmor download --list windows
pyarmor download --list windows.x86_64
```

Then specify platform name when obfuscating the scripts:

```
pyarmor obfuscate --platform linux.armv7 foo.py

# For project
pyarmor build --platform linux.armv7
```

3.3.1 Obfuscating scripts with different features

There may be many available dynamic libraries for one same platform. Each one has different features. For example, both of `windows.x86_64.0` and `windows.x86_64.7` work in the platform `windwos.x86_64`. The last number stands for the features:

- 0: No anti-debug, JIT, advanced mode features, high speed
- 7: Include anti-debug, JIT, advanced mode features, high security

It's possible to obfuscate the scripts with special feature. For example:

```
pyarmor obfuscate --platform linux.x86_64.7 foo.py
```

Note that the dynamic library with different features aren't compatible. For example, try to obfuscate the scripts with `--platform linux.x86_64.0` in the Windows, the obfuscated scripts don't work in the target machine:

```
pyarmor obfuscate --platform linux.x86_64.0 foo.py
```

Because the full features dynamic library is used in the Windows by default. To fix this problem, set the enviornment variable `PYARMOR_PLATFORM` to same feature platform as target machine. For example:

```
PYARMOR_PLATFORM=windows.x86_64.0 pyarmor obfuscate --platform linux.x86_64.0 foo.py
```

3.3.2 Running Obfuscated Scripts In Multiple Platforms

From v5.7.5, the platform names are standardized, all the available platform names list here *Standard Platform Names*. And the obfuscated scripts could be run in multiple platforms.

In order to support multiple platforms, all the dynamic libraries for these platforms need to be copied to *Runtime Package*. For example, obfuscating a script could run in Windows/Linux/MacOS:

```
pyarmor obfuscate --platform windows.x86_64 \
                  --platform linux.x86_64 \
                  --platform darwin.x86_64 \
                  foo.py
```

The *Runtime Package* also could be generated by command *runtime* once, then obfuscate the scripts without runtime files. For examples:

```
pyarmor runtime --platform windows.x86_64,linux.x86_64,darwin.x86_64
pyarmor obfuscate --no-runtime --recursive \
                  --platform windows.x86_64,linux.x86_64,darwin.x86_64 \
                  foo.py
```

Because the obfuscated scripts will check the dynamic library, the platforms must be specified even if there is option `--no-runtime`. But if the option `--no-cross-protection` is specified, the obfuscated scripts will not check the dynamic library, so no platform is required. For example:

```
pyarmor obfuscate --no-runtime --recursive --no-cross-protection foo.py
```

Note: If the feature number is specified in one of platform, for example, one is `windows.x86_64.0`, then all the other platforms must be same feature.

Note: If the obfuscated scripts don't work in other platforms, try to update all the downloaded files:

```
pyarmor download --update
```

If it still doesn't work, try to remove the cahced platform files in the path `$HOME/.pyarmor`

3.4 Obfuscating Scripts By Other Version Of Python

If there are multiple Python versions installed in the machine, the command *pyarmor* uses default Python. In case the scripts need to be obfuscated by other Python, run *pyarmor* by this Python explicitly.

For example, first find `pyarmor.py`:

```
find /usr/local/lib -name pyarmor.py
```

Generally it should be in the */usr/local/lib/python2.7/dist-packages/pyarmor* in most of linux.

Then run pyarmor as the following way:

```
/usr/bin/python3.6 /usr/local/lib/python2.7/dist-packages/pyarmor/pyarmor.py
```

It's convenient to create a shell script */usr/local/bin/pyarmor3*, the content is:

```
/usr/bin/python3.6 /usr/local/lib/python2.7/dist-packages/pyarmor/pyarmor.py "$*"
```

And

```
chmod +x /usr/local/bin/pyarmor3
```

then use *pyarmor3* as before.

In the Windows, create a bat file *pyarmor3.bat*, the content would be like this:

```
C:\Python36\python C:\Python27\Lib\site-packages\pyarmor\pyarmor.py %*
```

3.5 Run bootstrap code in plain scripts

Before v5.7.0 the *Bootstrap Code* could be inserted into plain scripts directly, but now, for the sake of security, the *Bootstrap Code* must be in the obfuscated scripts. It need another way to run the *Bootstrap Code* in plain scripts.

First create one bootstrap package `pytransform_bootstrap` by command *runtime*:

```
pyarmor runtime -i
```

Next move bootstrap package to the path of plain script:

```
mv dist/pytransform_bootstrap /path/to/script
```

It also could be copied to python system library, for examples:

```
mv dist/pytransform_bootstrap /usr/lib/python3.5/ (For Linux)
mv dist/pytransform_bootstrap C:/Python35/Lib/ (For Windows)
```

Then edit the plain script, insert one line:

```
import pytransform_bootstrap
```

Now any other obfuscated modules could be imported after this line.

Note: Before v5.8.1, create this bootstrap package by this way:

```
echo "" > __init__.py
pyarmor obfuscate -O dist/pytransform_bootstrap --exact __init__.py
```

3.5.1 Run unittest of obfuscated scripts

In most of obfuscated scripts there are no *Bootstrap Code*. So the unittest scripts may not work with the obfuscated scripts.

Suppose the test script is `/path/to/tests/test_foo.py`, first patch this test script, refer to *run bootstrap code in plain scripts*

After that it works with the obfuscated modules:

```
cd /path/to/tests
python test_foo.py
```

The other way is patch system package `unittest` directly. Make sure the bootstrap package `pytransform_bootstrap` is copied in the Python system library, refer to *run bootstrap code in plain scripts*

Then edit `/path/to/unittest/__init__.py`, insert one line:

```
import pytransform_bootstrap
```

Now all the unittest scripts could work with the obfuscated scripts. It's useful if there are many unittest scripts.

3.6 Let Python Interpreter Recognize Obfuscated Scripts Automatically

In a few cases, if Python Interpreter could recognize obfuscated scripts automatically, it will make everything simple:

- Almost all the obfuscated scripts will be run as main script
- In the obfuscated scripts call *multiprocessing* to create new process

- Or call *Popen*, *os.exec* etc. to run any other obfuscated scripts
- ...

Here are the base steps:

1. First create one bootstrap package `pytransform_bootstrap`:

```
pyarmor runtime -i
```

   Before v5.8.1, it need be created by obfuscating an empty package:

```
echo "" > __init__.py
pyarmor obfuscate -O dist/pytransform_bootstrap --exact __init__.py
```

2. Then create virtual python environment to run the obfuscated scripts, move the bootstrap package to virtual python library. For example:

```
# For windows
mv dist/pytransform_bootstrap venv/Lib/

# For linux
mv dist/pytransform_bootstrap venv/lib/python3.5/
```

4. Edit *venv/lib/site.py* or *venv/lib/pythonX.Y/site.py*, import *pytransform_bootstrap* before the main line:

```
import pytransform_bootstrap

if __name__ == '__main__':
    ...
```

It also could be inserted into the end of function `main`, or anywhere they could be executed as module `site` is imported.

After that in the virtual environment `python` could run the obfuscated scripts directly, because the module `site` is automatically imported during Python initialization.

Refer to https://docs.python.org/3/library/site.html

Note: The command *pyarmor* doesn't work in this virtual environment, it's only used to run the obfuscated scripts.

Note: Before v5.7.0, you need create the bootstrap package by the *Runtime Files* manually.

3.7 Obfuscating Python Scripts In Different Modes

Advanced Mode is introduced from PyArmor 5.5.0, it's disabled by default. Specify option `--advanced` to enable it:

```
pyarmor obfuscate --advanced 1 foo.py

# For project
cd /path/to/project
pyarmor config --advanced 1
pyarmor build -B
```

From PyArmor 5.2, the default *Restrict Mode* is 1. It could be changed by the option `--restrict`:

```
pyarmor obfuscate --restrict=2 foo.py
pyarmor obfuscate --restrict=3 foo.py

# For project
cd /path/to/project
pyarmor config --restrict 4
pyarmor build -B
```

All the restricts could be disabled by this way if required:

```
pyarmor obfuscate --restrict=0 foo.py

# For project
pyarmor config --restrict=0
pyarmor build -B
```

The modes of *Obfuscating Code Mode*, *Wrap Mode*, *Obfuscating module Mode* could not be changed in command obfucate. They only could be changed by command *config* when *Using Project*. For example:

```
pyarmor init --src=src --entry=main.py .
pyarmor config --obf-mod=1 --obf-code=1 --wrap-mode=0
pyarmor build -B
```

3.8 Using Plugin to Extend License Type

PyArmor could extend license type for obfuscated scripts by plugin. For example, check internet time other than local time.

First create plugin script check_ntp_time.py. The key function in this script is *check_ntp_time*, the other important function is *_get_license_data* which used to get extra data from the *license.lic* of obfuscated scripts.

Then insert 2 comments in the entry script foo.py:

```
# {PyArmor Plugins}
# PyArmor Plugin: check_ntp_time()
```

Now obfuscate entry script:

```
pyarmor obfuscate --plugin check_ntp_time foo.py
```

If the plugin file isn't in the current path, use absolute path instead:

```
pyarmor obfuscate --plugin /usr/share/pyarmor/check_ntp_time foo.py
```

Finally generate one license file for this obfuscated script, pass extra license data by option `-x`, this data could be got by function *_get_license_data* in the plugin script:

```
pyarmor licenses -x 20190501 rcode-001
cp licenses/rcode-001/license.lic dist/
```

More examples, refer to https://github.com/dashingsoft/pyarmor/tree/master/plugins

About how plugins work, refer to *How to Deal With Plugins*

Important: The output function name in the plugin must be same as plugin name, otherwise the plugin will not take effects.

3.9 Bundle Obfuscated Scripts To One Executable File

Run the following command to pack the script *foo.py* to one executable file *dist/foo.exe*. Here *foo.py* isn't obfuscated, it will be obfuscated before packing:

```
pyarmor pack -e " --onefile" foo.py
dist/foo.exe
```

If you don't want to bundle the *license.lic* of the obfuscated scripts into the executable file, but put it outside of the executable file. For example:

```
dist/
    foo.exe
    license.lic
```

So that we could generate different licenses for different users later easily. Here are basic steps:

1. First create runtime-hook script *copy_licese.py*:

   ```
   import sys
   from os.path import join, dirname
   with open(join(dirname(sys.executable), 'license.lic'), 'rb') as fs:
       with open(join(sys._MEIPASS, 'license.lic'), 'wb') as fd:
           fd.write(fs.read())
   ```

2. Then pack the scirpt with extra options:

   ```
   pyarmor pack --clean --without-license -x " --exclude copy_license.py" \
           -e " --onefile --icon logo.ico --runtime-hook copy_license.py" foo.py
   ```

   Option `--without-license` tells *pack* not to bundle the *license.lic* of obfuscated scripts to the final executable file. By option `--runtime-hook` of PyInstaller, the specified script `copy_licesen.py` will be executed before any obfuscated scripts are imported. It will copy outer `license.lic` to right path.

   Try to run `dist/foo.exe`, it should report license error.

3. Finally run *licenses* to generate new license for the obfuscated scripts, and copy new `license.lic` and `dist/foo.exe` to end users:

   ```
   pyarmor licenses -e 2020-01-01 code-001
   cp license/code-001/license.lic dist/

   dist/foo.exe
   ```

3.10 Bundle obfuscated scripts with customized spec file

If there is a customized spec file works, for example:

```
pyinstaller myscript.spec
```

It could be used to pack obfuscated scripts by little changed:

- Add module `pytransform` to *hiddenimports*
- Add extra path `DISTPATH/obf/temp` to *pathex* and *hookspath*

After changed, it may be like this:

```
a = Analysis(['myscript.py'],
             pathex=[os.path.join(DISTPATH, 'obf', 'temp'), ...],
             binaries=[],
             datas=[],
             hiddenimports=['pytransform', ...],
             hookspath=[os.path.join(DISTPATH, 'obf', 'temp'), ...],
```

Now run command *pack* by this way:

```
pyarmor pack -s myscript.spec myscript.py
```

That's all.

Note: This featuer is introduced since v5.8.0

Before v5.8.2, the extra path is `DISTPATH/obf`, not `DISTPATH/obf/temp`

3.11 Improving The Security By Restrict Mode

By default the scripts are obfuscated by restrict mode 1, that is, the obfuscated scripts can't be changed. In order to improve the security, obfuscating the scripts by restrict mode 2 so that the obfuscated scripts can't be imported out of the obfuscated scripts. For example:

```
pyarmor obfuscate --restrict 2 foo.py
```

Or obfuscating the scripts by restrict mode 3 for more security. It will even check each function call to be sure all the functions are called in the obfuscated scripts. For example:

```
pyarmor obfuscate --restrict 3 foo.py
```

However restrict mode 2 and 3 aren't applied to Python package. There is another solutiion for Python package to improve the security:

- The *.py* files which are used by outer scripts are obfuscated by restrice mode 1
- All the other *.py* files which are used only in the package are obfuscated by restrict mode 4

For example:

```
cd /path/to/mypkg
pyarmor obfuscate --exact __init__.py exported_func.py
pyarmor obfuscate --restrict 4 --recursive \
        --exclude __init__.py --exclude exported_func.py .
```

More information about restrict mode, refer to *Restrict Mode*

3.12 Checking Imported Function Is Obfuscated

Sometimes it need to make sure the imported functions from other module are obfuscated. For example, there are 2 scripts *main.py* and *foo.py*:

```
$ cat main.py

import foo

def start_server():
    foo.connect('root', 'root password')

$ cat foo.py

def connect(username, password):
    mysql.dbconnect(username, password)
```

In the obfuscated *main.py*, it need to be sure *foo.connect* is obfuscated. Otherwise the end users may replace the obfuscated *foo.py* with this plain code:

```
def connect(username, password):
    print('password is %s', password)
```

One solution is to check imported functions by decorator *assert_armored* in the *main.py*. For example:

```
import foo

def assert_armored(*names):
    def wrapper(func):
        def _execute(*args, **kwargs):
            for s in names:
                # For Python2
                # if not (s.func_code.co_flags & 0x20000000):
                # For Python3
                if not (s.__code__.co_flags & 0x20000000):
                    raise RuntimeError('Access violate')
                # Also check a piece of byte code for special function
                if s.__name__ == 'connect':
                    if s.__code__.co_code[10:12] != b'\x90\xA2':
                        raise RuntimeError('Access violate')
            return func(*args, **kwargs)
        return _execute
    return wrapper

@ assert_armored(foo.connect, foo.connect2)
def start_server():
    foo.connect('root', 'root password')
    foo.connect2('user', 'user password')
```

3.12.1 Plugin Implementation

First write a plugin script *asser_armored.py*:

```
def assert_armored(*names):
    def wrapper(func):
```

(continues on next page)

(continued from previous page)

```
        def _execute(*args, **kwargs):
            for s in names:
                # For Python2
                # if not (s.func_code.co_flags & 0x20000000):
                # For Python3
                if not (s.__code__.co_flags & 0x20000000):
                    raise RuntimeError('Access violate')
                # Also check a piece of byte code for special function
                if s.__name__ == 'connect':
                    if s.__code__.co_code[10:12] != b'\x90\xA2':
                        raise RuntimeError('Access violate')
            return func(*args, **kwargs)
        return _execute
    return wrapper
```

Then edit *main.py* , insert plugin markers. For examples:

```
import foo

# {PyArmor Plugins}

# PyArmor Plugin:  @assert_armored(foo.connect, foo.connect2)
def start_server():
    foo.connect('root', 'root password')
    ...
```

So the original script could be run normally when it's not obfuscated. Only when it's distributed, obfuscating the script with this plugin:

```
pyarmor obfuscate --plugin assert_armored main.py
```

Note: After v5.7.2, if you prefer, the marker could be this form:

```
# @pyarmor_assert_armored(foo.connect, foo.connect2)
```

3.13 Call *pyarmor* From Python Script

It's also possible to call PyArmor methods inside Python script not by *os.exec* or *subprocess.Popen* etc. For example

```
from pyarmor.pyarmor import main as call_pyarmor
call_pyarmor(['obfuscate', '--recursive', '--output', 'dist', 'foo.py'])
```

In order to suppress all normal output of pyarmor, call it with `--silent`

```
from pyarmor.pyarmor import main as call_pyarmor
call_pyarmor(['--silent', 'obfuscate', '--recursive', '--output', 'dist', 'foo.py'])
```

From v5.7.3, when *pyarmor* called by this way and something is wrong, it will raise exception other than call *sys.exit*.

3.14 Check license periodly when the obfuscated script is running

Generally only at the startup of the obfuscated scripts the license is checked. Since v5.9.3, it also could check the license per hour. Just generate a new license with `--enable-period-mode` and overwrite the default one. For example:

```
pyarmor obfuscate foo.py
pyarmor licenses --enable-period-mode code-001
cp licenses/code-001/license.lic ./dist
```

3.15 Work with Nuitka

Because the obfuscated scripts could be taken as normal scripts with an extra runtime package *pytransform*, they also could be translated to C program by Nuitka. When obfuscating the scprits, the option `--restrict 0` and `--no-cross-protection` should be set, otherwise the final C program could not work. For example, first obfustate the scripts:

```
pyarmor obfuscate --restrict 0 --no-cross-protection foo.py
```

Then translate the obfuscated one as normal python scripts by Nuitka:

```
cd ./dist
python -m nuitka --include-package pytransform foo.py
./foo.bin
```

There is one problem is that the imported modules (packages) in the obfuscated scripts could not be seen by Nuitka. To fix this problem, first generate the corresponding `.pyi` with original script, then copy it within the obfuscated one. For example:

```
# Generating "mymodule.pyi"
python -m nuitka --module mymodule.py

pyarmor obfuscate --restrict 0 --no-bootstrap mymodule.py
cp mymodule.pyi dist/

cd dist/
python -m nuitka --module mymodule.py
```

But it may not take advantage of Nuitka features by this way, because most of byte codes aren't translated to c code indeed.

Note: So long as the C program generated by Nuitka is linked against libpython to execute, pyarmor could work with Nuitka. But in the future, just as said in the Nuitka official website:

```
It will do this - where possible - without accessing libpython but in C
with its native data types.
```

In this case, pyarmor maybe not work with Nuitka.

CHAPTER 4

Examples

Here are some examples.

4.1 Obfuscating and Packing PyQt Application

There is a tool *easy-han* based on PyQt. Here list the main files:

```
config.json

main.py
ui_main.py
readers/
    __init__.py
    msexcel.py

tests/
vnev/py36
```

Here the shell script used to pack this tool by PyArmor:

```
cd /path/to/src
pyarmor pack -e " --name easy-han --hidden-import comtypes --add-data 'config.json;.'
↪" \
             -x " --exclude vnev --exclude tests" -s "easy-han.spec" main.py

cd dist/easy-han
./easy-han
```

By option `-e` passing extra options to run PyInstaller, to be sure these options work with PyInstaller:

```
cd /path/to/src
pyinstaller --name easy-han --hidden-import comtypes --add-data 'config.json;.' main.
↪py
```

(continues on next page)

(continued from previous page)

```
cd dist/easy-han
./easy-han
```

By option `-x` passing extra options to obfuscate the scripts, there are many *.py* files in the path *tests* and *vnev*, but all of them need not to be obfuscated. By passing option `--exclude` to exclude them, to be sure these options work with command *obfuscate*:

```
cd /path/to/src
pyarmor obfuscate --exclude vnev --exclude tests main.py
```

By option `-s` to specify the *.spec* filename, because PyInstaller changes the default filename of *.spec* by option `--name`, so it tell command *pack* the right filename.

Important: The command *pack* will obfuscate the scripts automatically, do not try to pack the obfuscated the scripts.

Note: From PyArmor 5.5.0, it could improve the security by passing the obfuscated option `--advanced` to enable *Advanced Mode*. For example:

```
pyarmor pack -x " --advanced 1 --exclude tests" foo.py
```

4.2 Running obfuscated Django site with Apache and mod_wsgi

Here is a simple site of Django:

```
/path/to/mysite/
    db.sqlite3
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
    polls/
        __init__.py
        admin.py
        apps.py
        migrations/
            __init__.py
        models.py
        tests.py
        urls.py
        views.py
```

First obfuscating all the scripts:

```
# Create target path
mkdir -p /var/www/obf_site

# Copy all files to target path, because pyarmor don't deal with any data files
```

(continues on next page)

(continued from previous page)

```
cp -a /path/to/mysite/* /var/www/obf_site/

cd /path/to/mysite

# Obfuscating all the scripts in the current path recursively, specify the entry␣
↪script "wsgi.py"
# The obfuscate scripts will be save to "/var/www/obf_site"
pyarmor obfuscate --src="." -r --output=/var/www/obf_site mysite/wsgi.py
```

Then edit the server configuration file of Apache:

```
WSGIScriptAlias / /var/www/obf_site/mysite/wsgi.py
WSGIPythonHome /path/to/venv

# The runtime files required by pyarmor are generated in this path
WSGIPythonPath /var/www/obf_site

<Directory /var/www/obf_site/mysite>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>
```

Finally restart Apache:

```
apachectl restart
```

CHAPTER 5

Using Project

Project is a folder include its own configuration file, which used to manage obfuscated scripts.

There are several advantages to manage obfuscated scripts by Project:

- Increment build, only updated scripts are obfuscated since last build
- Filter obfuscated scripts in the project, exclude some scripts
- Obfuscate the scripts with different modes
- More convenient to manage obfuscated scripts

5.1 Managing Obfuscated Scripts With Project

Use command *init* to create a project:

```
cd examples/pybench
pyarmor init --entry=pybench.py
```

It will create project configuration file `.pyarmor_config` in the current path. Or create project in another path:

```
pyarmor init --src=examples/pybench --entry=pybench.py projects/pybench
```

The project path *projects/pybench* will be created, and `.pyarmor_config` will be saved there.

The common usage for project is to do any thing in the project path:

```
cd projects/pybench
```

Show project information:

```
pyarmor info
```

Obfuscate all the scripts in this project by command *build*:

```
pyarmor build
```

Change the project configuration by command *config*.

For example, exclude the `dist`, `test`, the *.py* files in these folder will not be obfuscated:

```
pyarmor config --manifest "include *.py, prune dist, prune test"
```

Force rebuild:

```
pyarmor build --force
```

Run obfuscated script:

```
cd dist
python pybench.py
```

After some scripts changed, just run *build* again:

```
cd projects/pybench
pyarmor build
```

5.2 Obfuscating Scripts With Different Modes

First configure the different modes, refer to *The Modes of Obfuscated Scripts*:

```
pyarmor config --obf-mod=1 --obf-code=0
```

Then obfuscating scripts in new mode:

```
pyarmor build -B
```

5.3 Project Configuration File

Each project has a configure file. It's a json file named `.pyarmor_config` stored in the project path.

- name

  Project name.

- title

  Project title.

- src

  Base path to match files by manifest template string.

  It could be absolute path, or relative path based on project folder.

- manifest

  A string specifies files to be obfuscated, same as MANIFEST.in of Python Distutils, default value is:

  ```
  global-include *.py
  ```

It means all files anywhere in the *src* tree matching.

Multi manifest template commands are spearated by comma, for example:

```
global-include *.py, exclude __mainfest__.py, prune test
```

Refer to https://docs.python.org/2/distutils/sourcedist.html#commands

- is_package

  Available values: 0, 1, None

  When it's set to 1, the basename of *src* will be appended to *output* as the final path to save obfuscated scripts, but runtime files are still in the path *output*

  When init a project and no `--type` specified, it will be set to 1 if there is *__init__.py* in the path *src*, otherwise it's None.

- restrict_mode

  Available values: 0, 1, 2, 3, 4

  By default it's set to 1.

  Refer to *Restrict Mode*

- entry

  A string includes one or many entry scripts.

  When build project, insert the following bootstrap code for each entry:

```
from pytransform import pyarmor_runtime
pyarmor_runtime()
```

  The entry name is relative to *src*, or filename with absolute path.

  Multi entries are separated by comma, for example:

```
main.py, another/main.py, /usr/local/myapp/main.py
```

  Note that entry may be NOT obfuscated, if *manifest* does not specify this entry.

- output

  A path used to save output of build. It's relative to project path.

- capsule

  Warning: Removed since v5.9.0

  Filename of project capsule. It's relative to project path if it's not absolute path.

- obf_code

  How to obfuscate byte code of each code object, refer to *Obfuscating Code Mode*:

  – 0

  No obfuscate

  – 1 (Default)

  Obfuscate each code object by default algorithm

  - 2

  Obfuscate each code object by more complex algorithm

- wrap_mode

  Available values: 0, 1, None

  Whether to wrap code object with *try..final* block.

  The default value is *1*, refer to *Wrap Mode*

- obf_mod

  How to obfuscate whole code object of module, refer to *Obfuscating module Mode*:

  - 0

  No obfuscate

  - 1 (Default)

  Obfuscate byte-code by DES algorithm

- cross_protection

  How to proect dynamic library in obfuscated scripts:

  - 0

  No protection

  - 1

  Insert proection code with default template, refer to *Special Handling of Entry Script*

  - Filename

  Read the template of protection code from this file other than default template.

- runtime_path

  None or any path.

  When run obfuscated scripts, where to find dynamic library *_pytransform*. The default value is None, it means it's within the *Runtime Package* or in the same path of `pytransform.py`.

  It's useful when obfuscated scripts are packed into a zip file, for example, use py2exe to package obfuscated scripts. Set runtime_path to an empty string, and copy *Runtime Files* to same path of zip file, will solve this problem.

- plugins

  None or list of string

  Extend license type of obfuscated scripts, multi-plugins are supported. For example:

```
plugins: ["check_ntp_time", "show_license_info"]
```

  About the usage of plugin, refer to *Using Plugin to Extend License Type*

- package_runtime

  How to save the runtime files:

  - 0

  Save them in the same path with the obufscated scripts

  - 1 (Default)

Save them in the sub-path *pytransform* as a package

- enable_suffix

  Note: New in v5.8.7

  How to generate runtime package (module) and bootstrap code, it's useful as importing the scripts obfuscated by different developer:

  – 0 (Default)

  There is no suffix for the name of runtime package (module)

  – 1

  The name of runtime package (module) has a suffix, for example, `pytransform_vax_00001`

- platform

  Note: New in v5.9.0

  A string includes one or many platforms. Multi platforms are separated by comma.

  Leave it to None or blank if not cross-platform obfuscating

- license_file

  Note: New in v5.9.0

  Use this license file other than the default one.

  Leave it to None or blank to use the default one.

- bootstrap_code

  Note: New in v5.9.0

  How to generate *Bootstrap Code* for the obfuscated entry scripts:

  – 0

  Do not insert bootstrap code into entry script

  – 1 (Default)

  Insert the bootstrap code into entry script. If the script name is `__init__.py`, make a relative import with leading dots, otherwise make absolute import.

  – 2

  The bootstrap code will always be made an absolute import without leading dots in the entry script.

  – 3

  The bootstrap code will always be made a relative import with leading dots in the entry script.

CHAPTER 6

Man Page

PyArmor is a command line tool used to obfuscate python scripts, bind obfuscated scripts to fixed machine or expire obfuscated scripts.

The syntax of the `pyarmor` command is:

```
pyarmor <command> [options]
```

The most commonly used pyarmor commands are:

```
obfuscate    Obfuscate python scripts
licenses     Generate new licenses for obfuscated scripts
pack         Pack obfuscated scripts to one bundle
hdinfo       Show hardware information
```

The commands for project:

```
init         Create a project to manage obfuscated scripts
config       Update project settings
build        Obfuscate all the scripts in the project

info         Show project information
check        Check consistency of project
```

The other commands:

```
benchmark    Run benchmark test in current machine
register     Make registration file work
download     Download platform-dependent dynamic libraries
runtime      Generate runtime package separately
```

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

Note: From v5.7.1, the first character is command alias for most usage commands:

```
obfuscate, licenses, pack, init, config, build
```

For example:

```
pyarmor o => pyarmor obfuscate
```

6.1 obfuscate

Obfuscate python scripts.

SYNOPSIS:

```
pyarmor obfuscate <options> SCRIPT...
```

OPTIONS

-O, --output PATH	Output path, default is *dist*
-r, --recursive	Search scripts in recursive mode
-s, --src PATH	Specify source path if entry script is not in the top most path
--exclude PATH	Exclude the path in recusrive mode. Multiple paths are allowed, separated by ",", or use this option multiple times
--exact	Only obfuscate list scripts
--no-bootstrap	Do not insert bootstrap code to entry script
--bootstrap <0,1,2,3>	How to insert bootstrap code to entry script
--no-cross-protection	Do not insert protection code to entry script
--plugin NAME	Insert extra code to entry script, it could be used multiple times
--platform NAME	Distribute obfuscated scripts to other platform
--advanced <0,1>	Disable or enable advanced mode
--restrict <0,1,2,3,4>	Set restrict mode
-n, --no-runtime	DO NOT generate runtime files
--package-runtime <0,1>	Save the runtime files as package or not
--enable-suffix	Generate the runtime package with unique name

DESCRIPTION

PyArmor first checks whether *Global Capsule* exists in the `HOME` path. If not, make it.

Then find all the scripts to be obfuscated. There are 3 modes to search the scripts:

- Normal: find all the *.py* files in the same path of entry script
- Recursive: find all the *.py* files in the path of entry script recursively
- Exact: only these scripts list in the command line

Note that only the *.py* files are touched by this command, all the other files aren't copied to output path. If there are many data files in the package, first copy the whole package to the output path, then obfuscate the *.py* files, thus all the *.py* files in the output path are overwritten by the obfuscated ones.

If there is an entry script, PyArmor will modify it, insert cross protection code into the entry script.

Next obfuscate all these scripts in the default output path *dist*.

After that make the *Runtime Package* in the *dist* path.

Finally insert the *Bootstrap Code* into entry script.

If `--exact` is set, all the scripts in the command line are taken as entry scripts. Otherwise only the first script is entry script.

Option `--src` used to specify source path if entry script is not in the top most path. For example:

```
# if no option --src, the "./mysite" is the source path
pyarmor obfuscate --src "." --recursive mysite/wsgi.py
```

Option `--plugin` is used to extend license type of obfuscated scripts, it will inject the content of plugin script into the obfuscated scripts. The corresponding filename of plugin is *NAME.py*. More information about plugin, refer to *How to Deal With Plugins*, and here is a real example to show usage of plugin *Using Plugin to Extend License Type*

Option `--platform` is used to specify the target platform of obfuscated scripts if target platform is different from build platform. Use this option multiple times if the obfuscated scripts are being to run many platforms. From v5.7.5, the platform names are standardized, command *download* could list all the available platform names.

Option `--restrict` is used to set restrict mode, *Restrict Mode*

RUNTIME FILES

By default the runtime files will be saved in the separated folder `pytransform` as package:

```
pytransform/
    __init__.py
    _pytransform.so, or _pytransform.dll in Windows, _pytransform.dylib in MacOS
    pytransform.key
    license.lic
```

But if `--package-runtime` is *0*, they will be saved in the same path with obfuscated scripts as four separated files:

```
pytransform.py
_pytransform.so, or _pytransform.dll in Windows, _pytransform.dylib in MacOS
pytransform.key
license.lic
```

If the option `--enable-suffix` is set, the runtime package or module name will be `pytransform_xxx`, here `xxx` is unique suffix based on the registration code of PyArmor.

BOOTSTRAP CODE

By default, the following *Bootstrap Code* will be inserted into the entry script:

```
from pytransform import pyarmor_runtime
pyarmor_runtime()
```

If the entry script is `__init__.py`, the *Bootstrap Code* will make a relative import by using leading dots like this:

```
from .pytransform import pyarmor_runtime
pyarmor_runtime()
```

But the option `--bootstrap` is set to `2`, the *Bootstrap Code* always makes absolute import without leading dots. If it is set to `3`, the *Bootstrap Code* always makes relative import with leading dots.

If the option `--enable-suffix` is set, the bootstrap code may like this:

```
from pytransform_vax_000001 import pyarmor_runtime
pyarmor_runtime(suffix='vax_000001')
```

If `--no-bootstrap` is set, or `--bootstrap` is *0*, then no bootstrap code will be inserted into the entry scripts.

EXAMPLES

- Obfuscate all the *.py* only in the current path:

```
pyarmor obfuscate foo.py
```

- Obfuscate all the *.py* in the current path recursively:

```
pyarmor obfuscate --recursive foo.py
```

- Obfuscate all the *.py* in the current path recursively, but entry script not in top most path:

```
pyarmor obfuscate --src "." --recursive mysite/wsgi.py
```

- Obfuscate a script *foo.py* only, no runtime files:

```
pyarmor obfuscate --no-runtime --exact foo.py
```

- Obfuscate all the *.py* in a path recursive, no entry script, no generate runtime package:

```
pyarmor obfuscate --recursive --no-runtime .
pyarmor obfuscate --recursive --no-runtime src/
```

- Obfuscate all the *.py* in the current path recursively, exclude all the *.py* in the path *build* and *tests*:

```
pyarmor obfuscate --recursive --exclude build,tests foo.py
pyarmor obfuscate --recursive --exclude build --exclude tests foo.py
```

- Obfuscate only two scripts *foo.py*, *moda.py* exactly:

```
pyarmor obfuscate --exact foo.py moda.py
```

- Obfuscate all the *.py* file in the path *mypkg/*:

```
pyarmor obfuscate --output dist/mypkg mypkg/__init__.py
```

- Obfuscate all the *.py* files in the current path, but do not insert cross protection code into obfuscated script `dist/foo.py`:

```
pyarmor obfuscate --no-cross-protection foo.py
```

- Obfuscate all the *.py* files in the current path, but do not insert bootstrap code at the beginning of obfuscated script `dist/foo.py`:

```
pyarmor obfuscate --no-bootstrap foo.py
```

- Insert the content of `check_ntp_time.py` into *foo.py*, then obfuscating *foo.py*:

```
pyarmor obfuscate --plugin check_ntp_time foo.py
```

- Only plugin *assert_armored* is called then inject it into the *foo.py*:

```
pyarmor obfuscate --plugin @assert_armored foo.py
```

- Obfuscate the scripts in Macos and run obfuscated scripts in Ubuntu:

```
pyarmor obfuscate --platform linux.x86_64 foo.py
```

- Obfuscate the scripts in advanced mode:

```
pyarmor obfuscate --advanced 1 foo.py
```

- Obfuscate the scripts with restrict mode 2:

```
pyarmor obfuscate --restrict 2 foo.py
```

- Obfuscate all the *.py* files in the current path except *__init__.py* with restrice mode 4:

```
pyarmor obfuscate --restrict 4 --exclude __init__.py --recursive .
```

- Obfuscate a package with unique runtime package name:

```
cd /path/to/mypkg
pyarmor obfuscate -r --enable-suffix --output dist/mypkg __init__.py
```

6.2 licenses

Generate new licenses for obfuscated scripts.

SYNOPSIS:

```
pyarmor licenses <options> CODE
```

OPTIONS

-O, --output OUTPUT Output path, *stdout* is supported

-e, --expired YYYY-MM-DD Expired date for this license

-d, --bind-disk SN Bind license to serial number of harddisk

-4, --bind-ipv4 IPV4 Bind license to ipv4 addr

-m, --bind-mac MACADDR Bind license to mac addr

-x, --bind-data DATA Pass extra data to license, used to extend license type

--disable-restrict-mode Disable all the restrict modes

--enable-period-mode Check license per hour when the obfuscated script is running

DESCRIPTION

In order to run obfuscated scripts, it's necessarey to hava a *license.lic*. As obfuscating the scripts, there is a default *license.lic* created at the same time. In this license the obfuscated scripts can run on any machine and never expired.

This command is used to generate new licenses for obfuscated scripts. For example:

```
pyarmor licenses --expired 2019-10-10 mycode
```

An expired license will be generated in the default output path plus code name *licenses/mycode*, then overwrite the old one in the same path of obfuscated script:

```
cp licenses/mycode/license.lic dist/pytransform/
```

Another example, bind obfuscated scripts in mac address and expired on 2019-10-10:

```
pyarmor licenses --expired 2019-10-10 --bind-mac 2a:33:50:46:8f tom
cp licenses/tom/license.lic dist/pytransform/
```

Before this, run command *hdinfo* to get hardware information:

```
pyarmor hdinfo
```

By option *-x* any data could be saved into the license file, it's mainly used to extend license tyoe. For example:

```
pyarmor licenses -x "2019-02-15" tom
```

In the obfuscated scripts, the data passed by *-x* could be got by this way:

```
from pytransfrom import get_license_info
info = get_license_info()
print(info['DATA'])
```

It also could output the license key in the stdout other than a file:

```
pyarmor --silent licenses --output stdout -x "2019-05-20" reg-0001
```

Note: Here is a real example *Using Plugin to Extend License Type*

6.3 pack

Obfuscate the scripts and pack them into one bundle.

SYNOPSIS:

```
pyarmor pack <options> SCRIPT | PROJECT
```

OPTIONS

-O, --output PATH Directory to put final built distributions in.

-e, --options OPTIONS Pass these extra options to *pyinstaller*

-x, --xoptions OPTIONS Pass these extra options to *pyarmor obfuscate*

-s FILE Use external .spec file to pack the scripts

--without-license Do not generate license for obfuscated scripts

--with-license FILE Use this license file other than default one

--clean Remove cached files before packing

--debug Do not remove build files after packing

--name Name to assign to the bundled (default: the script's basename)

DESCRIPTION

The command *pack* first calls PyInstaller to generate *.spec* file which name is same as entry script. The options specified by `--options` will be pass to PyInstaller to generate *.spec* file. It could any option accepted by PyInstaller except `-y`, `--noconfirm`, `-n`, `--name`, `--distpath`.

If there is in trouble, make sure this *.spec* works with PyInstaller. For example:

```
pyinstaller myscript.spec
```

Then *pack* will obfuscates all the *.py* files in the same path of entry script. It will call *pyarmor obfuscate* with options `-r`, `--output`, and the extra options specified by `--xoptions`. However if packing a project, that is to say, the last argument is project path other than one script, *pack* will obfuscate the scripts by command *build* with option *-B*, and all the `--xoptons` will be ignored. In this case config the project to control how to obfuscate the scripts.

Next *pack* patches the *.spec* file so that the original scripts could be replaced with the obfuscated ones.

Finally *pack* call PyInstaller with this pacthed *.spec* file to generate the final distributions.

For more information, refer to *How To Pack Obfuscated Scripts*.

If you'd like to change the final bundle name, specify the option `--name` directly, do not pass it by the option `-e`.

If you have a worked *.spec* file, just specify it by option `-s`, for example:

```
pyarmor pack -s foo.spec foo.py
```

The main script must be list in the command line, otherwise *pack* doesn't know where to find the scripts to be obfuscated. And in this case the option `-e` will be ignored.

If the option `--debug` is set, for example:

```
pyarmor pack --debug foo.py
```

The following generated files will be kept, generally all of them are removed after packing end:

```
foo-patched.spec
dist/obf/temp/hook-pytransform.py
dist/obf/*.py                       # All the obfuscated scripts
```

The patched *foo-patched.spec* could be used by pyinstaller to pack the obfuscated scripts directly, for example:

```
pyinstaller -y --clean foo-patched.spec
```

If some scripts are modified, just obfuscate them again, then run this command to pack them quickly. All the options for command *obfuscate* could be got from the output of command *pack*.

If there are many data files or hidden imports, it's better to write a hook file to find them easily. First create a hook file named `hook-sys.py`:

```
from PyInstaller.utils.hooks import collect_data_files, collect_all
datas, binaries, hiddenimports = collect_all('my_module_name')
datas += collect_data_files('submodule')
hiddenimports += ['_gdbm', 'socket', 'h5py.defs']
datas += [ ('/usr/share/icons/education_*.png', 'icons') ]
```

Then call pack with extra option `--additional-hooks-dir .` to tell pyinstaller find the hook in the current path:

```
pyarmor pack -e " --additional-hooks-dir ." foo.py
```

More information about pyinstaller hook, refer to https://pyinstaller.readthedocs.io/en/stable/hooks.html#understanding-pyinstaller-hooks

When something is wrong, turn on Python debug flag to show more information:

```
PYTHONDEBUG=y pyarmor pack ...

# In windows
set PYTHONDEBUG=y
pyarmor pack ...
```

EXAMPLES

- Obfuscate *foo.py* and pack them into the bundle *dist/foo*:

```
pyarmor pack foo.py
```

- Remove the build folder, and start a clean pack:

```
pyarmor pack --clean foo.py
```

- Pack the obfuscated scripts by an exists *myfoo.spec*:

```
pyarmor pack -s myfoo.spec foo.py
```

- Pass extra options to run *PyInstaller*:

```
pyarmor pack -e " -w --icon app.ico" foo.py
```

- Pass extra options to obfuscate scripts:

```
pyarmor pack -x " --exclude venv --exclude test" foo.py
```

- Pack the obfuscated script to one file and in advanced mode:

```
pyarmor pack -e " --onefile" -x " --advanced" foo.py
```

- Pack the obfuscated scripts and expired on 2020-12-25:

```
pyarmor licenses -e 2020-12-25 cy2020
pyarmor pack --with-license licenses/cy2020/license.lic foo.py
```

- Change the final bundle name to *my_app* other than *foo*:

```
pyarmor pack --name my_app foo.py
```

- Pack a project with advanced mode:

```
pyarmor init --entry main.py
pyarmor config --advanced 1
pyarmor pack .
```

Note: Since v5.9.0, possible pack one project directly by specify the project path in the command line. For example, create a project in the current path, then pack it:

```
pyarmor init --entry main.py
pyarmor pack .
```

By this way the obfuscated scripts could be fully controlled.

Important: The command *pack* will obfuscate the scripts automatically, do not try to pack the obfuscated the scripts.

6.4 hdinfo

Show hardware information of this machine, such as serial number of hard disk, mac address of network card etc. The information got here could be as input data to generate license file for obfuscated scripts.

SYNOPSIS:

```
pyarmor hdinfo
```

If *pyarmor* isn't installed, downlad this tool *hdinfo*

https://github.com/dashingsoft/pyarmor-core/tree/master/#hdinfo

And run it directly:

```
hdinfo
```

It will print the same hardware information as *pyarmor hdinfo*

6.5 init

Create a project to manage obfuscated scripts.

SYNOPSIS:

```
pyarmor init <options> PATH
```

OPTIONS

-t, --type <auto,app,pkg> Project type, default value is *auto*

-s, --src SRC Base path of python scripts, default is current path

-e, --entry ENTRY Entry script of this project

DESCRIPTION

This command will create a project in the specify *PATH*, and a file *.pyarmor_config* will be created at the same time, which is project configuration of JSON format.

If the option `--type` is set to *auto*, which is the default value, the project type will set to *pkg* if the entry script is *__init__.py*, otherwise to *app*.

The *init* command will set *is_package* to *1* if the new project is configured as *pkg*, otherwise it's set to *0*.

After project is created, use command *config* to change the project settings.

EXAMPLES

- Create a project in the current path:

```
pyarmor init --entry foo.py
```

- Create a project in the build path *obf*:

```
pyarmor init --entry foo.py obf
```

- Create a project for package:

```
pyarmor init --entry __init__.py
```

- Create a project in the path *obf*, manage the scripts in the path */path/to/src*:

```
pyarmor init --src /path/to/src --entry foo.py obf
```

6.6 config

Update project settings.

SYNOPSIS:

```
pyarmor config <options> [PATH]
```

OPTIONS

--name NAME Project name

--title TITLE Project title

--src SRC Project src, base path for matching scripts

--output PATH Output path for obfuscated scripts

--manifest TEMPLATE Manifest template string

--entry SCRIPT Entry script of this project

--is-package <0,1> Set project as package or not

--restrict <0,1,2,3,4> Set restrict mode

--obf-mod <0,1> Disable or enable to obfuscate module

--obf-code <0,1,2> Disable or enable to obfuscate function

--wrap-mode <0,1> Disable or enable wrap mode

--advanced <0,1> Disable or enable advanced mode

--cross-protection <0,1> Disable or enable to insert cross protection code into entry script

--runtime-path RPATH Set the path of runtime files in target machine

--plugin NAME Insert extra code to entry script, it could be used multiple times

--package-runtime <0,1> Save the runtime files as package or not

--bootstrap <0,1,2,3> How to insert bootstrap code to entry script

--enable-suffix <0,1> Generate the runtime package with unique name

--with-license FILENAME Use this license file other than the default one

DESCRIPTION

Run this command in project path to change project settings:

```
pyarmor config --option new-value
```

Or specify the project path at the end:

```
pyarmor config --option new-value /path/to/project
```

Option `--manifest` is comma-separated list of manifest template command, same as MANIFEST.in of Python Distutils.

Option `--entry` is comma-separated list of entry scripts, relative to src path of project.

If option `--plugin` is set to empty string, all the plugins will be removed.

For the details of each option, refer to *Project Configuration File*

EXAMPLES

- Change project name and title:

```
pyarmor config --name "project-1"  --title "My PyArmor Project"
```

- Change project entries:

```
pyarmor config --entry foo.py,hello.py
```

- Exclude path *build* and *dist*, do not search *.py* file from these paths:

```
pyarmor config --manifest "global-include *.py, prune build, prune dist"
```

- Obfuscate script with wrap mode off:

```
pyarmor config --wrap-mode 0
```

- Set plugin for entry script. The content of *check_ntp_time.py* will be insert into entry script as building project:

```
pyarmor config --plugin check_ntp_time.py
```

- Remove all plugins:

```
pyarmor config --plugin ''
```

6.7 build

Build project, obfuscate all scripts in the project.

SYNOPSIS:

```
pyarmor config <options> [PATH]
```

OPTIONS

-B, --force	Force to obfuscate all scripts
-r, --only-runtime	Generate extra runtime files only

-n, --no-runtime DO NOT generate runtime files

-O, --output OUTPUT Output path, override project configuration

--platform NAME Distribute obfuscated scripts to other platform

--package-runtime <0,1> Save the runtime files as package or not

DESCRIPTION

Run this command in project path:

```
pyarmor build
```

Or specify the project path at the end:

```
pyarmor build /path/to/project
```

The option `--no-runtime` may impact on the *Bootstrap Code*, the bootstrap code will make absolute import without leading dots in entry script.

About option `--platform` and `--package-runtime`, refer to command *obfuscate*

EXAMPLES

- Only obfuscate the scripts which have been changed since last build:

```
pyarmor build
```

- Force build all the scripts:

```
pyarmor build -B
```

- Generate runtime files only, do not try to obfuscate any script:

```
pyarmor build -r
```

- Obfuscate the scripts only, do not generate runtime files:

```
pyarmor build -n
```

- Save the obfuscated scripts to other path, it doesn't change the output path of project settings:

```
pyarmor build -B -O /path/to/other
```

- Build project in Macos and run obfuscated scripts in Ubuntu:

```
pyarmor build -B --platform linux.x86_64
```

6.8 info

Show project information.

SYNOPSIS:

```
pyarmor info [PATH]
```

DESCRIPTION

Run this command in project path:

```
pyarmor info
```

Or specify the project path at the end:

```
pyarmor info /path/to/project
```

6.9 check

Check consistency of project.

SYNOPSIS:

```
pyarmor check [PATH]
```

DESCRIPTION

Run this command in project path:

```
pyarmor check
```

Or specify the project path at the end:

```
pyarmor check /path/to/project
```

6.10 banchmark

Check the performance of obfuscated scripts.

SYNOPSIS:

```
pyarmor benchmark <options>
```

OPTIONS:

-m, --obf-mode <0,1> Whether to obfuscate the whole module

-c, --obf-code <0,1,2> Whether to obfuscate each function

-w, --wrap-mode <0,1> Whether to obfuscate each function with wrap mode

--debug Do not remove test path

DESCRIPTION

This command will generate a test script, obfuscate it and run it, then output the elapsed time to initialize, import obfuscated module, run obfuscated functions etc.

EXAMPLES

- Test performance with default mode:

```
pyarmor benchmark
```

- Test performance with no wrap mode:

```
pyarmor benchmark --wrap-mode 0
```

- Check the test scripts which saved in the path *.benchtest*:

```
pyarmor benchmark --debug
```

6.11 register

Make registration keyfile effect, or show registration information.

SYNOPSIS:

```
pyarmor register [KEYFILE]
```

DESCRIPTION

This command is used to register the purchased keyfile to take it effects:

```
pyarmor register /path/to/pyarmor-regfile-1.zip
```

Show registration information:

```
pyarmor register
```

6.12 download

List and download platform-dependent dynamic libraries.

SYNOPSIS:

```
pyarmor download <options> NAME
```

OPTIONS:

--help-platform	Display all available standard platform names
-L, --list FILTER	List available dynamic libraries in different platforms
-O, --output PATH	Save downloaded library to this path
--update	Update all the downloaded dynamic libraries

DESCRIPTION

This command mainly used to download available dynamic libraries for cross platform.

List all available standard platform names. For examples:

```
pyarmor download
pyarmor download --help-platform
pyarmor download --help-platform windows
pyarmor download --help-platform linux.x86_64
```

Then download one from the list. For example:

```
pyarmor download linux.armv7
pyarmor download linux.x86_64
```

By default the download file will be saved in the path `~/.pyarmor/platforms` with different platform names.

Option `--list` could filter the platform by name, arch, features, and display the information in details. For examples:

```
pyarmor download --list
pyarmor download --list windows
pyarmor download --list windows.x86_64
pyarmor download --list JIT
pyarmor download --list armv7
```

After *pyarmor* is upgraded, however these downloaded dynamic libraries won't be upgraded. The option `--update` could be used to update all these downloaded files. For example:

```
pyarmor download --update
```

6.13 runtime

Geneate *Runtime Package* separately.

SYNOPSIS:

```
pyarmor runtime <options>
```

OPTIONS:

-O, --output PATH Output path, default is *dist*

-n, --no-package Generate runtime files without package

-i, --inside Generate bootstrap script which is used inside one package

-L, --with-license FILE Replace default license with this file

--platform NAME Generate runtime package for specified platform

--enable-suffix Generate the runtime package with unique name

DESCRIPTION

This command is used to generate the runtime package separately.

The runtiem package could be shared if the scripts are obufscated by same *Global Capsule*. So generate it once, then need not generate the runtime files when obfuscating the scripts later.

It also generates a bootstrap script `pytransform_bootstrap.py` in the output path. This script is obfuscated from an empty script, and there is *Bootstrap Code* in it. It's mainly used to run *Bootstrap Code* in the plain script. For example, once it's imported, all the other obfuscated modules could be imported in one plain script:

```
import pytransform_bootstrap
import obf_foo
```

If option `--inside` is specified, it will generate bootstrap package `pytransform_bootstrap` other than one single script.

About option `--platform` and `--enable-suffix`, refer to command *obfuscate*

EXAMPLES

- Generate *Runtime Package* `pytransform` in the default path *dist*:

```
pyarmor runtime
```

- Not generate a package, but four separate files *Runtime Files*:

```
pyarmor runtime -n
```

- Generate bootstrap package `dist/pytransform_boostrap`:

```
pyarmor runtime -i
```

- Generate *Runtime Package* for platform *armv7* with expired license:

```
pyarmor licenses --expired 2020-01-01 code-001
pyarmor runtime --with-license licenses/code-001/license.lic --platform linux.
↪armv7
```

CHAPTER 7

Understanding Obfuscated Scripts

7.1 Global Capsule

The `.pyarmor_capsule.zip` in the `HOME` path called *Global Capsule*. *PyArmor* will read data from *Global Capsule* when obfuscating scripts or generating licenses for obfuscated scripts.

All the trial version of PyArmor shares one same `.pyarmor_capsule.zip`, which is created implicitly when executing command `pyarmor obfuscate`. It uses 1024 bits RSA keys, called *public capsule*.

For purchased version, each user will receive one exclusive *private capsule*, which use 2048 bits RSA key.

The capsule can't help restoring the obfuscated scripts at all. If your *private capsuel* got by someone else, the risk is that he/she may generate new license for your obfuscated scripts.

Generally this capsule is only in the build machine, it's not used by the obfuscated scripts, and should not be distributed to the end users.

7.2 Obfuscated Scripts

After the scripts are obfuscated by *PyArmor*, in the *dist* folder you find all the required files to run obfuscated scripts:

```
dist/
    myscript.py
    mymodule.py

    pytransform/
        __init__.py
        _pytransform.so, or _pytransform.dll in Windows, _pytransform.dylib in MacOS
        pytransform.key
        license.lic
```

The obfuscated scripts are normal Python scripts. The module *dist/mymodule.py* would be like this:

```
__pyarmor__(__name__, __file__, b'\x06\x0f...', 1)
```

The entry script *dist/myscript.py* would be like this:

```
from pytransform import pyarmor_runtime
pyarmor_runtime()
__pyarmor__(__name__, __file__, b'\x0a\x02...', 1)
```

7.2.1 Entry Script

In PyArmor, entry script is the first obfuscated script to be run or to be imported in a python interpreter process. For example, *__init__.py* is entry script if only one single python package is obfuscated.

7.3 Bootstrap Code

The first 2 lines in the entry script called *Bootstrap Code*. It's only in the entry script:

```
from pytransform import pyarmor_runtime
pyarmor_runtime()
```

For the obfuscated package which entry script is *__init__.py*. The bootstrap code may make a relateive import by leading ".":

```
from .pytransform import pyarmor_runtime
pyarmor_runtime()
```

And there is another form if the runtime path is specified as obfuscating scripts:

```
from pytransform import pyarmor_runtime
pyarmor_runtime('/path/to/runtime')
```

Since v5.8.7, the runtime package may has a suffix. For example:

```
from pytransform_vax_000001 import pyarmor_runtime
pyarmor_runtime(suffix='_vax_000001')
```

7.4 Runtime Package

The package *pytransform* which is in the same folder with obfuscated scripts called *Runtime Packge*. It's required to run the obfuscated script, and it's the only dependency of obfuscated scripts.

Generally this package is in the same folder with obfuscated scripts, but it can be moved anywhere. Only this package in any Python Path, the obfuscated scripts can be run as normal scripts. And all the scripts obfuscated by the same *Global Capsule* could share this package.

There are 4 files in this package:

```
pytransform/
    __init__.py                 A normal python module
    _pytransform.so/.dll/.lib   A dynamic library implements core functions
    pytransform.key             Data file
    license.lic                 The license file for obfuscated scripts
```

Before v5.7.0, the runtime package has another form *Runtime Files*

7.4.1 Runtime Files

They're not in one package, but as four separated files:

```
pytransform.py                A normal python module
_pytransform.so/.dll/.lib     A dynamic library implements core functions
pytransform.key               Data file
license.lic                   The license file for obfuscated scripts
```

Obviously *Runtime Package* is more clear than *Runtime Files*.

Since v5.8.7, the runtime package (module) may has a suffix, for example:

```
pytransform_vax_000001/
    __init__.py
    ...

pytransform_vax_000001.py
...
```

7.5 The License File for Obfuscated Script

There is a special runtime file *license.lic*, it's required to run the obfuscated scripts.

When executing `pyarmor obfuscate`, a default one will be generated, which allows obfuscated scripts run in any machine and never expired.

In order to bind obfuscated scripts to fix machine, or expire the obfuscated scripts, use command `pyarmor licenses` to generate a new *license.lic* and overwrite the default one.

Note: In PyArmor, there is another *license.lic*, which locates in the source path of PyArmor. It's required to run *pyarmor*, and issued by me, :)

7.6 Key Points to Use Obfuscated Scripts

- The obfuscated scripts are normal python scripts, so they can be seamless to replace original scripts.
- There is only one thing changed, the *bootstrap code* must be executed before running or importing any obfuscated scripts.
- The *runtime package* must be in any Python Path, so that the *bootstrap code* can run correctly.
- The *bootstrap code* will load dynamic library *_pytransform.so/.dll/.dylib* by *ctypes*. This file is dependent-platform, all the prebuilt dynamic libraries list here *Support Platfroms*
- By default the *bootstrap code* searchs dynamic library *_pytransform* in the *runtime package*. Check *pytransform._load_library* to find the details.
- If the dynamic library *_pytransform* isn't within the *runtime package*, change the *bootstrap code*:

```
from pytransform import pyarmor_runtime
pyarmor_runtime('/path/to/runtime')
```

Both of runtime files *license.lic* and *pytransform.key* should be in this path either.

- When starts a fresh python interpreter process by *multiprocssing.Process*, *os.exec*, *subprocess.Popen* etc., make sure the *bootstrap code* are called in new process before running any obfuscated script.

More information, refer to *How to Obfuscate Python Scripts* and *How to Run Obfuscated Script*

7.7 The Differences of Obfuscated Scripts

There are something changed after Python scripts are obfuscated:

- The major/minor version of Python in build machine should be same as in target machine. Because the scripts will be compiled to byte-code before they're obfuscated, so the obfuscated scripts can't be run by all the Python versions as the original scripts could. Especially for Python 3.6, it introduces word size instructions, and it's totally different from Python 3.5 and before. It's recommeded to run the obfuscated scripts with same major and minor version of Python.
- If Python interpreter is compiled with Py_TRACE_REFS or Py_DEBUG, it will crash to run obfuscated scripts.
- The callback function set by `sys.settrace`, `sys.setprofile`, `threading.settrace` and `threading.setprofile` will be ignored by obfuscated scripts.
- Some function in the module `inspect` may not work, and any other module or package may not work if it visits the source or byte code of the obfuscated scripts.
- It will crash to visit the attribute `co_const` of code object directly if the script is obfuscated in advanced mode.
- The attribute `__file__` of code object in the obfuscated scripts will be `<frozen name>` other than real filename. So in the traceback, the filename is shown as `<frozen name>`.

  Note that `__file__` of moudle is still filename. For example, obfuscate the script `foo.py` and run it:

```
def hello(msg):
    print(msg)

# The output will be 'foo.py'
print(__file__)

# The output will be '<frozen foo>'
print(hello.__file__)
```

7.8 About Third-Party Interpreter

About third-party interperter, for example Jython, and any embeded Python C/C++ code, they should satisfy the following conditions at least to run the obfuscated scripts:

- They must be load offical Python dynamic library, which should be built from the soure https://github.com/python/cpython , and the core source code should not be modified.
- On Linux, *RTLD_GLOBAL* must be set as loading *libpythonXY.so* by *dlopen*, otherwise obfuscated scripts couldn't work.

Note: Boost::python does not load *libpythonXY.so* with *RTLD_GLOBAL* by default, so it will raise error "No PyCode_Type found" as running obfuscated scripts. To solve this problem, try to call the method *sys.setdlopenflags(os.RTLD_GLOBAL)* as initializing.

- The module *ctypes* must be exists and *ctypes.pythonapi._handle* must be set as the real handle of Python dynamic library, PyArmor will query some Python C APIs by this handle.

CHAPTER 8

How PyArmor Does It

Look at what happened after `foo.py` is obfuscated by PyArmor. Here are the files list in the output path `dist`:

```
foo.py

pytransform/
    __init__.py
    _pytransform.so, or _pytransform.dll in Windows, _pytransform.dylib in MacOS
    pytransform.key
    license.lic
```

`dist/foo.py` is obfuscated script, the content is:

```
from pytransform import pyarmor_runtime
pyarmor_runtime()
__pyarmor__(__name__, __file__, b'\x06\x0f...')
```

There is an extra folder *pytransform* called *Runtime Package*, which are the only required to run or import obfuscated scripts. So long as this package is in any Python Path, the obfuscated script *dist/foo.py* can be used as normal Python script. That is to say:

The original python scripts can be replaced with obfuscated scripts seamlessly.

8.1 How to Obfuscate Python Scripts

How to obfuscate python scripts by PyArmor?

First compile python script to code object:

```
char *filename = "foo.py";
char *source = read_file( filename );
PyCodeObject *co = Py_CompileString( source, "<frozen foo>", Py_file_input );
```

Then change code object as the following way

- Wrap byte code `co_code` within a `try...finally` block:

```
wrap header:

        LOAD_GLOBALS    N (__armor_enter__)     N = length of co_consts
        CALL_FUNCTION   0
        POP_TOP
        SETUP_FINALLY   X (jump to wrap footer) X = size of original byte code

changed original byte code:

        Increase oparg of each absolute jump instruction by the size of wrap␣
↪header

        Obfuscate original byte code

        ...

wrap footer:

        LOAD_GLOBALS    N + 1 (__armor_exit__)
        CALL_FUNCTION   0
        POP_TOP
        END_FINALLY
```

- Append function names `__armor_enter, __armor_exit__` to `co_consts`
- Increase `co_stacksize` by 2
- Set CO_OBFUSCAED (0x80000000) flag in `co_flags`
- Change all code objects in the `co_consts` recursively

Next serializing reformed code object and obfuscate it to protect constants and literal strings:

```
char *string_code = marshal.dumps( co );
char *obfuscated_code = obfuscate_algorithm( string_code  );
```

Finally generate obfuscated script:

```
sprintf( buf, "__pyarmor__(__name__, __file__, b'%s')", obfuscated_code );
save_file( "dist/foo.py", buf );
```

The obfuscated script is a normal Python script, it looks like this:

```
__pyarmor__(__name__, __file__, b'\x01\x0a...')
```

8.2 How to Deal With Plugins

In PyArmor, the plugin is used to inject python code into the obfuscted script before the script is obfuscated, thus the plugin code could be executed when the obfuscated script is running. For example, use a plugin to check internet time:

```
pyarmor obfuscate --plugin check_ntp_time foo.py
```

Why not insert the plugin code into the script directly? Because most of them must be called in the obufscated scripts. For example, get the license information of the obfuscated scripts.

Each plugin is a normal Python script, PyArmor searches it by this way:

- If the plugin has absolute path, then find the corresponding *.py* file exactly.
- If it has relative path, first search the related *.py* file in the current path, then `$HOME/.pyarmor/plugins`, finally `{pyarmor_folder}/plugins`.
- Raise exception if not found

When there is plugin specified as obfuscating the script, each comment line will be scanned to find any plugin marker. There are 2 types of plugin marker:

- Plguin Definition Marker
- Plugin Call Marker

The plugin definition marker has this form:

```
# {PyArmor Plugins}
```

It must be one leading comment line, no indentation. Generally there is only one in a script, all the plugins will be injected here. If there is no plugin definition marker, none of plugins will be injected.

The plugin call maker has 3 forms, any comment line starts with these patterns is Call Marker:

```
# PyArmor Plugin:
# pyarmor_
# @pyarmor_
```

They could appear many times, any indentation, but have to behind plugin definition marker.

For the first form `# PyArmor Plugin:`, PyArmor just remove this pattern and one following whitespace exactly, and leave the rest part of this line as it is. For example:

```
# PyArmor Plugin: check_ntp_time() ==> check_ntp_time()
```

So long as there is any plugin specified in the command line, these replacements will be taken place. The rest part could be any valid Python code. For examples:

```
# PyArmor Plugin: print('This is plugin code') ==> print('This is plugin code')
# PyArmor Plugin: if sys.flags.debug:           ==> if sys.flags.debug:
# PyArmor Plugin:     check_something():        ==>     check_something()
```

For the second form `# pyarmor_`, it's only used to call function deinfed in the plugin. And if this function name is not specified as plugin name, PyArmor doesn't touch this marker. For example, obfuscating a script with plugin *check_multi_mac*, the first marker is replaced, the second not:

```
# pyarmor_check_multi_mac() ==> check_multi_mac()
# pyarmor_check_code() ==> # pyarmor_check_code()
```

The last form is almost same as the second, but `# @pyarmor_` will be replaced with `@`, it's mainly used to inject a decorator. For example:

```
# @pyarmor_assert_obfuscated(foo.connect) ==> @assert_obfuscated(foo.connect)
```

If the plugin doesn't include a leading `@` in command line, it will be always injected into the obfuscated scripts. For example:

```
pyarmor obfuscate --plugin check_multi_mac --plugin assert_armored foo.py
```

However, if there is a leading `@`, it couldn't be injected into the obfuscated scripts, until this plugin name appears in any plugin call marker or plugin decorator marker. For examples, if there is no any plugin call marker or decorator marker in the *foo.py*, both of plugins will be ignored:

```
pyarmor obfuscate --plugin @assert_armored foo.py
pyarmor obfuscate --plugin @/path/to/check_ntp_time foo.py
```

8.3 Special Handling of Entry Script

There are 2 extra changes for entry script:

- Before obfuscating, insert protection code to entry script.
- After obfuscated, insert bootstrap code to obfuscated script.

Before obfuscating entry scipt, PyArmor will search the content line by line. If there is line like this:

```
# {PyArmor Protection Code}
```

PyArmor will replace this line with protection code.

If there is line like this:

```
# {No PyArmor Protection Code}
```

PyArmor will not patch this script.

If both of lines aren't found, insert protection code before the line:

```
if __name__ == '__main__'
```

Do nothing if no *__main__* line found.

Here it's the default template of protection code:

```
def protect_pytransform():

    import pytransform

    def check_obfuscated_script():
        CO_SIZES = 49, 46, 38, 36
        CO_NAMES = set(['pytransform', 'pyarmor_runtime', '__pyarmor__',
                        '__name__', '__file__'])
        co = pytransform.sys._getframe(3).f_code
        if not ((set(co.co_names) <= CO_NAMES)
                and (len(co.co_code) in CO_SIZES)):
            raise RuntimeError('Unexpected obfuscated script')

    def check_mod_pytransform():
        def _check_co_key(co, v):
            return (len(co.co_names), len(co.co_consts), len(co.co_code)) == v
        for k, (v1, v2, v3) in {keylist}:
            co = getattr(pytransform, k).{code}
            if not _check_co_key(co, v1):
                raise RuntimeError('unexpected pytransform.py')
            if v2:
                if not _check_co_key(co.co_consts[1], v2):
                    raise RuntimeError('unexpected pytransform.py')
            if v3:
                if not _check_co_key(co.{closure}[0].cell_contents.{code}, v3):
                    raise RuntimeError('unexpected pytransform.py')
```

(continues on next page)

(continued from previous page)

```
    def check_lib_pytransform():
        filename = pytransform.os.path.join({rpath}, {filename})
        size = {size}
        n = size >> 2
        with open(filename, 'rb') as f:
            buf = f.read(size)
        fmt = 'I' * n
        checksum = sum(pytransform.struct.unpack(fmt, buf)) & 0xFFFFFFFF
        if not checksum == {checksum}:
            raise RuntimeError("Unexpected %s" % filename)
    try:
        check_obfuscated_script()
        check_mod_pytransform()
        check_lib_pytransform()
    except Exception as e:
        print("Protection Fault: %s" % e)
        pytransform.sys.exit(1)

protect_pytransform()
```

All the string template `{xxx}` will be replaced with real value by PyArmor.

To prevent PyArmor from inserting this protection code, pass `--no-cross-protection` as obfuscating the scripts:

```
pyarmor obfuscate --no-cross-protection foo.py
```

After the entry script is obfuscated, the *Bootstrap Code* will be inserted at the beginning of the obfuscated script.

8.4 How to Run Obfuscated Script

How to run obfuscated script `dist/foo.py` by Python Interpreter?

The first 2 lines, which called `Bootstrap Code`:

```
from pytransform import pyarmor_runtime
pyarmor_runtime()
```

It will fulfil the following tasks

- Load dynamic library `_pytransform` by `ctypes`
- Check `license.lic` is valid or not
- Add 3 cfunctions to module `builtins`: `__pyarmor__`, `__armor_enter__`, `__armor_exit__`

The next code line in `dist/foo.py` is:

```
__pyarmor__(__name__, __file__, b'\x01\x0a...')
```

`__pyarmor__` is called, it will import original module from obfuscated code:

```
static PyObject *
__pyarmor__(char *name, char *pathname, unsigned char *obfuscated_code)
{
```

(continues on next page)

(continued from previous page)

```
    char *string_code = restore_obfuscated_code( obfuscated_code );
    PyCodeObject *co = marshal.loads( string_code );
    return PyImport_ExecCodeModuleEx( name, co, pathname );
}
```

After that, in the runtime of this python interpreter

- `__armor_enter__` is called as soon as code object is executed, it will restore byte-code of this code object:

```
static PyObject *
__armor_enter__(PyObject *self, PyObject *args)
{
    // Got code object
    PyFrameObject *frame = PyEval_GetFrame();
    PyCodeObject *f_code = frame->f_code;

    // Increase refcalls of this code object
    // Borrow co_names->ob_refcnt as call counter
    // Generally it will not increased  by Python Interpreter
    PyObject *refcalls = f_code->co_names;
    refcalls->ob_refcnt ++;

    // Restore byte code if it's obfuscated
    if (IS_OBFUSCATED(f_code->co_flags)) {
        restore_byte_code(f_code->co_code);
        clear_obfuscated_flag(f_code);
    }

    Py_RETURN_NONE;
}
```

- `__armor_exit__` is called so long as code object completed execution, it will obfuscate byte-code again:

```
static PyObject *
__armor_exit__(PyObject *self, PyObject *args)
{
    // Got code object
    PyFrameObject *frame = PyEval_GetFrame();
    PyCodeObject *f_code = frame->f_code;

    // Decrease refcalls of this code object
    PyObject *refcalls = f_code->co_names;
    refcalls->ob_refcnt --;

    // Obfuscate byte code only if this code object isn't used by any function
    // In multi-threads or recursive call, one code object may be referenced
    // by many functions at the same time
    if (refcalls->ob_refcnt == 1) {
        obfuscate_byte_code(f_code->co_code);
        set_obfuscated_flag(f_code);
    }

    // Clear f_locals in this frame
    clear_frame_locals(frame);

    Py_RETURN_NONE;
}
```

8.5 How To Pack Obfuscated Scripts

The obfuscated scripts generated by PyArmor can replace Python scripts seamlessly, but there is an issue when packing them into one bundle by PyInstaller:

All the dependencies of obfuscated scripts CAN NOT be found at all

To solve this problem, the common solution is

1. Find all the dependencies by original scripts.
2. Add runtimes files required by obfuscated scripts to the bundle
3. Replace original scripts with obfuscated in the bundle
4. Replace entry script with obfuscated one

PyArmor provides command *pack* to achieve this. But in some cases maybe it doesn't work. This document describes what the command *pack* does, and also could be as a guide to bundle the obfuscated scripts by yourself.

First install `pyinstaller`:

```
pip install pyinstaller
```

Then obfuscate scripts to `dist/obf`:

```
pyarmor obfuscate --output dist/obf --runtime-mode 0 hello.py
```

Next generate specfile, add runtime files required by obfuscated scripts:

```
pyinstaller --add-data dist/obf/license.lic:. \
            --add-data dist/obf/pytransform.key:. \
            --add-data dist/obf/_pytransform.*:. \
            -p dist/obf --hidden-import pytransform \
            hello.py
```

> In windows, the `:` should be replace with `;` in the command line.

And patch specfile `hello.spec`, insert the following lines after the `Analysis` object. The purpose is to replace all the original scripts with obfuscated ones:

```
src = os.path.abspath('.')
obf_src = os.path.abspath('dist/obf')

for i in range(len(a.scripts)):
    if a.scripts[i][1].startswith(src):
        x = a.scripts[i][1].replace(src, obf_src)
        if os.path.exists(x):
            a.scripts[i] = a.scripts[i][0], x, a.scripts[i][2]

for i in range(len(a.pure)):
    if a.pure[i][1].startswith(src):
        x = a.pure[i][1].replace(src, obf_src)
        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]
```

Run patched specfile to build final distribution:

```
pyinstaller --clean -y hello.spec
```

Note: Option `--clean` is required, otherwise the obfuscated scripts will not be replaced because the cached *.pyz* will be used.

Check obfuscated scripts work:

```
# It works
dist/hello/hello.exe

rm dist/hello/license.lic

# It should not work
dist/hello/hello.exe
```

CHAPTER 9

Runtime Module *pytransform*

If you have realized that the obfuscated scripts are black box for end users, you can do more in your own Python scripts.In these cases, `pytransform` would be useful.

The `pytransform` module is distributed with obfuscated scripts, and must be imported before running any obfuscated scripts. It also can be used in your python scripts.

9.1 Contents

exception PytransformError
This is raised when any pytransform api failed. The argument to the exception is a string indicating the cause of the error.

get_expired_days()
Return how many days left for time limitation license.

>0: valid in these days

-1: never expired

Note: If the obfuscated script has been expired, it will raise exception and quit directly. All the code in the obfuscated script will not run, so this function will never return 0.

---

get_license_info()
Get license information of obfuscated scripts.

It returns a dict with keys:

- expired: Expired date
- IFMAC: mac address bind to this license
- HARDDISK: serial number of harddisk bind to this license
- IPV4: ipv4 address bind to this license

- DATA: any data stored in this licese, used by extending license type
- CODE: registration code of this license

The value *None* means no this key in the license.

Raise `Exception` if license is invalid, for example, it has been expired.

get_license_code()
Return a string, which is specified as generating the licenses for obfucated scripts.

Raise `Exception` if license is invalid.

get_hd_info(*hdtype*, *size=256*)
Get hardware information by *hdtype*, *hdtype* could one of

HT_HARDDISK return the serial number of first harddisk

HT_IFMAC return mac address of first network card

HT_IPV4 return ipv4 address of first network card

HT_DOMAIN return domain name of target machine

Raise `Exception` if something is wrong.

HT_HARDDISK, HT_IFMAC, HT_IPV4, HT_DOMAIN
Constant for *hdtype* when calling *get_hd_info()*

9.2 Examples

Copy those example code to any script, for example *foo.py*, obfuscate it, then run the obfuscated script.

Show left days of license

```
from pytransform import PytransformError, get_license_info, get_expired_days
try:
    code = get_license_info()['CODE']
    left_days = get_expired_days()
    if left_days == -1:
        print('This license for %s is never expired' % code)
    else:
        print('This license for %s will be expired in %d days' % (code, left_days))
except Exception as e:
    print(e)
```

More usage refer to *Using Plugin to Extend License Type*

Note: Though *pytransform.py* is not obfuscated when running the obfuscated script, it's also protected by *PyArmor*. If it's changed, the obfuscated script will raise protection exception.

Refer to *Special Handling of Entry Script*

CHAPTER 10

Support Platfroms

The core of PyArmor is written by C, the prebuilt dynamic libraries include the common platforms and some embeded platforms.

Some of them are distributed with PyArmor source package, in these platforms, *pyarmor* could run without downloading anything. Refer to *Prebuilt Libraries Distributed with PyArmor*.

For the other platforms, *pyarmor* first searches path `~/.pyarmor/platforms/SYSTEM/ARCH`, `SYSTEM.ARCH` is one of *Standard Platform Names*. If there is none, PyArmor will download it from remote server automatically. Refer to *The Others Prebuilt Libraries For PyArmor*.

For all the latest platforms, refer to https://github.com/dashingsoft/pyarmor-core/blob/master/platforms/index.json

There may be serveral dynamic libraries with different features in each platform. The platform name with feature number suffix combines an unique name. For example, `windows.x86_64.7` means anti-debug, JIT and andvanced mode supported, `windows.x86_64.0` means no any feature.

Note that the dynamic library with different features aren't compatible. For example, try to obfuscate the scripts with target platform `linux.x86_64.0` in the Windows, the obfuscated scripts don't work in the target machine. Because the full features dynamic library `windows.x86_64.7` is used in the Windows by default. Now the common platforms are full features, most of the others not yet.

In some platforms, *pyarmor* doesn't know it but there is available dynamic library in the table *The Others Prebuilt Libraries For PyArmor*. Just download it and save it in the path `~/.pyarmor/platforms/SYSTEM/ARCH`, this command `pyarmor -d download` will also display this path at the beginning. It's appreicated to send this platform information to jondy.zhao@gmail.com so that it could be recognized by *pyarmor* automatically. This script will display the required information by *pyarmor*:

```
from platform import *
print('system name: %s' % system())
print('machine: %s' % machine())
print('processor: %s' % processor())
print('aliased terse platform: %s' % platform(aliased=1, terse=1))

if system().lower().startswith('linux'):
    print('libc: %s' % libc_ver())
    print('distribution: %s' % linux_distribution())
```

Contact jondy.zhao@gmail.com if you'd like to run PyArmor in other platform.

10.1 Standard Platform Names

These names are used in the command *obfuscate*, *build*, *runtime*, *download* to specify platform.

- windows.x86
- windows.x86_64
- linux.x86
- linux.x86_64
- darwin.x86_64
- vs2015.x86
- vs2015.x86_64
- linux.arm
- linux.armv6
- linux.armv7
- linux.aarch32
- linux.aarch64
- android.aarch64
- android.armv7 (New in 5.9.3)
- uclibc.armv7 (New in 5.9.4)
- linux.ppc64
- darwin.arm64
- freebsd.x86_64
- alpine.x86_64
- alpine.arm
- poky.x86

10.2 Platform Tables

Table 1: Table-1. Prebuilt Libraries Distributed with PyArmor

Name	Platform	Arch	Features	Download	Description
windows.x86	Windows	i686	Anti-Debug, JIT, ADV	_pytransform.dll	Cross compile by i686-pc-mingw32-gcc in cygwin
windows.x86_64	Windows	AMD64	Anti-Debug, JIT, ADV	_pytransform.dll	Cross compile by x86_64-w64-mingw32-gcc in cygwin
linux.x86	Linux	i686	Anti-Debug, JIT, ADV	_pytransform.so	Built by GCC
linux.x86_64	Linux	x86_64	Anti-Debug, JIT, ADV	_pytransform.so	Built by GCC
darwin.x86_64	MacOSX	x86_64, intel	Anti-Debug, JIT, ADV	_pytransform.dylib	Built by CLang with MacOSX10.11

Table 2: Table-2. The Others Prebuilt Libraries For PyArmor

Name	Platform	Arch	Features	Download	Description
vs2015.x86	Windows	x86		_pytransform.dll	Built by VS2015
vs2015.x86_64	Windows	x64		_pytransform.dll	Built by VS2015
linxu.arm	Linux	armv5		_pytransform.so	32-bit Armv5 (arm926ej-s)
linxu.armv6	Linux	armv6		_pytransform.so	32-bit Armv6 (-marm -march=armv6 -mfloat-abi=hard)
linux.armv7	Linux	armv7	Anti-Debug, JIT	_pytransform.so	32-bit Armv7 Cortex-A, hard-float, little-endian
linux.aarch32	Linux	aarch32	Anti-Debug, JIT	_pytransform.so	32-bit Armv8 Cortex-A, hard-float, little-endian
linux.aarch64	Linux	aarch64	Anti-Debug, JIT	_pytransform.so	64-bit Armv8 Cortex-A, little-endian
linux.ppc64	Linux	ppc64le		_pytransform.so	For POWER8
darwin.arm64	iOS	arm64		_pytransform.dylib	Built by CLang with iPhoneOS9.3.sdk
freebsd.x86_64	FreeBSD	x86_64		_pytransform.so	Not support harddisk serial number
alpine.x86_64	Alpine Linux	x86_64		_pytransform.so	Built with musl-1.1.21 for Docker
alpine.arm	Alpine Linux	arm		_pytransform.so	Built with musl-1.1.21, 32-bit Armv5T, hard-float, little-endian
poky.x86	Inel Quark	i586		_pytransform.so	Cross compile by i586-poky-linux
android.aarch64	Android	aarch64		_pytransform.so	Build by android-ndk-r20/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android21-clang
android.armv7	Android	armv7l		_pytransform.so	Build by android-ndk-r20/toolchains/llvm/prebuilt/linux-x86_64/bin/armv7a-linux-android21-clang
uclibc.armv7	Linux	armv7l		_pytransform.so	Build by armv7-buildroot-uclibceabihf-gcc

CHAPTER 11

The Modes of Obfuscated Scripts

PyArmor could obfuscate the scripts in many modes in order to balance the security and performance. In most of cases, the default mode works fine. But if the performace is to be bottle-block or in some special cases, maybe you need understand what the differents of these modes and obfuscate the scripts in different mode so that they could work as desired.

11.1 Advanced Mode

This feature **Advanced Mode** is introduced from PyArmor 5.5.0.In this mode the structure of PyCode_Type is changed a little to improve the security. And a hook also is injected into Python interpreter so that the modified code objects could run normally. Besides if some core Python C APIs are changed unexpectedly, the obfuscated scripts in advanced mode won't work. Because this feature is highly depended on the machine instruction set, it's only available for x86/x64 arch now. And pyarmor maybe makes mistake if Python interpreter is compiled by old gcc or some other *C* compiles. It's welcome to report the issue if Python interpreter doesn't work in advanced mode.

Take this into account, the advanced mode is disabled by default. In order to enable it, pass option `--advanced` to command *obfuscate*:

```
pyarmor obfuscate --advanced 1 foo.py
```

In next minor version, this mode may be enabled by default.

Upgrade Notes:

Before upgrading, please estimate Python interpreter in product environments to be sure it works in advanced mode. Here is the guide

https://github.com/dashingsoft/pyarmor-core/tree/v5.3.0/tests/advanced_mode/README.md

It is recommended to upgrade in the next minor version.

Note: In trial version the module could not be obfuscated by advanced mdoe if there are more than about 30 functions in this module, (It still could be obfuscated by non-advanced mode).

11.2 Obfuscating Code Mode

In a python module file, generally there are many functions, each function has its code object.

- obf_code == 0

The code object of each function will keep it as it is.

- obf_code == 1 (Default)

In this case, the code object of each function will be obfuscated in different ways depending on wrap mode.

- obf_code == 2

Almost same as obf_mode 1, but obfuscating bytecode by more complex algorithm, and so slower than the former.

11.3 Wrap Mode

- wrap_mode == 0

When wrap mode is off, the code object of each function will be obfuscated as this form:

```
0   JUMP_ABSOLUTE            n = 3 + len(bytecode)

3    ...
     ... Here it's obfuscated bytecode of original function
     ...

n   LOAD_GLOBAL              ? (__armor__)
n+3 CALL_FUNCTION            0
n+6 POP_TOP
n+7 JUMP_ABSOLUTE            0
```

When this code object is called first time

1. First op is JUMP_ABSOLUTE, it will jump to offset n
2. At offset n, the instruction is to call PyCFunction *__armor__*. This function will restore those obfuscated bytecode between offset 3 and n, and move the original bytecode at offset 0
3. After function call, the last instruction is to jump to offset 0. The really bytecode now is executed.

After the first call, this function is same as the original one.

- wrap_mode == 1 (Default)

When wrap mode is on, the code object of each function will be wrapped with *try...finally* block:

```
LOAD_GLOBALS    N (__armor_enter__)     N = length of co_consts
CALL_FUNCTION   0
POP_TOP
SETUP_FINALLY   X (jump to wrap footer) X = size of original byte code

Here it's obfuscated bytecode of original function

LOAD_GLOBALS    N + 1 (__armor_exit__)
CALL_FUNCTION   0
POP_TOP
END_FINALLY
```

When this code object is called each time

1. *__armor_enter__* will restore the obfuscated bytecode
2. Execute the real function code
3. In the final block, *__armor_exit__* will obfuscate bytecode again.

11.4 Obfuscating module Mode

- obf_mod == 1 (Default)

The final obfuscated scripts would like this:

```
__pyarmor__(__name__, __file__, b'\x02\x0a...', 1)
```

The third parameter is serialized code object of the Python script. It's generated by this way:

```
PyObject *co = Py_CompileString( source, filename, Py_file_input );
obfuscate_each_function_in_module( co, obf_mode );
char *original_code = marshal.dumps( co );
char *obfuscated_code = obfuscate_whole_module( original_code  );
sprintf( buffer, "__pyarmor__(__name__, __file__, b'%s', 1)", obfuscated_code );
```

- obf_mod == 0

In this mode, the last statement would be like this to keep the serialized module as it is:

```
sprintf( buffer, "__pyarmor__(__name__, __file__, b'%s', 0)", original_code );
```

And the final obfuscated scripts would be:

```
__pyarmor__(__name__, __file__, b'\x02\x0a...', 0)
```

All of these modes only could be changed in the project for now, refer to *Obfuscating Scripts With Different Modes*

11.5 Restrict Mode

From PyArmor 5.7.0, the *Bootstrap Code* must be in the obfuscated scripts and must be specified as entry script. For example, there are 2 scripts *foo.py* and *test.py* in the same folder, obfuscated by this command:

```
pyarmor obfuscate foo.py
```

Inserting the *bootstrap code* into obfuscated script *dist/test.py* by manual doesn't work, because it's not specified as entry script. It must be run this command to insert the *Bootstrap Code*:

```
pyarmor obfuscate --no-runtime --exact test.py
```

If you need insert the *Bootstrap Code* into plain script, first obfuscate an empty script like this:

```
echo "" > pytransform_bootstrap.py
pyarmor obfuscate --no-runtime --exact pytransform_bootstrap.py
```

Then import *pytransform_bootstrap* in the plain script.

From PyArmor 5.5.6, there are 4 restrice modes:

- Mode 1

In this mode, obfuscated scripts must be one of the following formats:

```
__pyarmor__(__name__, __file__, b'...')

Or

from pytransform import pyarmor_runtime
pyarmor_runtime()
__pyarmor__(__name__, __file__, b'...')

Or

from pytransform import pyarmor_runtime
pyarmor_runtime('...')
__pyarmor__(__name__, __file__, b'...')
```

No any other statement can be inserted into obfuscated scripts.

For examples, the obfuscate scirpt *b.py* doesn't work, because there is an extra code "print":

```
$ cat b.py
from pytransform import pyarmor_runtime
pyarmor_runtime()
__pyarmor__(__name__, __file__, b'...')
print(__name__)

$ python b.py
```

- Mode 2

In this mode, except that the obfuscated can't changed, there are 2 restricts:

- The entry script must be obfuscated
- The obfuscated scripts could not be imported out of the obfuscated script

For example, this command will raise error if the *foo.py* is obfuscated by restrict mode 2:

```
$ python -c'import foo'
```

- Mode 3

In this mode, there is another restrict base on Mode 2:

- All the functions in the obfuscated script cound not be called out of the obfuscated scripts.

- Mode 4

It's similar with Mode 3, but there is a exception:

- The entry script could be plain script

It's mainly used for obfuscating Python package. The *__init__.py* is obfuscated by restrict mode 1, all the other scripts are obfuscated by restrict mode 4.

For example, it's the content of *mypkg/__init__.py*

```
# mypkg/
#     __init__.py is obfuscated by restrict mode 1
#     foo.py is obfuscated by restrict mode 4
```

(continues on next page)

(continued from previous page)

```
# The "foo.hello" could not be called by plain script directly
from .foo import hello

# The "open_hello" could be called by plain scirpt
def open_hello(msg):
    print('This is public hello: %s' % msg)

# The "proxy_hello" could be called by plain scirpt
def proxy_hello(msg):
    print('This is proxy hello: %s' % msg)
    # The "foo.hello" could be called by obfuscated "__init__.py"
    hello(msg)
```

Note: Mode 2 and 3 could not be used to obfuscate the Python package, because it will be imported from other plain scripts.

Note: Restrict mode is applied to one single script, different scripts could be obfuscated by different restrict mode.

From PyArmor 5.2, Restrict Mode 1 is default.

Obfuscating the scripts by other restrict mode:

```
pyarmor obfuscate --restrict=2 foo.py
pyarmor obfuscate --restrict=4 foo.py

# For project
pyarmor config --restrict=2
pyarmor build -B
```

All the above restricts could be disabled by this way if required:

```
pyarmor obfuscate --restrict=0 foo.py

# For project
pyarmor config --restrict=0
pyarmor build -B
```

For more examples, refer to *Improving The Security By Restrict Mode*

CHAPTER 12

The Performance of Obfuscated Scripts

Run command *banchmark* to check the performance of obfuscated scripts:

```
pyarmor benchmark
```

Here it's sample output:

```
INFO     Start benchmark test ...
INFO     Obfuscate module mode: 1
INFO     Obfuscate code mode: 1
INFO     Obfuscate wrap mode: 1
INFO     Benchmark bootstrap ...
INFO     Benchmark bootstrap OK.
INFO     Run benchmark test ...
Test script: bfoo.py
Obfuscated script: obfoo.py
---------------------------------------

load_pytransform: 28.429590911694085 ms
init_pytransform: 10.701080723946758 ms
verify_license: 0.515428636879825 ms
total_extra_init_time: 40.34842417122847 ms

import_no_obfuscated_module: 9.601499631936461 ms
import_obfuscated_module: 6.858413569322354 ms

re_import_no_obfuscated_module: 0.007263492985840059 ms
re_import_obfuscated_module: 0.0058666674116400475 ms

run_empty_no_obfuscated_code_object: 0.015085716201360122 ms
run_empty_obfuscated_code_object: 0.0058666674116400475 ms

run_one_thousand_no_obfuscated_bytecode: 0.003911111607760032 ms
run_one_thousand_obfuscated_bytecode: 0.005307937181960043 ms
```

(continues on next page)

(continued from previous page)

```
run_ten_thousand_no_obfuscated_bytecode: 0.003911111607760032 ms
run_ten_thousand_obfuscated_bytecode: 0.005587302296800045 ms

--------------------------------------
INFO     Remove test path: .\.benchtest
INFO     Finish benchmark test.
```

The total extra init time is about *40ms*. It includes the time of loading dynamic library, initialzing it and verifing license.

Note that the time of importing obfuscated module is less than of importing no obfuscated module, because the obfuscated scripts has been compiled as byte-code, the original scripts need extra time to compile.

List all available options:

```
pyarmor benchmark -h
```

Specify other options to check the performance in different mode. For example:

```
pyarmor benchmark --wrap-mode 0
```

Look at the scripts used to run benchmark test:

```
pyarmor benchmark --debug
```

All the used files are saved in the folder *.benchtest*

CHAPTER 13

The Security of PyArmor

PyArmor will obfuscate python module in two levels. First obfucate each function in module, then obfuscate the whole module file. For example, there is a file *foo.py*:

```
def hello():
    print('Hello world!')

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

if __name == '__main__':
    hello()
    print('1 + 1 = %d' % sum(1, 1))
```

PyArmor first obfuscates the function *hello* and *sum*, then obfuscates the whole moudle *foo*. In the runtime, only current called function is restored and it will be obfuscated as soon as code object completed execution. So even trace code in any `c` debugger, only a piece of code object could be got one time.

13.1 Cross Protection for *_pytransform*

The core functions of *PyArmor* are written by *c* in the dynamic library *_pytransform*. *_pytransform* protects itself by JIT technical, and the obfuscated scripts is protected by *_pytransform*. On the other hand, the dynamic library *_pytransform* is checked in the obfuscated script to be sure it's not changed. This is called Cross Protection.

The dynamic library *_pytransform.so* uses JIT technical to achieve two tasks:

- Keep the des key used to encrypt python scripts from tracing by any c debugger
- The code segment can't be changed any more. For example, change instruction *JZ* to *JNZ*, so that *_pytransform.so* can execute even if checking license failed

How JIT works?

First PyArmor defines an instruction set based on GNU lightning.

Then write some core functions by this instruction set in c file, maybe like this:

```
t_instruction protect_set_key_iv = {
    // function 1
    0x80001,
    0x50020,
    ...

    // function 2
    0x80001,
    0xA0F80,
    ...
}

t_instruction protect_decrypt_buffer = {
    // function 1
    0x80021,
    0x52029,
    ...

    // function 2
    0x80001,
    0xC0901,
    ...
}
```

Build *_pytransform.so*, calculate the codesum of code segment of *_pytransform.so*

Replace the related instructions with real codesum got before, and obfuscate all the instructions except "function 1" in c file. The updated file maybe likes this:

```
t_instruction protect_set_key_iv = {
    // plain function 1
    0x80001,
    0x50020,
    ...

    // obfuscated function 2
    0xXXXXX,
    0xXXXXX,
    ...
}

t_instruction protect_decrypt_buffer = {
    // plain function 1
    0x80021,
    0x52029,
    ...

    // obfuscated function 2
    0xXXXXX,
    0xXXXXX,
    ...
}
```

Finally build *_pytransform.so* with this changed c file.

When running obfuscated script, *_pytransform.so* loaded. Once a proected function is called, it will

1. Generate code from *function 1*

2. **Run *function 1*:**
    - check codesum of code segment, if not expected, quit
    - check tickcount, if too long, quit
    - check there is any debugger, if found, quit
    - clear hardware breakpoints if possible
    - restore next function *function 2*
3. Generate code from *function 2*
4. Run *function 2*, do same thing as *function 1*

After repeat some times, the real code is called. All of that is to be sure there is no breakpoint in protection code.

In order to protect *_pytransform* in Python script, some extra code will be inserted into the entry script, refer to *Special Handling of Entry Script*

CHAPTER 14

When Things Go Wrong

When there is in trouble, try to solve it by these ways.

As running `pyarmor`:

- Check the console output, is there any wrong path, or any odd information
- Run *pyarmor* with debug option `-d` to get more information. For example:

```
pyarmor -d obfuscate --recurisve foo.py
```

- Set Python debug flag to get more information. For example:

```
PYTHONDEBUG=y pyarmor obfuscate --recurisve foo.py

# In Windows
set PYTHONDEBUG=y
pyarmor obfuscate --recurisve foo.py
```

As running the obfuscated scripts:

- Turn on Python debug option by `-d` to print more information. For example:

```
python -d obf_foo.py
PYTHONDEBUG=y python obf_foo.py

# In Windows
set PYTHONDEBUG=y
python obf_foo.py
```

After python debug option is on, there will be a log file *pytransform.log* generated in the current path, which includes more debug information.

14.1 Segment fault

In the following cases, obfuscated scripts will crash

- Running obfuscated script by the debug version Python
- Obfuscating scripts by Python 2.6 but running the obfuscated scripts by Python 2.7

After PyArmor 5.5.0, some machines may be crashed because of advanced mode. A quick workaround is to disable advanced mode by editing the file `pytransform.py` which locates in the installed path of `pyarmor` , in the function `_load_library`, uncomment about line 202. The final code looks like this:

```
# Disable advanced mode if required
m.set_option(5, c_char_p(1))
```

14.2 Bootstrap Problem

14.2.1 Could not find *_pytransform*

Generally, the dynamic library *_pytransform* is in the *Runtime Package*, before v5.7.0, it's in the same path of obfuscated scripts. It may be:

- *_pytransform.so* in Linux
- *_pytransform.dll* in Windows
- *_pytransform.dylib* in MacOS

First check whether the file exists. If it exists:

- Check the permissions of dynamic library

  If there is no execute permissions in Windows, it will complain: *[Error 5] Access is denied*

- Check whether *ctypes* could load *_pytransform*:

```
from pytransform import _load_library
m = _load_library(path='/path/to/dist')
```

- Try to set the runtime path in the *Bootstrap Code* of entry script:

```
from pytransform import pyarmor_runtime
pyarmor_runtime('/path/to/dist')
```

Still doesn't work, report an issue

14.2.2 ERROR: Unsupport platform linux.xxx

In some machines *pyarmor* could not recognize the platform and raise error. If there is available dynamic library in the table *Table-2. The Others Prebuilt Libraries For PyArmor*. Just download it and save it in the path `~/.pyarmor/platforms/SYSTEM/ARCH`, this command `pyarmor -d download` will also display this path at the beginning.

If there is no any available one, contact jondy.zhao@gmail.com if you'd like to run *pyarmor* in this platform.

14.2.3 /lib64/libc.so.6: version 'GLIBC_2.14' not found

In some machines there is no *GLIBC_2.14*, it will raise this exception.

One solution is patching *_pytransform.so* by the following way.

First check version information:

```
readelf -V /path/to/_pytransform.so
...

Version needs section '.gnu.version_r' contains 2 entries:
 Addr: 0x00000000000056e8  Offset: 0x0056e8  Link: 4 (.dynstr)
  000000: Version: 1  File: libdl.so.2  Cnt: 1
  0x0010:   Name: GLIBC_2.2.5  Flags: none  Version: 7
  0x0020: Version: 1  File: libc.so.6  Cnt: 6
  0x0030:   Name: GLIBC_2.7  Flags: none  Version: 8
  0x0040:   Name: GLIBC_2.14  Flags: none Version: 6
  0x0050:   Name: GLIBC_2.4  Flags: none  Version: 5
  0x0060:   Name: GLIBC_2.3.4  Flags: none  Version: 4
  0x0070:   Name: GLIBC_2.2.5  Flags: none  Version: 3
  0x0080:   Name: GLIBC_2.3  Flags: none  Version: 2
```

Then replace the entry of *GLIBC_2.14* with *GLIBC_2.2.5*:

- Copy 4 bytes at 0x56e8+0x10=0x56f8 to 0x56e8+0x40=0x5728
- Copy 4 bytes at 0x56e8+0x18=0x5700 to 0x56e8+0x48=0x5730

Here are sample commands:

```
xxd -s 0x56f8 -l 4 _pytransform.so | sed "s/56f8/5728/" | xxd -r - _pytransform.so
xxd -s 0x5700 -l 4 _pytransform.so | sed "s/5700/5730/" | xxd -r - _pytransform.so
```

Note: From v5.7.9, this patch is not required. In cross-platform all you need to do is specify the platform to *centos6.x86_64* to fix this issue. For example:

```
pyarmor obfuscate --platform centos6.x86_64 foo.py
```

14.3 Obfuscating Scripts Problem

14.3.1 Warning: code object xxxx isn't wrapped

It means this function isn't been obfuscated, because it includes some special instructions.

For example, there is 2-bytes instruction *JMP 255*, after the code object is obfuscated, the operand is increased to *267*, and the instructions will be changed to:

```
EXTEND 1
JMP 11
```

In this case, it's complex to obfuscate the code object with wrap mode. So the code object is left as it's, but all the other code objects still are obfuscated.

In later version, it will be obfuscated with non wrap mode.

In current version add some unused code in this function so that the operand isn't the critical value may avoid this warning.

Note: This has been fixed in v5.5.0.

14.3.2 Error: Try to run unauthorized function

If there is any file *license.lic* or *pytransform.key* in the current path, pyarmor maybe reports this error. One solution is to remove all of that files, the other solution to upgrade PyArmor to v5.4.5 later.

14.3.3 'XXX' codec can't decode byte 0xXX

Add the exact source encode at the begin of the script. For example:

```
# -*- coding: utf-8 -*-
```

Refer to https://docs.python.org/2.7/tutorial/interpreter.html#source-code-encoding

14.3.4 Why plugin doesn't work

If the plugin script doesn't work as expected, first check the plugin script could be injected into the entry script by set Python debug flag:

```
# In linux
export PYTHONDEBUG=y
# In Windows
set PYTHONDEBUG=y

pyarmor obfuscate --exact --plugin check_ntp_time foo.py
```

It will generate patched file `foo.py.pyarmor-patched`, make sure the content of plugin script has been inserted into the right place, and the verify function will be executed.

14.4 Running Obfuscated Scripts Problem

14.4.1 The *license.lic* generated doesn't work

The key is that the capsule used to obfuscate scripts must be same as the capsule used to generate licenses.

The *Global Capsule* will be changed if the trial license file of *PyArmor* is replaced with normal one, or it's deleted occasionally (which will be generated implicitly as running command *pyarmor obfuscate* next time).

In any cases, generating new license file with the different capsule will not work for the obfuscated scripts before. If the old capsule is gone, one solution is to obfuscate these scripts by the new capsule again.

14.4.2 NameError: name '__pyarmor__' is not defined

No *Bootstrap Code* are executed before importing obfuscated scripts.

When creating new process by *Popen* or *Process* in mod *subprocess* or *multiprocessing*, to be sure that *Bootstrap Code* will be called before importing any obfuscated code in sub-process. Otherwise it will raise this exception.

14.4.3 Marshal loads failed when running xxx.py

1. Check whether the version of Python to run obfuscated scripts is same as the version of Python to obfuscate script
2. Run obfuscated script by *python -d* to show more error message.
3. Be sure the capsule used to generated the license file is same as the capsule used to obfuscate the scripts. The filename of the capsule will be shown in the console when the command is running.

14.4.4 _pytransform can not be loaded twice

When the function *pyarmor_runtime* is called twice, it will complaint *_pytransform can not be loaded twice*

For example, if an obfuscated module includes the following lines:

```
from pytransform import pyarmor_runtime
pyarmor_runtime()
__pyarmor__(....)
```

When importing this module from entry script, it will report this error. The first 2 lines should be in the entry script only, not in the other module.

This limitation is introduced from v5.1, to disable this check, just edit *pytransform.py* and comment these lines in function *pyarmor_runtime*:

```
if _pytransform is not None:
    raise PytransformError('_pytransform can not be loaded twice')
```

Note: This limitation has been removed from v5.3.5.

14.4.5 Check restrict mode failed

Use obfuscated scripts in wrong way, by default all the obfuscated scripts can't be changed any more.

Besides packing the obfuscated scripts will report this error either. Do not pack the obfuscated scripts, but pack the plain scripts directly.

For more information, refer to *Restrict Mode*

14.4.6 Protection Fault: unexpected xxx

Use obfuscated scripts in wrong way, by default, all the runtime files can't be changed any more. Do not touch the following files

- pytransform.py
- _pytransform.so/.dll/.dylib

For more information, refer to *Special Handling of Entry Script*

14.4.7 Run obfuscated scripts reports: Invalid input packet

If the scripts are obfuscated in different platform, check the notes in *Distributing Obfuscated Scripts To Other Platform*

Before v5.7.0, check if there is any of *license.lic* or *pytransform.key* in the current path. Make sure they're generated for the obfuscated scripts. If not, rename them or move them to other path.

Because the obfuscated scripts will first search the current path, then search the path of runtime module *pytransform.py* to find the file *license.lic* and *pytransform.key*. If they're not generated for the obfuscated script, this error will be reported.

14.4.8 OpenCV fails because of *NEON - NOT AVAILABLE*

In some Raspberry Pi platform, run the obfuscated scripts to import OpenCV fails:

```
********************************************** ***************
* FATAL ERROR: *
* This OpenCV build doesn't support current CPU / HW configuration *
* *
* Use OPENCV_DUMP_CONFIG = 1 environment variable for details *
********************************************** ***************

Required baseline features:
NEON - NOT AVAILABLE
terminate called after throwing an instance of 'cv :: Exception'
  what (): OpenCV (3.4.6) /home/pi/opencv-python/opencv/modules/core/src/system.
↪cpp:538: error:
(-215: Assertion failed) Missing support for required CPU baseline features. Check␣
↪OpenCV build
configuration and required CPU / HW setup. in function 'initialize'
```

One solution is to specify optioin `--platform` to *linux.armv7.0*:

```
pyarmor obfuscate --platform linux.armv7.0 foo.py
pyarmor build --platform linux.armv7.0
pyarmor runtime --platform linux.armv7.0
```

The other solution is to set environment variable *PYARMOR_PLATFORM* to *linux.armv7.0*. For examples:

```
PYARMOR_PLATFORM=linux.armv7.0 pyarmor obfuscate foo.py
PYARMOR_PLATFORM=linux.armv7.0 pyarmor build

Or,

export PYARMOR_PLATFORM=linux.armv7.0
pyarmor obfuscate foo.py
pyarmor build
```

14.5 Packing Obfuscated Scripts Problem

14.5.1 No module name pytransform

If report this error as running command *pyarmor pack*:

- Make sure the script specified in the command line is not obfuscated

- Run *pack* with extra option `--clean` to remove cached *myscript.spec*:

```
pyarmor pack --clean foo.py
```

14.6 PyArmor Registration Problem

14.6.1 Purchased pyarmor is not private

Even obfuscated with purchased version, license from trial version works:

- Make sure command *pyarmor register* shows correct registration information
- Make sure *Global Capsule* file *~/.pyarmor_capsule.zip* is same as the one in the keyfile *pyarmor-regfile-1.zip*
- Try to reboot system.

14.7 Known Issues

14.7.1 Obfuscate scripts in cross platform

From v5.6.0 to v5.7.0, there is a bug for cross platform. The scripts obfuscated in linux64/windows64/darwin64 don't work after copied to one of this target platform:

```
armv5, android.aarch64, ppc64le, ios.arm64, freebsd, alpine, alpine.arm, poky-i586
```

14.8 Misc. Questions

14.8.1 How easy is to recover obfuscated code

If someone tries to break the obfuscation, he first must be an expert in the field of reverse engineer, and be an expert of Python, who should understand the structure of code object of python, how python interpreter each instruction. If someone of them start to reverse, he/she must step by step thousands of machine instruction, and research the algorithm by machine codes. So it's not an easy thing to reverse pyarmor.

CHAPTER 15

License

The software is distributed as Free To Use But Restricted. Free trial version never expires, the limitations are

- The maximum size of code object is 32756 bytes in trial version
- The scripts obfuscated by trial version are not private. It means anyone could generate the license file which works for these obfuscated scripts.
- In trial version if obfuscating the Python scripts in advanced mode, the limitation is no more than about 32 functions (code objects) in one module.
- Without permission the trial version may not be used for the Python scripts of any commercial product.

About the license file of obfuscated scripts, refer to *The License File for Obfuscated Script*

A registration code is required to obfuscate big code object or generate private obfuscated scripts.

There are 2 basic types of licenses issued for the software. These are:

- A personal license for home users. The user purchases one license to use the software on his own computer.

  Home users may use their personal license to obfuscate all the python scripts which are property of the license owner, to generate private license files for the obfuscated scripts and distribute them and all the required files to any other machine or device.

  Home users could NOT obfuscate any python script which is NOT property of the license owner.

- A enterprise license for business users. The user purchases one license to use the software for one product serials of an organization.

  One product serials include the current version and any other latter versions of the same product.

  Business users may use their enterprise license on all computers and embedded devices to obfuscate all the python scripts of this product serials, to generate private license files for these obfuscated scripts and distribute them and all the required files to any other machine and device.

  Without permission of the software owner the license purchased for one product serials should not be used for other product serials. Business users should purchase new license for different product serials.

15.1 Purchase

To buy a license, please visit the following url

https://order.shareit.com/cart/add?vendorid=200089125&PRODUCT{[}300871197{]}=1

A registration keyfile generally named "pyarmor-regfile-1.zip" will be sent to your email immediately after payment is completed successfully. There are 3 files in the archive:

- REAME.txt
- license.lic (registration code)
- .pyarmor_capsule.zip (private capsule)

Run the following command to take this keyfile effects:

```
pyarmor register /path/to/pyarmor-regfile-1.zip
```

Check the registeration information:

```
pyarmor register
```

If the version of PyArmor < 5.6, unzip this registration file, then

- Copy "license.lic" in the archive to the installed path of PyArmor
- Copy ".pyarmor_capsule.zip" in the archive to user HOME path

After the registration keyfile takes effect, you need obfuscate the scripts again.

The registration code is valid forever, it can be used permanently.

CHAPTER 16

Change Logs

16.1 5.9.5

- Change the plugin search policy, do not support enviorment variable *PYARMOR_PLUGIN*, but search folder *plugins* in the pyarmor package path.
- Add a new path *plugins* in the package source, there are several common plugins. So it's easy to check internet time time by this way:

```
pyarmor obfuscate --plugin check_ntp_time foo.py
```

  Before that both of these lines should be inserted into `foo.py`:

```
# {PyArmor Plugins}
# PyArmor Plugin: check_ntp_time()
```

- Fix pack bug: *pyi-makespec: error: unrecognized arguments: -y* if extra options are passed
- Document command *pack* in details: https://pyarmor.readthedocs.io/en/latest/man.html#pack

16.2 5.9.4

- Fix pack issue: *pyi-makespec* doesn't work
- Add new platform: *uclibc-armv7*
- Fix issue: guess encoding failed if there are non-ascii characters in the second line
- Document how to work with Nuitka, https://pyarmor.readthedocs.io/en/latest/advanced.html#work-with-nuitka

16.3 5.9.3

- Add new option `--enable-period-mode` in the command *licenses*
- When running the obfuscated scripts it will check license periodly (per hour) if the option `--enable-period-mode` is set in the license file

16.4 5.9.2

- Fix bug: the command *pyarmor runtime –platform alpine.x86_64* raises error (#201)
- Fix bug: the platform *linux.armv6* doesn't work in Raspberry PI Zero W, rebuild the dynamic library with *-march=armv6 -mfloat-abi=hard -marm*

16.5 5.9.1

- Python debugger and profile tool could work with the plain python scripts even if the obfuscated packages are imported. Note that the obfuscated scripts still couldn't be traced.
- Refine *pack* command, use *pyi-makespec* to generate *.spec* file
- Fix advanced mode fails in some linux platforms
- Support platform *linux.armv6*
- Fix python38 issue: in the wrap mode the footer block isn't executed

16.6 5.9.0

pyarmor-webui is published as a separated package, it has been removed from source package of pyarmor. Now it's a full feature webui, and could be installed by *pip install pyarmor-webui*.

- Support environment variable *PYARMOR_HOME* as one extra path to find the *license.lic* of pyarmor. Now the search order is:
  - In the package path of pyarmor
  - *$PYARMOR_HOME/.pyarmor/license.lic*
  - *$HOME/.pyarmor/license.lic*
  - *$USERPROFILE/.pyarmor/license.lic* (Only for Windows)
- In command *licenses* if option *output* is set, do not append extra path *licenses* in the final output path
- In command *obfuscate* with option *–exact*, all the scripts list in the command line will be taken as entry script.
- The last argument in command *pack* could be a project path or .json file
- Add new option `--name` in the command *pack*
- Add new project attribute *license_file*, *bootstrap_code*
- Add new option `--with-license`, `--bootstrap` in the command *config*
- Add new option `--bootstrap` in the command *obfuscate*

- The options `--package-runtime` doesn't support *2* and *3*, use `--bootstrap=2` or `--bootstrap=3` instead
- For command *licenses* the generated license could be printed to stdout by setting the option `--output` to *stdout*

16.7 5.8.9

- Fix cross platform issue for vs2015.x86 and vs2015.x86_64
- In command *config* add option `--advanced` as alias of `--advanced-mode`

16.8 5.8.8

- Fix issue: the obfuscated scripts will crash when importing the packages obfuscated with advanced mode by other registered pyarmor

16.9 5.8.7

In this version, the scripts could be obfuscated with option *–enable-suffix*, then the name of the runtime package and builtin functions will be unique. By this way the scripts obfuscated by different capsule could run in the same Python interpreter.

For example, the bootstrap code may like this with suffix *_vax_000001*:

```
from pytransform_vax_000001 import pyarmor_runtime
pyarmor_runtime(suffix="_vax_000001")
```

Refer to https://pyarmor.readthedocs.io/en/latest/advanced.html#obfuscating-package-no-conflict-with-others

- Add option *–enable-suffix* in the commands *obfuscate*, *config* and *runtime*
- Add option *–with-license* in the command *pack*
- Fix issue: the executable file made by *pack* raises protection fault exception on MacOSX

16.10 5.8.6

- Raise exception other than *sys.exit(1)* when pyarmor_runtime fails
- Refine cross protection code to improve the security
- Fix issue: advanced mode fails in some MacOSX machines with python2.7

16.11 5.8.5

- Add platform data file *index.json* to source package
- Refine core library for platform MacOSX

16.12 5.8.4

- Fix issue: advanced mode doesn't work in some MacOSX machines.
- Fix issue: can't get the serial number of SSD harddisk in MacOSX platform

16.13 5.8.3

- Fix issue: the *_pytransform.dll* for windows.x86_64 is not latest

16.14 5.8.2

- Fix issue: the option *–exclude* in command *obfuscate* could not exclude *.py* files
- Refine command *pack*

16.15 5.8.1

- Fix issue: check license failed if there is no environment variable *HOME* in linux platform
- Add new value *3* for option *–package-runtime*, the bootstrap code will always use relative import with an extra leading dot
- The command *runtime* also generates bootstrap script *pytransform_bootstrap.py*
- Add option *–inside* in command *runtime* to generate bootstrap package *pytransform_bootstrap*
- Document how to run unittest of obfuscated scripts, refer to https://pyarmor.readthedocs.io/en/latest/advanced.html#run-unittest-of-obfuscated-scripts

16.16 5.8.0

- Move the license file of pyarmor from the install path of pyarmor package to user home path *~/.pyarmor*
- Refine error messages so that the users could solve most of problems by the hints.
- Refine command *pack*, use hook *hook-pytransform.py* to add the runtime files.
- The command *pack* supports customized spec file, refer to https://pyarmor.readthedocs.io/en/latest/advanced.html#bundle-obfuscated-scripts-with-customized-spec-file
- In runtime module *pytransform*, the functions may raise *Exception* instead of *PytransformError* in some cases.
- In command *register*, add option *–legency* to store *license.lic* in the traditional way
- Fix platform name issue: in some linux platforms the platform name may not be right

16.17 5.7.10

- Fix new linux platform *centos6.x86_64* issue: raise TypeError when run *pyarmor* twice.

16.18 5.7.9

- Support new linux platform *centos6.x86_64*, arch is x86_64, glibc < 2.14
- Do not print traceback if no option *–debug* specified as running *pyarmor*

16.19 5.7.8

- When the obfuscated scripts raise exception, eliminate the very long line from traceback to make it clear

16.20 5.7.7

- Fix issue: *pyarmor* load *_pytransform.dll* faild by 32-bit Python in 64-bit Windows.

16.21 5.7.6

- Add option *–update* for command *download* to update all the downloaded dynamic libraries automatically
- Fix issue: the obfuscated script raises unexpected exception when the license is expired

16.22 5.7.5

- Standardize platform names, refer to https://pyarmor.readthedocs.io/en/v5.7.5/platforms.html#standard-platform-names
- Run obfuscated scripts in multiple platforms, refer to https://pyarmor.readthedocs.io/en/v5.7.5/advanced.html#running-obfuscated-scripts-in-multiple-platforms
- Downloaded dynamic library files by command *command* will be saved in the *~/.pyarmor/platforms* other than the installed path of pyarmor package.
- Refine *platforms* folder structure according to new standard platform name
- In command *obfuscate*, *build*, *runtime*, specify the option *–platform* multiple times, so that the obfuscated scripts could run in these platforms

16.23 5.7.4

- Fix issue: command *obfuscate* fails if the option *–src* is specifed

16.24 5.7.3

- Refine `pytransform` to handle error message of core library
- Refine command online help message
- Sort the scripts being to obfuscated to fix some random errors (#143)

- Raise exception other than call *sys.exit* if *pyarmor* is called from another Python script directly
- **In the function *get_license_info* of module `pytransform`**
  - Change the value to *None* if there is no corresponding information
  - Change the key name *expired* to upper case *EXPIRED*

16.25 5.7.2

- Fix plugin codec issue (#138): 'gbk' codec can't decode byte 0x82 in position 590: illegal multibyte sequence
- Project src may be relative path base on project path
- Refine plugin and document it in details: https://pyarmor.readthedocs.io/en/v5.7.2/how-to-do.html#how-to-deal-with-plugins
- Add common option *–debug* for *pyarmor* to show more information in the console
- Project commands, for examples *build*, *cofig*, the last argument supports any valid project configuration file

16.26 5.7.1

- Add command *runtime* to generate runtime package separately
- Add the first character as alias for command *obfuscate, licenses, pack, init, config, build*
- Fix cross platform obfuscating scripts don't work issue (#136). This bug should be exists from v5.6.0 to v5.7.0 Related target platforms *armv5, android.aarch64, ppc64le, ios.arm64, freebsd, alpine, alpine.arm, poky-i586*

16.27 5.7.0

There are 2 major changes in this version:

1. The runtime files are saved in the separated folder *pytransform* as package:

```
dist/
    obf_foo.py

    pytransform/
        __init__.py
        license.lic
        pytransform.key
        ...
```

Upgrade notes:

- If you have generated new runtime file "license.lic", it should be copied to *dist/pytransform* other than *dist/*
- If you'd like to save the runtime files in the same folder with obfuscated scripts as before, obfuscating the scripts with option *package-runtime* like this:

```
pyarmor obfuscate --package-runtime=0 foo.py
pyarmor build --package-runtime=0
```

2. The bootstrap code must be in the obfuscated scripts, and it must be entry script as obfuscating.

Upgrade notes:

- If you have inserted bootstrap code into the obfuscated script *dist/foo.py* which is obfuscated but not as entry script manually. Do it by this command after v5.7.0:

```
pyarmor obfuscate --no-runtime --exact foo.py
```

- If you need insert bootstrap code into plain script, first obfuscate an empty script like this:

```
echo "" > pytransform_bootstrap.py
pyarmor obfuscate --no-runtime --exact pytransform_bootstrap.py
```

  Then import *pytransform_bootstrap* in the plain script.

Other changes:

- Change default value of project attribute *package_runtime* from 0 to 1
- Change default value of option *–package-runtime* from 0 to 1 in command *obfuscate*
- Add option *–no-runtime* for command *obfuscate*
- Add optioin *–disable-restrict-mode* for command *licenses*

16.28 5.6.8

- Add option *–package-runtime* in command *obfuscate*, *config* and *build*
- Add attribute *package_runtime* for project
- Refine default cross protection code
- Remove deprecated flag for option *–src* in command *obfuscate*
- Fix help message errors in command *obfuscate*

16.29 5.6.7

- Fix issue (#129): “Invalid input packet” on raspberry pi (armv7)
- Add new obfuscation mode: obf_code == 2

16.30 5.6.6

- Remove unused exported symbols from core libraries

16.31 5.6.5

- Fix win32 issue: verify license failed in some cases
- Refine core library to improve security

16.32 5.6.4

- Fix segmentation fault issue for Python 3.8

16.33 5.6.3

- Add option *-x* in command *licenses* to save extra data in the license file. It's mainly used to extend license type.

16.34 5.6.2

- Fix *pyarmor-webui* start issue in some cases: can't import name '_project'

16.35 5.6.1

- The command *download* will check the version of dynamic library to be sure it works with the current PyArmor.

16.36 5.6.0

In this version, new *private capsule*, which use 2048 bits RSA key to improve security for obfucated scripts, is introduced for purchased users. All the trial versions still use one same *public capsule* which use 1024 bits RSA keys. After purchasing PyArmor, a keyfile which includes license key and *private capsule* will be sent to customer by email.

For the previous purchased user, the old private capsules which are generated implicitly by PyArmor after registered PyArmor still work, but maybe not supported later. Contact jondy.zhao@gmail.com if you'd like to use new *private capsule*.

The other changes:

- Command *register* are refined according to new private capsule

Upgrade Note for Previous Users

There are 2 solutions:

1. Still use old license code.

It's recommanded that you have generated some customized "license.lic" for the obfuscated scrips and these "license.lic" files have been issued to your customers. If use new key file, all the previous "license.lic" does not work, you need generate new one and resend to your customers.

Actually the command *pip install –upgrade pyarmor* does not overwrite the purchased license code, you need not run command *pyarmor register* again. It should still work, you can check it by run *pyarmor -v*.

Or in any machine in which old version pyarmor is running, compress the following 2 files to one archive "pyarmor-regfile.zip":

- license.lic, which locates in the installed path of pyarmor
- .pyarmor_capsule.zip, which locates in the user HOME path

Then register this keyfile in the new version of pyarmor

```
pyarmor register pyarmor-regfile.zip
```

2. Use new key file.

It's recommanded that you have not yet issued any customized "license.lic" to your customers.

Forward the purchased email received from MyCommerce to jondy.zhao@gmail.com, and the new key file will be sent to the registration email, no fee for this upgrading.

16.37 5.5.7

- Fix webui bug: raise "name 'output' is not defined" as running *packer*

16.38 5.5.6

- Add new restrict mode 2, 3 and 4 to improve security of the obfuscated scripts, refer to *Restrict Mode*
- In command *obfuscate*, option *–restrict* supports new value 2, 3 and 4
- In command *config*, option *–disable-restrict-mode* is deprecrated
- In command *config*, add new option *–restrict*
- In command *obfuscate* the last argument could be a directory

16.39 5.5.5

- Win32 issue: the obfuscated scripts will print extra message.

16.40 5.5.4

- Fix issue: the output path isn't correct when building a package with multiple entries
- Fix issue: the obfuscated scripts raise SystemError "unknown opcode" if advanced mode is enabled in some MacOS machines

16.41 5.5.3

- Fix issue: it will raise error "Invalid input packet" to import 2 independent obfuscated packages in 64-bit Windows.

16.42 5.5.2

- Fix bug of command *pack*: the obfuscated modules aren't packed into the bundle if there is an attribute *_code_cache* in the *a.pure*

16.43 5.5.1

- Fix bug: it could not obfuscate more than 32 functions in advanced mode even pyarmor isn't trial version.
- In command *licenses*, the output path of generated license file is truncated if the registration code is too long, and all the invalid characters for path are removed.

16.44 5.5.0

- Fix issue: Warning: code object xxxx isn't wrapped (#59)
- Refine command *download*, fix some users could not download library file from pyarmor.dashingsoft.com
- Introduce advanced mode for x86/x64 arch, it has some limitations in trial version
- Add option *–advanced* for command *obfuscate*
- Add new property *advanced_mode* for project

A new feature **Advanced Mode** is introduced in this version. In this mode the structure of PyCode_Type is changed a little to improve the security. And a hook also is injected into Python interpreter so that the modified code objects could run normally. Besides if some core Python C APIs are changed unexpectedly, the obfuscated scripts in advanced mode won't work. Because this feature is highly depended on the machine instruction set, it's only available for x86/x64 arch now. And pyarmor maybe makes mistake if Python interpreter is compiled by old gcc or some other *C* compiles. It's welcome to report the issue if Python interpreter doesn't work in advanced mode.

Take this into account, the advanced mode is disabled by default. In order to enable it, pass option *–advanced* to command *obfuscate*. But in next minor version, this mode may be enable by default.

Upgrade Notes:

Before upgrading, please estimate Python interpreter in product environments to be sure it works in advanced mode. Here is the guide

https://github.com/dashingsoft/pyarmor-core/tree/v5.3.0/tests/advanced_mode/README.md

It is recommended to upgrade in the next minor version.

16.45 5.4.6

- Add option *–without-license* for command *pack*. Sample usage refer to https://pyarmor.readthedocs.io/en/latest/advanced.html#bundle-obfuscated-scripts-to-one-executable-file
- Add option *–debug* for command *pack*. If this option isn't set, all the build files will be removed after packing.

16.46 5.4.5

- Enhancement: In Linux support to get the serial number of NVME harddisk
- Fix issue: After run command *register*, pyarmor could not generate capsule if there is *license.lic* in the current path

16.47 5.4.4

- Fix issue: In Linux could not get the serial number of SCSI harddisk
- Fix issuse: In Windows the serial number is not right if the leading character is alpha number

16.48 5.4.3

- Add function *get_license_code* in runtime module *pytransform*, which mainly used in plugin to extend license type. Refer to https://pyarmor.readthedocs.io/en/latest/advanced.html#using-plugin-to-extend-license-type
- Fix issue: the command *download* always shows trial version

16.49 5.4.2

- Option *–exclude* can use multiple times in command *obfuscate*
- Exclude build path automatically in command *pack*

16.50 5.4.1

- New feature: do not obfuscate functions which name starts with *lambda_*
- Fix issue: it will raise *Protection Fault* as packing obfuscated scripts to one file

16.51 5.4.0

- Do not obfuscate lambda functions by default
- Fix issue: local variable *platname* referenced before assignment

16.52 5.3.13

- Add option *–url* for command *download*

16.53 5.3.12

- Add integrity checks for the downloaded binaries (#85)

16.54 5.3.11

- Fix issue: get wrong harddisk's serial number for some special cases in Windows

16.55 5.3.10

- Query harddisk's serial number without administrator in Windows

16.56 5.3.9

- Remove the leading and trailing whitespace of harddisk's serial number

16.57 5.3.8

- Fix non-ascii path issue in Windows

16.58 5.3.7

- Fix bug: the bootstrap code isn't inserted correctly if the path of entry script is absolute path.

16.59 5.3.6

- Fix bug: protection code can't find the correct dynamic library if distributing obfuscated scripts to other platforms.
- Document how to distribute obfuscated scripts to other platforms https://pyarmor.readthedocs.io/en/latest/advanced.html#distributing-obfuscated-scripts-to-other-platform

16.60 5.3.5

- The bootstrap code could run many times in same Python interpreter.
- Remove extra . from the bootstrap code of *__init__.py* as building project without runtime files.

16.61 5.3.4

- Add command *download* used to download platform-dependent dynamic libraries
- Keep shell line for obfuscated entry scripts if there is first line starts with *#!*
- Fix issue: if entry script is not in the *src* path, bootstrap code will not be inserted.

16.62 5.3.3

- Refine *benchmark* command
- Document the performance of obfuscated scripts https://pyarmor.readthedocs.io/en/latest/performance.html
- Add command *register* to take registration code effects

- Rename trial license file *license.lic* to *license.tri*

16.63 5.3.2

- Fix bug: if there is only one comment line in the script it will raise IndexError as obfuscating this script.

16.64 5.3.1

- Refine *pack* command, and make output clear.
- Document plugin usage to extend license type for obufscated scripts. Refer to https://pyarmor.readthedocs.io/en/latest/advanced.html#using-plugin-to-extend-license-type

16.65 5.3.0

- In the trial version of PyArmor, it will raise error as obfuscating the code object which size is greater than 32768 bytes.
- Add option *–plugin* in command *obfuscate*
- Add property *plugins* for Project, and add option *–plugin* in command *config*
- Change default build path for command *pack*, and do not remove it after command finished.

16.66 5.2.9

- Fix segmentation fault issue for python3.5 and before: run too big obfuscated code object (>65536 bytes) will crash (#67)
- Fix issue: missing bootstrap code for command *pack* (#68)
- Fix issue: the output script is same as original script if obfuscating scripts with option *–exact*

16.67 5.2.8

- Fix issue: *pyarmor -v* complains *not enough arguments for format string*

16.68 5.2.7

- In command *obfuscate* add new options *–exclude*, *–exact*, *–no-bootstrap*, *–no-cross-protection*.
- In command *obfuscate* deprecate the options *–src*, *–entry*, *–cross-protection*.
- In command *licenses* deprecate the option *–bind-file*.

16.69 5.2.6

- Fix issue: raise codec exception as obfuscating the script of utf-8 with BOM
- Change the default path to user home for command *capsule*
- Disable restrict mode by default as obfuscating special script *__init__.py*
- Refine log message

16.70 5.2.5

- Fix issue: raise IndexError if output path is '.' as building project
- For Python3 convert error message from bytes to string as checking license failed
- Refine version information

16.71 5.2.4

- Fix arm64 issue: verify rsa key failed when running the obufscated scripts(#63)
- Support ios (arm64) and ppc64le for linux

16.72 5.2.3

- Refine error message when checking license failed
- Fix issue: protection code raises ImportError in the package file *__init.py__*

16.73 5.2.2

- Improve the security of dynamic library.

16.74 5.2.1

- Fix issue: in restrict mode the bootstrap code in *__init__.py* will raise exception.
- Add option *–cross-protection* in command *obfuscate*

16.75 5.2.0

- Use global capsule as default capsule for project, other than creating new one for each project
- Add option *–obf-code*, *–obf-mod*, *–wrap-mode*, *–cross-protection* in command *config*
- Add new attributes for project: *obf_code*, *obf_mod*, *wrap_mode*, *cross_protection*
- Deprecrated project attributes *obf_code_mode*, *obf_module_mode*, use *obf_code*, *obf_mod*, *wrap_mode* instead

- Change the behaviours of *restrict mode*, refer to https://pyarmor.readthedocs.io/en/latest/advanced.html#restrict-mode
- Change option *–restrict* in command *obfuscate* and *licenses*
- Remove option *–no-restrict* in command *obfuscate*
- Remove option *–clone* in command *init*

16.76 5.1.2

- Improve the security of PyArmor self

16.77 5.1.1

- Refine the procedure of encrypt script
- Reform module *pytransform.py*
- Fix issue: it will raise exception if no entry script when obfuscating scripts
- Fix issue: 'gbk' codec can't decode byte 0xa1 in position 28 (#51)
- Add option *–upgrade* for command *capsule*
- Merge runtime files *pyshield.key*, *pyshield.lic* and *product.key* into *pytransform.key*

Upgrade notes

The capsule created in this version will include a new file *pytransform.key* which is a replacement for 3 old runtime files: *pyshield.key*, *pyshield.lic* and *product.key*.

The old capsule which created in the earlier version still works, it stills use the old runtime files. But it's recommended to upgrade the old capsule to new version. Just run this command:

```
pyarmor capsule --upgrade
```

All the license files generated for obfuscated scripts by old capsule still work, but all the scripts need to be obfuscated again to take new capsule effects.

16.78 5.1.0

- Add extra code to protect dynamic library *_pytransform* when obfuscating entry script
- Fix compling error when obfuscating scripts in windows for Python 26/30/31 (newline issue)

16.79 5.0.5

- Refine *protect_pytransform* to improve security, refer to https://pyarmor.readthedocs.io/en/latest/security.html

16.80 5.0.4

- Fix *get_expired_days* issue, remove decorator *dllmethod*
- Refine output message of *pyarmor -v*

16.81 5.0.3

- Add option *-q*, *–silent*, suppress all normal output when running any PyArmor command
- Refine runtime error message, make it clear and more helpful
- Add new function *get_hd_info* in module *pytransform* to get hardware information
- Remove function *get_hd_sn* from module *pytransform*, use *get_hd_info* instead
- Remove useless function *version_info*, *get_trial_days* from module *pytransform*
- Remove attribute *lib_filename* from module *pytransform*, use *_pytransform._name* instead
- Add document https://pyarmor.readthedocs.io/en/latest/pytransform.html
- Refine document https://pyarmor.readthedocs.io/en/latest/security.html

16.82 5.0.2

- Export *lib_filename* in the module pytransform in order to protect dynamic library *_pytransform*. Refer to https://pyarmor.readthedocs.io/en/latest/security.html

16.83 5.0.1

Thanks to GNU lightning, from this version, the core routines are protected by JIT technicals. That is to say, there is no binary code in static file for core routines, they're generated in runtime.

Besides, the pre-built dynamic library for linux arm32/64 are packed into the source package.

Fixed issues:

- The module *multiprocessing* starts new process failed in obfuscated script:

  AttributeError: '__main__' object has no attribute 'f'

16.84 4.6.3

- Fix backslash issue when running *pack* command with *PyInstaller*
- When PyArmor fails, if *sys.flags.debug* is not set, only print error message, no traceback printed

16.85 4.6.2

- Add option *–options* for command *pack*
- For Python 3, there is no new line in the output when *pack* command fails

16.86 4.6.1

- Fix license issue in 64-bit embedded platform

16.87 4.6.0

- Fix crash issue for special code object in Python 3.6

16.88 4.5.5

- Fix stack overflow issue

16.89 4.5.4

- Refine platform name to search dynamic library *_pytransform*

16.90 4.5.3

- Print the exact message when checking license failed to run obfuscated scripts.

16.91 4.5.2

- Add documentation https://pyarmor.readthedocs.io/en/latest/
- Exclude *dist*, *build* folder when executing *pyarmor obfuscate –recursive*

16.92 4.5.1

- Fix #41: can not find dynamic library *_pytransform*

16.93 4.5.0

- Add anti-debug code for dynamic library *_pytransform*

16.94 4.4.2

- Change default capsule to user home other than the source path of *pyarmor*

16.95 4.4.2

This patch mainly changes webui, make it simple more:

- WebUI : remove source field in tab Obfuscate, and remove ipv4 field in tab Licenses
- WebUI Packer: remove setup script, add output path, only support PyInstaller

16.96 4.4.1

- Support Py2Installer by a simple way
- For command *obfuscate*, get default *src* and *entry* from first argument, *–src* is not required.
- Set no restrict mode as default for new project and command *obfuscate*, *licenses*

16.97 4.4.0

- Pack obfuscated scripts by command *pack*

In this version, introduces a new command *pack* used to pack obfuscated scripts with *py2exe* and *cx_Freeze*. Once the setup script of *py2exe* or *cx_Freeze* can bundle clear python scripts, *pack* could pack obfuscated scripts by single command: *pyarmor pack –type cx_Freeze /path/to/src/main.py*

- Pack obfuscated scripts by WebUI packer

WebUI is well reformed, simple and easy to use.

http://pyarmor.dashingsoft.com/demo/index.html

16.98 4.3.4

- Fix start pyarmor issue for *pip install* in Python 2

16.99 4.3.3

- Fix issue: missing file in wheel

16.100 4.3.2

- Fix *pip* install issue in MacOS
- Refine sample scripts to make workaround for py2exe/cx_Freeze simple

16.101 4.3.1

- Fix typos in examples
- Fix bugs in sample scripts

16.102 4.3.0

In this version, there are three significant changes:

[Simplified WebUI](http://pyarmor.dashingsoft.com/demo/index.html) [Clear Examples](src/examples/README.md), quickly understand the most features of Pyarmor [Sample Shell Scripts](src/examples), template scripts to obfuscate python source files

- Simply webui, easy to use, only input one filed to obfuscate python scripts
- The runtime files will be always saved in the same path with obfuscated scripts
- Add shell scripts *obfuscate-app*, *obfuscate-pkg*, *build-with-project*, *build-for-2exe* in *src/examples*, so that users can quickly obfuscate their python scripts by these template scripts.
- If entry script is *__init__.py*, change the first line of bootstrap code *from pytransform import pyarmor runtime* to *from .pytransform import pyarmor runtime*
- Rewrite examples/README.md, make it clear and easy to understand
- Do not generate entry scripts if only runtime files are generated
- Remove choice *package* for option *–type* in command *init*, only *pkg* reserved.

16.103 4.2.3

- Fix *pyarmor-webui* can not start issue
- Fix *runtime-path* issue in webui
- Rename platform name *macosx_intel* to *macosx_x86_64* (#36)

16.104 4.2.2

- Fix webui import error.

16.105 4.2.1

- Add option *–recursive* for command *obfuscate*

16.106 4.1.4

- Rewrite project long description.

16.107 4.1.3

- Fix Python3 issue for *get_license_info*

16.108 4.1.2

- Add function *get_license_info* in *pytransform.py* to show license information

16.109 4.1.1

- Fix import *main* from *pyarmor* issue

16.110 4.0.3

- Add command *capsule*
- Find default capsule in the current path other than *–src* in command *obfuscate*
- Fix pip install issue #30

16.111 4.0.2

- Rename *pyarmor.py* to *pyarmor-depreted.py*
- Rename *pyarmor2.py* to *pyarmor.py*
- Add option *–capsule*, *-disable-restrict-mode* and *–output* for command *licenses*

16.112 4.0.1

- Add option *–capsule* for command *init*, *config* and *obfuscate*
- Deprecate option *–clone* for command *init*, use *–capsule* instead
- Fix *sys.settrace* and *sys.setprofile* issues for auto-wrap mode

16.113 3.9.9

- Fix segmentation fault issues for *asyncio*, *typing* modules

16.114 3.9.8

- Add documentation for examples (examples/README.md)

16.115 3.9.7

- Fix windows 10 issue: access violation reading 0x000001ED00000000

16.116 3.9.6

- Fix the generated license bind to fixed machine in webui is not correct
- Fix extra output path issue in webui

16.117 3.9.5

- Show registration code when printing version information

16.118 3.9.4

- Rewrite long description of package in pypi

16.119 3.9.3

- Fix issue: *__file__* is not really path in main code of module when import obfuscated module

16.120 3.9.2

- Replace option *–disable-restrict-mode* with *–no-restrict* in command *obfuscate*
- Add option *–title* in command *config*
- Change the output path of entry scripts when entry scripts belong to package
- Refine document *user-guide.md* and *mechanism.md*

16.121 3.9.1

- Add option *–type* for command *init*
- Refine document *user-guide.md* and *mechanism.md*

16.122 3.9.0

This version introduces a new way *auto-wrap* to protect python code when it's imported by outer scripts.

Refer to [Mechanism Without Restrict Mode](src/mechanism.md#mechanism-without-restrict-mode)

- Add new mode *wrap* for *–obf-code-mode*

- Remove *func.__refcalls__* in *__wraparmor__*
- Add new project attribute *is_package*
- Add option *–is-package* in command *config*
- Add option *–disable-restrict-mode* in command *obfuscate*
- Reset *build_time* when project configuration is changed
- Change output path when *is_package* is set in command *build*
- Change default value of project when find *__init__.py* in comand *init*
- Project attribute *entry* supports absolute path

16.123 3.8.10

- Fix shared code object issue in *__wraparmor__*

16.124 3.8.9

- Clear frame as long as *tb* is not *Py_None* when call *__wraparmor__*
- Generator will not be obfucated in *__wraparmor__*

16.125 3.8.8

- Fix bug: the *frame.f_locals* still can be accessed in callback function

16.126 3.8.7

- The *frame.f_locals* of *wrapper* and wrapped function will return an empty dictionary once *__wraparmor__* is called.

16.127 3.8.6

- The *frame.f_locals* of *wrapper* and wrapped function return an empty dictionary, all the other frames still return original value.

16.128 3.8.5

- The *frame.f_locals* of all frames will always return an empty dictionary to protect runtime data.
- Add extra argument *tb* when call *__wraparmor__* in decorator *wraparmor*, pass None if no exception.

16.129 3.8.4

- Do not touch *frame.f_locals* when raise exception, let decorator *wraparmor* to control everything.

16.130 3.8.3

- Fix issue: option *–disable-restrict-mode* doesn't work in command *licenses*
- Remove freevar *func* from *frame.f_locals* when raise exception in decorator *wraparmor*

16.131 3.8.2

- Change module filename to <*frozen modname*> in traceback, set attribute *__file__* to real filename when running obfuscated scripts.

16.132 3.8.1

- Try to access original func_code out of decorator *wraparmor* is forbidden.

16.133 3.8.0

- Add option *–output* for command *build*, it will override the value in project configuration file.
- Fix issue: defalut project output path isn't relative to project path.
- Remove extra file "product.key" after obfuscating scripts.

16.134 3.7.5

- Remove dotted name from filename in traceback, if it's not a package.

16.135 3.7.4

- Strip *__init__* from filename in traceback, replace it with package name.

16.136 3.7.3

- Remove brackets from filename in traceback, and add dotted prefix.

16.137 3.7.2

- Change filename in traceback to <*frozen [modname]*>, other than original filename

16.138 3.7.1

- Fix issue #12: module attribute *__file__* is filename in build machine other than filename in target machine.
- Builtins function *__wraparmor__* only can be used in the decorator *wraparmor*

16.139 3.7.0

- Fix issue #11: use decorator "wraparmor" to obfuscate func_code as soon as function returns.
- Document usage of decorator "wraparmor", refer to **src/user-guide.md#use-decorator-to-protect-code-objects-when-disable-restrict-mode**

16.140 3.6.2

- Fix issue #8 (Linux): option –manifest broken in shell script

16.141 3.6.1

- Add option "Restrict Mode" in web ui
- Document restrict mode in details (user-guide.md)

16.142 3.6.0

- Introduce restrict mode to avoid obfuscated scripts observed from no obfuscated scripts
- Add option –disable-restrict-mode for command "config"

16.143 3.5.1

- Support pip install pyarmor

16.144 3.5.0

- Fix Python3.6 issue: can not run obfuscated scripts, because it uses a 16-bit wordcode instead of bytecode
- Fix Python3.7 issue: it adds a flag in pyc header
- Fix option –obf-module-mode=none failed
- Add option –clone for command "init"
- Generate runtime files to separate path "runtimes" when project runtime-path is set
- Add advanced usages in user-guide

16.145 3.4.3

- Fix issue: raise exception when project entry isn't obfuscated

16.146 3.4.2

- Add webui to manage project

16.147 3.4.1

- Fix README.rst format error.
- Add title attribute to project
- Print new command help when option is -h, –help

16.148 3.4.0

Pyarmor v3.4 introduces a group new commands. For a simple package, use command **obfuscate** to obfuscate scripts directly. For complicated package, use Project to manage obfuscated scripts.

Project includes 2 files, one configure file and one project capsule. Use manifest template string, same as MANIFEST.in of Python Distutils, to specify the files to be obfuscated.

To create a project, use command **init**, use command **info** to show project information. **config** to update project settings, and **build** to obfuscate the scripts in the project.

Other commands, **benchmark** to metric performance, **hdinfo** to show hardware information, so that command **licenses** can generate license bind to fixed machine.

All the old commands **capsule**, **encrypt**, **license** are deprecated, and will be removed from v4.

A new document src/user-guide.md is written for this new version.

16.149 3.3.1

- Remove unused files in distribute package

16.150 3.3.0

In this version, new obfuscate mode 7 and 8 are introduced. The main difference is that obfuscated script now is a normal python file (.py) other than compiled script (.pyc), so it can be used as common way.

Refer to https://github.com/dashingsoft/pyarmor/blob/v3.3.0/src/mechanism.md

- Introduce new mode: 7, 8
- Change default mode from 3 to 8
- Change benchmark.py to test new mode

- Update webapp and tutorial
- Update usage
- Fix issue of py2exe, now py2exe can work with python scripts obfuscated by pyarmor
- Fix issue of odoo, now odoo can load python modules obfuscated by pyarmor

16.151 3.2.1

- Fix issue: the traceback of an exception contains the name "<pytransform>" instead of the correct module name
- Fix issue: All the constant, co_names include function name, variable name etc still are in clear text. Refer to https://github.com/dashingsoft/pyarmor/issues/5

16.152 3.2.0

From this version, a new obfuscation mode is introduced. By this way, no import hooker, no setprofile, no settrace required. The performance of running or importing obfuscation python scripts has been remarkably improved. It's significant for Pyarmor.

- Use this new mode as default way to obfuscate python scripts.
- Add new script "benchmark.py" to check performance in target machine: python benchmark.py
- Change option "–bind-disk" in command "license", now it must be have a value

16.153 3.1.7

- Add option "–bind-mac", "–bind-ip", "–bind-domain" for command "license"
- Command "hdinfo" show more information(serial number of hdd, mac address, ip address, domain name)
- Fix the issue of dev name of hdd for Banana Pi

16.154 3.1.6

- Fix serial number of harddisk doesn't work in mac osx.

16.155 3.1.5

- Support MACOS

16.156 3.1.4

- Fix issue: load _pytransfrom failed in linux x86_64 by subprocess.Popen
- Fix typo in error messge when load _pytransfrom failed.

16.157 3.1.3

A web gui interface is introduced as Pyarmor WebApp and support MANIFEST.in

- In encrypt command, save encrypted scripts with same file structure of source.
- Add a web gui interface for pyarmor.
- Support MANIFEST.in to list files for command encrypt
- Add option –manifest, file list will be written here
- DO NOT support absolute path in file list for command encrypt
- Option –main support format "NAME:ALIAS.py"

16.158 3.1.2

- Refine decrypted mechanism to improve performance
- Fix unknown opcode problem in recursion call
- Fix wrapper scripts generated by -m in command 'encrypt' doesn't work
- Raise ImportError other than PytransformError when import encrypted module failed

16.159 3.1.1

In this version, introduce 2 extra encrypt modes to improve performance of encrypted scripts.

- Fix issue when import encrypted package
- Add encrypted mode 2 and 3 to improve performance
- Refine module pyimcore to improve performance

16.160 3.0.1

It's a milestone for Pyarmor, from this version, use ctypes import dynamic library of core functions, other than by python extensions which need to be built with every python version.

Besides, in this version, a big change which make Pyarmor could avoid soure script got by c debugger.

- Use ctypes load core library other than python extentions which need built for each python version.
- "__main__" block not running in encrypted script.
- Avoid source code got by c debugger.
- Change default outoupt path to "build" in command "encrypt"
- Change option "–bind" to "–bind-disk" in command "license"
- Document usages in details

16.161 2.6.1

- Fix encrypted scripts don't work in multi-thread framework (Django).

16.162 2.5.5

- Add option '-i' for command 'encrypt' so that the encrypted scripts will be saved in the original path.

16.163 2.5.4

- Verbose tracelog when checking license in trace mode.
- In license command, change default output filename to "license.lic.txt".
- Read bind file when generate license in binary mode other than text mode.

16.164 2.5.3

- Fix problem when script has line "from __future__ import with_statement"
- Fix error when running pyarmor by 32bit python on the 64bits Windows.
- (Experimental)Support darwin_15-x86_64 platform by adding extensions/pytransform-2.3.3.darwin_15.x86_64-py2.7.so

16.165 2.5.2

- License file can mix expire-date with fix file or fix key.
- Fix log error: not enough arguments for format string

16.166 2.5.1

- License file can bind to ssh private key file or any other fixed file.

16.167 2.4.1

- Change default extension ".pyx" to ".pye", because it confilcted with CPython.
- Custom the extension of encrypted scripts by os environment variable: PYARMOR_EXTRA_CHAR
- Block the hole by which to get bytescode of functions.

16.168 2.3.4

- The trial license will never be expired (But in trial version, the key used to encrypt scripts is fixed).

16.169 2.3.3

- Refine the document

16.170 2.3.2

- Fix error data in examples of wizard

16.171 2.3.1

- Implement Run function in the GUI wizard
- Make license works in trial version

16.172 2.2.1

- Add a GUI wizard
- Add examples to show how to use pyarmor

16.173 2.1.2

- Fix syntax-error when run/import encrypted scripts in linux x86_64

16.174 2.1.1

- Support armv6

16.175 2.0.1

- Add option '–path' for command 'encrypt'
- Support script list in the file for command 'encrypt'
- Fix issue to encrypt an empty file result in pytransform crash

16.176 1.7.7

- Add option '–expired-date' for command 'license'
- Fix undefined 'tfm_desc' for arm-linux
- Enhance security level of scripts

16.177 1.7.6

- Print exactaly message when pyarmor couldn't load extension "pytransform"
- Fix problem "version 'GLIBC_2.14' not found"
- Generate "license.lic" which could be bind to fixed machine.

16.178 1.7.5

- Add missing extensions for linux x86_64.

16.179 1.7.4

- Add command "licene" to generate more "license.lic" by project capsule.

16.180 1.7.3

- Add information for using registration code

16.181 1.7.2

- Add option –with-extension to support cross-platform publish.
- Implement command "capsule" and add option –with-capsule so that we can encrypt scripts with same capsule.
- Remove command "convert" and option "-K/–key"

16.182 1.7.1

- Encrypt pyshield.lic when distributing source code.

16.183 1.7.0

- Enhance encrypt algorithm to protect source code.
- Developer can use custom key/iv to encrypt source code
- Compiled scripts (.pyc, .pyo) could be encrypted by pyshield
- Extension modules (.dll, .so, .pyd) could be encrypted by pyshield

CHAPTER 17

Indices and tables

- genindex
- modindex
- search

Index

G

get_expired_days() (*built-in function*), 61
get_hd_info() (*built-in function*), 62
get_license_code() (*built-in function*), 62
get_license_info() (*built-in function*), 61

P

PytransformError, 61