PyArmor Documentation

Release 5.6.0

Jondy Zhao

Oct 22, 2019

Contents

Version PyArmor 5.6

Homepage http://pyarmor.dashingsoft.com/

Contact jondy.zhao@gmail.com

Authors Jondy Zhao

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 is available, for example:

```
pip install pyarmor
```

or upgrade to a newer version:

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

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 a simple 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`, and of `pyarmor-webui` is *`pyarmor-folder`*`/pyarmor-webui.py`.

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:

```
myscript.py

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

The rest files called `Runtime Files`, all of them are 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, all the *Runtime Files* need to be distributed to end users too.

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

2.3 Generating License For Obfuscated Scripts

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

By default there is `dist/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 code-001
```

PyArmor generates new license file:

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

Overwrite default license with new one:

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

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/

cd dist/
python myscript.py
```

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 write what ever you want by Python. Refer to *Runtime Module pytransform* for more information.

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

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.

3.1 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 not return 0.

get_license_info()

Get license information of obfuscated scripts.

It returns a dict with keys *expired*, *CODE*, *IFMAC*.

The value of *expired* is == -1 means no time limitation.

Raise `PytransformError` 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 `PytransformError` 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

Raise `PytransformError` if something is wrong.

`HT_HARDDISK, HT_IFMAC`

Constant for *hdtype* when calling `get_hd_info()`

3.2 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 PytransformError as e:
    print(e)
```

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

CHAPTER 4

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.

4.1 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 5

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.

Checking the performance for other mode by this way:

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

Look at the scripts used by this command:

```
pyarmor benchmark --debug
```

CHAPTER 6

Understanding Obfuscated Scripts

6.1 Global Capsule

The `.pyarmor_capsule.zip` in the `HOME` path called *Global Capsule*. It's created implicitly when executing command `pyarmor obfuscate`. *PyArmor* will read data from *Global Capsule* when obfuscating scripts or generating licenses for obfuscated scripts.

6.2 Obfuscated Scripts

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

```
myscript.py
mymodule.py

pytransform.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...')
```

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

```
from pytransform import pyarmor_runtime
pyarmor_runtime()

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

6.3 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()
```

6.4 Runtime Files

Except obfuscated scripts, all the other files are called *Runtime Files*:

- *pytransform.py*, a normal python module
- *_pytransform.so*, or *_pytransform.dll*, or *_pytransform.dylib* a dynamic library implements core functions
- *pytransform.key*, data file
- *license.lic*, the license file for obfuscated scripts

All of them are required to run obfuscated scripts.

6.5 The License File for Obfuscated Script

There is a special runtime file *license.lic*. The default one, which generated as executing `pyarmor obfuscate`, allows obfuscated scripts run in any machine and never expired.

To change this behaviour, use command `pyarmor licenses` to generate new *license.lic* and overwrite the default one.

6.6 Key Points to Use Obfuscated Scripts

- The obfuscated script is a normal python script, so it can be seamless to replace original script.
- There is only one thing changed, the following code must be run before using any obfuscated script:

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

  It must in the obfuscated script and only be called one time in the same Python interpreter. It will create some builtin function to deal with obfuscated code.

- The extra runtime file *pytransform.py* must be in any Python path in target machine. *pytransform.py* need load dynamic library *_pytransform* by *ctypes*. It may be

  - *_pytransform.so* in Linux
  - *_pytransform.dll* in Windows
  - *_pytransform.dylib* in MacOS

  This file is dependent-platform, download the right one to the same path of *pytransform.py* according to target platform. All the prebuilt dynamic libraries list here Support Platfroms

- By default *pytransform.py* search dynamic library *_pytransform* in the same path. Check *pytransform._load_library* to find the details.

- All the other *Runtime Files* should in the same path as dynamic library *_pytransform*
- If *Runtime Files* locate in some other path, change *Bootstrap Code*:

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

6.7 Running Obfuscated Scripts

The obfuscated scripts is a normal python script, it can be run by normal python interpreter:

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

Firt *Bootstrap Code* is executed:

- Import *pyarmor_runtime* from *pytransform.py*
- **Execute *pyarmor_runtime***
  - Load dynamic library *_pytransform* by *ctypes*
  - Check *license.lic* in the same path
  - Add there builtin functions *__pyarmor__*, *__enter_armor__*, *__exit_armor__*

After that:

- Call *__pyarmor__* to import the obfuscated module.
- Call *__enter_armor__* to restore code object of function before executing each function
- Call *__exit_armor__* to obfuscate code object of function after each function return

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

6.8 Search path for *license.lic* and *pytransform.key*

As the obfuscated script is running, it first searches the current path, then search the path of runtime module *pytransform.py* to find the file *license.lic* and *pytransform.key*.

Sometimes there is any of *license.lic* or *pytransform.py* in the current path, but it's not for obfuscated scripts. In this case, try to run obfuscated scripts, it will report this error:

```
Invalid input packet.
Check license failed.
```

6.9 Two types of *license.lic*

In PyArmor, there are 2 types of *license.lic*

- *license.lic* of PyArmor, which locates in the source path of PyArmor. It's required to run *pyarmor*
- *license.lic* of Obfuscated Scripts, which is generated as obfuscating scripts by the end user of PyArmor. It's required to run the obfuscated scripts.

The relation between 2 *license.lic* is:

```
license.lic of PyArmor --> .pyarmor_capsule.zip --> license.lic of Obfuscated Scripts
```

When obfuscating scripts with command *pyarmor obfuscate* or *pyarmor build*, the *Global Capsule* is used implicitly. If there is no *Global Capsule*, PyArmor will read *license.lic* of PyArmor as input to generate the *Global Capsule*.

When runing command *pyarmor licenses*, it will generate a new *license.lic* for obfuscated scripts. Here the *Global Capsule* will be as input file to generate this *license.lic* of Obfuscated Scripts.

CHAPTER 7

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.

7.1 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 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 if there are more than about 30 functions in one module, this module could not be obfuscated by advanced mdoe (It still could be obfuscated by non-advanced mode).

7.2 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.

7.3 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.

7.4 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_algorithm( original_code  );
sprintf( buffer, "__pyarmor__(__name__, __file__, b'%s', 1)", obfuscated_code );
```

- obf_mod == 0

In this mode, 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)
```

They're eanbled by default. And they only could be changed in the project, refer to *Obfuscating Scripts With Different Modes*

7.5 Restrict Mode

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.

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. It could be disabled by this way if required:

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

Also obfuscating the scripts by other restrict mode:

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

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

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.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...')
```

All the other extra files called *Runtime Files*, which are required to run or import obfuscated scripts. So long as runtime files are in any Python path, 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 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 `dist/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)
{
    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
```

(continues on next page)

(continued from previous page)

```
    // 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;
}
```

8.3 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():
        CO_NAMES = set(['Exception', 'LoadLibrary', 'None', 'PYFUNCTYPE',
                        'PytransformError', '__file__', '_debug_mode',
```

(continues on next page)

(continued from previous page)

```
                        '_get_error_msg', '_handle', '_load_library',
                        '_pytransform', 'abspath', 'basename', 'byteorder',
                        'c_char_p', 'c_int', 'c_void_p', 'calcsize', 'cdll',
                        'dirname', 'encode', 'exists', 'exit',
                        'format_platname', 'get_error_msg', 'init_pytransform',
                        'init_runtime', 'int', 'isinstance', 'join', 'lower',
                        'normpath', 'os', 'path', 'platform', 'print',
                        'pyarmor_init', 'pythonapi', 'restype', 'set_option',
                        'str', 'struct', 'sys', 'system', 'version_info'])

        colist = []

        for name in ('dllmethod', 'init_pytransform', 'init_runtime',
                     '_load_library', 'pyarmor_init', 'pyarmor_runtime'):
            colist.append(getattr(pytransform, name).{code})

        for name in ('init_pytransform', 'init_runtime'):
            colist.append(getattr(pytransform, name).{closure}[0].cell_contents.{code}
↪)
        colist.append(pytransform.dllmethod.{code}.co_consts[1])

        for co in colist:
            if not (set(co.co_names) < CO_NAMES):
                raise RuntimeError('Unexpected pytransform.py')

    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.

CHAPTER 9

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, py2exe, py2app, cx_Freeze:

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

To solve this problem, the common solution is

1. Find all the dependenices by original scripts.
2. Add runtimes files required by obfuscated scripts to the bundle
3. Replace original scipts with obfuscated in the bundle
4. Replace entry scrirpt 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.

Depend on what tool used, there are different ways.

First obfuscate scripts to `dist/obf`:

```
pyarmor obfuscate --output dist/obf hello.py
```

9.1 Work with PyInstaller

Install `pyinstaller`:

```
pip install pyinstaller
```

Generate specfile, add the obfuscated entry script and data files required by obfuscated scripts:

```
pyinstaller --add-data dist/obf/license.lic
            --add-data dist/obf/pytransform.key
```

(continues on next page)

(continued from previous page)

```
            --add-data dist/obf/_pytransform.*
            hello.py dist/obf/hello.py
```

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

```
a.scripts[-1] = 'hello', r'dist/obf/hello.py', 'PYSOURCE'
for i in range(len(a.pure)):
    if a.pure[i][1].startswith(a.pathex[0]):
        x = a.pure[i][1].replace(a.pathex[0], os.path.abspath('dist/obf'))
        if os.path.exists(x):
            if hasattr(a.pure, '_code_cache'):
                with open(x) as f:
                    a.pure._code_cache[a.pure[i][0]] = compile(f.read(), a.pure[i][1],
↪ 'exec')
            a.pure[i] = a.pure[i][0], x, a.pure[i][2]
```

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
```

9.2 Work with py2exe

For Python3.3 and later:

```
pip install py2exe
```

Build bundle executable to `dist` with separated library:

```
build_exe --library library.zip hello.py
```

Build bundle executable with the obfuscated entry to `dist/obf/dist`, all the other obfuscated scripts should be include by `-i name` or `-p pkgname`:

```
( cd dist/obf;
  build_exe --library library.zip -i queens hello.py )
```

Update `dist/obf/library.zip`, which only includes the obfuscated scripts, merge all the dependenices files from `dist/library.zip` into it.

Copy all the files to final output:

```
cp -a dist/obf/dist/* dist/
```

Copy runtime files required by obfuscated scripts to finial output:

```
( cd dist/obf;
  cp *.key *.lic _pytransform.dll ../dist/ )
```

Check obfuscated scripts work:

```
# It works
dist/hello.exe

rm dist/license.lic

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

For Python2, write a `setup.py` and run `py2exe` as the following way:

```
python setup.py py2exe hello.py
```

9.3 Work with cx_Freeze 5

Install `cx_Freeze`:

```
pip install cx_Freeze
```

Build bundle executable to `dist`:

```
cxfreeze --target-dir=dist hello.py
```

Build bundle executable with the obfuscated entry to `dist/obf/dist`, all the other obfuscated scripts should be include by `--include-modules NAMES`:

```
cd dist/obf
cxfreeze --target-dir=dist --include-modules=queens hello.py
```

Update `dist/obf/python34.zip`, which only includes the obfuscated scripts, merge all the dependenices files from `dist/python34.zip` into it.

Copy all the files to final output:

```
cp -a dist/obf/dist/* dist/
```

Copy runtime files required by obfuscated scripts to finial output:

```
( cd dist/obf;
  cp *.key *.lic _pytransform.dll ../dist/ )
```

Check obfuscated scripts work:

```
# It works
dist/hello.exe

rm dist/license.lic

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

CHAPTER 10

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
- More convenient to manage obfuscated scripts

10.1 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:

```
pyarmor build
```

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
```

10.2 Obfuscating Scripts With Different Modes

Configure mode to obfuscate scripts:

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

Obfuscating scripts in new mode:

```
pyarmor build -B
```

10.3 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.

  Generally it's absolute path.

- 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, and 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.

- disable_restrict_mode [DEPRECRATED]

  Available values: 0, 1, None

  When it's None or 0, obfuscated scripts can not be imported from outer scripts.

  When it's set to 1, it the obfuscated scripts are allowed to be imported by outer scripts.

  By default it's set to 0.

  Refer to *Restrict Mode*

- 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

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

- obf_module_mode [DEPRECRATED]

  How to obfuscate whole code object of module:

        - none

        No obfuscate

        - des

        Obfuscate whole code object by DES algorithm

        The default value is *des*

- obf_code_mode [DEPRECRATED]

    How to obfuscate byte code of each code object:

    - none

      No obfuscate

    - des

      Obfuscate byte-code by DES algorithm

    - fast

      Obfuscate byte-code by a simple algorithm, it's faster than DES

    - wrap

      The wrap code is different from *des* and *fast*. In this mode, when code object start to execute, byte-code is restored. As soon as code object completed execution, byte-code will be obfuscated again.

    The default value is *wrap*.

- obf_code

  How to obfuscate byte code of each code object:

  - 0

    No obfuscate

  - 1

    Obfuscate each code object by default algorithm

  Refer to *Obfuscating Code Mode*

- wrap_mode

  Available values: 0, 1, None

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

  Refer to *Wrap Mode*

- obf_mod

  How to obfuscate whole code object of module:

  - 0

    No obfuscate

  - 1

    Obfuscate byte-code by DES algorithm

  Refer to *Obfuscating module Mode*

- 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 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*

CHAPTER 11

The Differences of Obfuscated Scripts

There are something changed after Python scripts are obfuscated:

- The major 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 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.
- 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__)
  ```

11.1 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 12

Advanced Topics

12.1 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 runtime files, save them 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'
```

12.2 Distributing Obfuscated Scripts To Other Platform

First list and download dynalic library of target platform by command *download*:

```
pyarmor download --list
pyarmor download linux_x86_64
```

Then specify platform name as obfuscating the scripts:

```
pyarmor obfuscate --platform linux_x86_64 foo.py
```

For project:

```
pyarmor build --platform linux_x86_64
```

12.3 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 %*
```

12.4 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 obfuscate all the scripts:

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

In the output path *dist*, there are 4 runtime files generated at the same time:

- pytransform.py
- pytransform.key
- _pytransform.so (.dll or .dylib)
- license.lic

2. Create a new path *pytransform* in the Python system library, it would be *lib/site-packages* (on Windows) or *lib/pythonX.Y/site-packages* (on Linux)
3. Copy 4 runtime files to this path, rename *pytransform.py* as *__init__.py*
4. Edit *lib/site.py* (on Windows) or *lib/pythonX.Y/site.py* (on Linux), insert *Bootstrap Code* before the line *if __name__ == '__main__'*:

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

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

After that *python* could run the obfuscated scripts directly, becausee the module *site* is automatically imported during Python initialization.

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

12.5 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 foo.py
```

From PyArmor 5.2, *Restrict Mode* is default setting. It could be disabled by this way if required:

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

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

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

12.6 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 `check_ntp_time.py`:

```
# Uncomment the next 2 lines for debug as the script isn't obfuscated,
# otherwise runtime module "pytransform" isn't available in development
# from pytransform import pyarmor_init
# pyarmor_init()

from ntplib import NTPClient
from time import mktime, strptime
import sys

def get_license_data():
    from ctypes import py_object, PYFUNCTYPE
    from pytransform import _pytransform
    prototype = PYFUNCTYPE(py_object)
    dlfunc = prototype(('get_registration_code', _pytransform))
    rcode = dlfunc().decode()
    index = rcode.find(';', rcode.find('*CODE:'))
    return rcode[index+1:]

def check_expired():
    NTP_SERVER = 'europe.pool.ntp.org'
    EXPIRED_DATE = get_license_data()
    c = NTPClient()
    response = c.request(NTP_SERVER, version=3)
    if response.tx_time > mktime(strptime(EXPIRED_DATE, '%Y%m%d')):
        sys.exit(1)
```

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

```
...

# {PyArmor Plugins}

...

def main():
    # PyArmor Plugin: check_expired()

if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO)
    main()
```

Now obfuscate entry script:

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

By this way, the content of `check_ntp_time.py` will be insert after the first comment:

```
# {PyArmor Plugins}

... the conent of check_ntp_time.py
```

At the same time, the prefix of second comment will be stripped:

```
def main():
    # PyArmor Plugin: check_expired()
    check_expired()
```

So the plugin takes effect.

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
```

Or set environment variable *PYARMOR_PLUGIN*. For example:

```
export PYARMOR_PLUGIN=/usr/share/pyarmor/plugins
pyarmor obfuscate --plugin check_ntp_time foo.py
```

Finally generate one license file for this obfuscated script:

```
pyarmor licenses -x 20190501 MYPRODUCT-0001
cp licenses/MYPRODUCT-0001/license.lic dist/
```

Note: It's better to insert the content of *ntplib.py* into the plugin so that *NTPClient* needn't be imported out of obfuscated scripts.

12.7 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
dist/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 \
           -e " --onefile --icon logo.ico --runtime-hook copy_license.py" foo.py
   ```

Option *–without-license* tells *pyamor* 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 *pyarmor 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 tom
   cp license/tom/license.lic dist/

   dist/foo.exe
   ```

12.8 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*

CHAPTER 13

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
```

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

13.1 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
--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
--no-cross-protection	Do not insert protection code to entry script
--plugin NAME	Insert extra code to entry script
--platform NAME	Distribute obfuscated scripts to other platform
--advanced	Enable advanced mode
--restrict <0,1,2,3,4>	Set restrict mode

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

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 generate default `license.lic` for obfuscated scripts and make all the other *Runtime Files* in the *dist* path.

Finally insert *Bootstrap Code* into entry script.

The entry script is only the first script if there are more than one script in command line.

Option *–plugin* is used to extend license type of obfuscated scripts, it will insert the content of plugin into entry script. The corresponding filename of plugin is *NAME.py*. *Name* may be absolute path if it's not in the current path, or specify plugin path by environment variable *PYARMOR_PLUGIN*.

About the usage of plugin, refer to *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.

Option *–restrict* is used to set restrict mode, *Restrict Mode*

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, 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
```

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

```
pyarmor download --list
pyarmor download linux_x86_64

pyarmor obfuscate --platform linux_x86_64 foo.py
```

- Obfuscate the scripts in advanced mode:

```
pyarmor obfuscate --advanced 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 .
```

13.2 licenses

Generate new licenses for obfuscated scripts.

SYNOPSIS:

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

OPTIONS

-O, --output OUTPUT Output path

-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

DESCRIPTION

In order to run obfuscated scripts, it's necessarey to hava a **:flile:'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/
```

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/
```

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'])
```

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

13.3 pack

Obfuscate the scripts and pack them into one bundle.

SYNOPSIS:

```
pyarmor pack <options> SCRIPT
```

OPTIONS

-t, --type TYPE cx_Freeze, py2exe, py2app, PyInstaller(default).

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

-e, --options OPTIONS Extra options to run external tool

-x, --xoptions OPTIONS Extra options to obfuscate scripts

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

--clean Remove last build path before packing

--debug Do not remove build files after packing

DESCRIPTION

PyArmor first packes the script by calling the third-party tool such as PyInstaller, gets the dependencies and other required files.

Then obfuscates all the *.py* files in the same path of entry script.

Next replace the original scripts with the obfuscated ones.

Finally pack all of them into one bundle.

Option *–options* could pass any extra options to external tool. *PyInstaller* is called by this way:

```
pyinstaller --distpath DIST -y EXTRA_OPTIONS SCRIPT
```

EXTRA_OPTIONS is replaced with this option.

Option *–xoptions* could pass any extra options to obfuscate scripts. By default, *pack* will obfuscate scripts like this:

```
pyarmor obfuscate -r --output DIST EXTRA_OPTIONS SCRIPT
```

EXTRA_OPTIONS is replaced with this option.

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

Important: Do not pack the obfuscated scripts, but plain scripts directly.

EXAMPLES

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

  ```
  pyarmor pack 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
```

- If the application name is changed by passing option *-n* of *PyInstaller*, the option *-s* must be specified at the same time. For example:

```
pyarmor pack -e " -n my_app" -s "my_app.spec" foo.py
```

13.4 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*

13.5 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 `.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
  ```

13.6 config

Update project settings.

SYNOPSIS:

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

OPTIONS

--name NAME Project name

--title TITLE Project title

--src SRC Project src

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

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

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

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

--advanced-mode <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

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.

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
  ```

- Clear all plugins:

  ```
  pyarmor config --plugin clear
  ```

13.7 build

Build project, obfuscate all scripts in the project.

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

DESCRIPTION

Run this command in project path:

```
pyarmor build
```

Or specify the project path at the end:

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

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 download --list
  pyarmor download linux_x86_64

  pyarmor build -B --platform linux_x86_64
  ```

13.8 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
```

13.9 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
```

13.10 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> 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
  ```

13.11 register

Make registration keyfile effect, or show registration information.

SYNOPSIS:

```
pyarmor register [KEYFILE]
```

DESCRIPTION

A registration keyfile will be sent to you by email as attachments after purchasing PyArmor. This command is used to register the keyfile to take it effects:

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

Show registration information:

```
pyarmor register
```

13.12 download

List and download platform-dependent dynamic libraries.

SYNOPSIS:

```
pyarmor download <options> PLAT-ID
```

OPTIONS:

--list PATTERN List available dynamic libraries in different platforms

-O, --output NAME Save downloaded file to another path

DESCRIPTION

In some machines maybe PyArmor could not recognize the platform and raise error. For example:

```
ERROR: Unsupport platform linux32/armv7l
```

In this case, check all the available prebuilt libraries:

```
pyarmor download --list
```

And download *armv7* from this list:

```
pyarmor download --output linux32/armv7l armv7
```

Filter could be applied to list the platforms, for example:

```
pyarmor download --list linux32
```

CHAPTER 14

Examples

Here are some examples.

14.1 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 --exclude tests" foo.py
```

CHAPTER 15

When Things Go Wrong

Turn on debugging output to get more error information:

```
python -d pyarmor.py ...
PYTHONDEBUG=y pyarmor ...
```

15.1 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

15.2 Could not find *_pytransform*

Generally, the dynamic library *_pytransform* is 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

15.3 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.

If obfuscate scripts by command *pyarmor obfuscate*, *Global Capsule* is used implicitly. If obfuscate scripts by command *pyarmor build*, the project capsule is used.

When generating new licenses for obfuscated scripts, if run command *pyarmor licenses* in project path, the project capsule is used implicitly, otherwise *Global Capsule*.

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.

15.4 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.

15.5 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. Check whether the capsule is generated based on current license of PyArmor. Try to move global capsule *~/.pyarmor_capsule.zip* to any other path, then obfuscate scripts again.
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.

15.6 _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.

15.7 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*

15.8 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*

15.9 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.

15.10 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.

15.11 Check license failed: Invalid input packet.

If print this error as running the obfuscated scripts, check if there is any of *license.lic* or *pytransform.key* in the current path. To be 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.

CHAPTER 16

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 35728 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.
- 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.

- A enterprise license for business users. The user purchases one license to use the software for one product serials of an organization.

  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.

16.1 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 by 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 17

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 package, refer to *Prebuilt Libraries Distributed with PyArmor*.

Some of them are not, refer to *All The Others Prebuilt Libraries For PyArmor*. In these platforms, in order to run pyarmor, first download the corresponding prebuilt dynamic library, then put it in the installed path of PyArmor package.

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

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

OS	Arch	Download	Description
Windows	i686	_pytransform.dll	Cross compile by i686-pc-mingw32-gcc in cygwin
Windows	AMD64	_pytransform.dll	Cross compile by x86_64-w64-mingw32-gcc in cygwin
Linux	i686	_pytransform.so	Built by GCC
Linux	x86_64	_pytransform.so	Built by GCC
MacOSX	x86_64, intel	_pytransform.dylib	Built by CLang with MacOSX10.11

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

OS	Arch	Download	Description
Windows	x86	_pytransform.dll	Built by VS2015
Windows	x64	_pytransform.dll	Built by VS2015
Linux	armv5	_pytransform.so	32-bit Armv5 (arm926ej-s)
Linux	armv7	_pytransform.so	32-bit Armv7 Cortex-A, hard-float, little-endian
Linux	aarch32	_pytransform.so	32-bit Armv8 Cortex-A, hard-float, little-endian
Linux	aarch64	_pytransform.so	64-bit Armv8 Cortex-A, little-endian
Linux	ppc64le	_pytransform.so	For POWER8
iOS	arm64	_pytransform.dylib	Built by CLang with iPhoneOS9.3.sdk
FreeBSD	x86_64	_pytransform.so	Not support harddisk serial number
Alpine Linux	x86_64	_pytransform.so	Built with musl-1.1.21 for Docker
Alpine Linux	arm	_pytransform.so	Built with musl-1.1.21, 32-bit Armv5T, hard-float, little-endian
Inel Quark	i586	_pytransform.so	Cross compile by i586-poky-linux
Android	aarch64	_pytransform.so	Build by android-ndk-r20/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android21-clang

CHAPTER 18

Change Logs

18.1 5.6.4

- Fix segmentation fault issue for Python 3.8

18.2 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.

18.3 5.6.2

- Fix *pyarmor-webui* start issue in some cases: can't import name '_project'

18.4 5.6.1

- The command *download* will check the version of dynamic library to be sure it works with the current PyArmor.

18.5 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 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 just obfuscate the scripts, and do not issue 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.

18.6 5.5.7

- Fix webui bug: raise "name 'output' is not defined" as running *packer*

18.7 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

18.8 5.5.5

- Win32 issue: the obfuscated scripts will print extra message.

18.9 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

18.10 5.5.3

- Fix issue: it will raise error "Invalid input packet" to import 2 independent obfuscated packages in 64-bit Windows.

18.11 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*

18.12 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.

18.13 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.

18.14 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.

18.15 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

18.16 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

18.17 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

18.18 5.4.2

- Option *–exclude* can use multiple times in command *obfuscate*
- Exclude build path automatically in command *pack*

18.19 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

18.20 5.4.0

- Do not obfuscate lambda functions by default
- Fix issue: local variable *platname* referenced before assignment

18.21 5.3.13

- Add option *–url* for command *download*

18.22 5.3.12

- Add integrity checks for the downloaded binaries (#85)

18.23 5.3.11

- Fix issue: get wrong harddisk's serial number for some special cases in Windows

18.24 5.3.10

- Query harddisk's serial number without administrator in Windows

18.25 5.3.9

- Remove the leading and trailing whitespace of harddisk's serial number

18.26 5.3.8

- Fix non-ascii path issue in Windows

18.27 5.3.7

- Fix bug: the bootstrap code isn't inserted correctly if the path of entry script is absolute path.

18.28 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

18.29 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.

18.30 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.

18.31 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*

18.32 5.3.2

- Fix bug: if there is only one comment line in the script it will raise IndexError as obfuscating this script.

18.33 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

18.34 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.

18.35 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*

18.36 5.2.8

- Fix issue: *pyarmor -v* complains *not enough arguments for format string*

18.37 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*.

18.38 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

18.39 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

18.40 5.2.4

- Fix arm64 issue: verify rsa key failed when running the obufscated scripts(#63)
- Support ios (arm64) and ppc64le for linux

18.41 5.2.3

- Refine error message when checking license failed
- Fix issue: protection code raises ImportError in the package file __*init.py*__

18.42 5.2.2

- Improve the security of dynamic library.

18.43 5.2.1

- Fix issue: in restrict mode the bootstrap code in *__init__.py* will raise exception.
- Add option *–cross-protection* in command *obfuscate*

18.44 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*

18.45 5.1.2

- Improve the security of PyArmor self

18.46 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.

18.47 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)

18.48 5.0.5

- Refine *protect_pytransform* to improve security, refer to https://pyarmor.readthedocs.io/en/latest/security.html

18.49 5.0.4

- Fix *get_expired_days* issue, remove decorator *dllmethod*
- Refine output message of *pyarmor -v*

18.50 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

18.51 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

18.52 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'*

18.53 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

18.54 4.6.2

- Add option *–options* for command *pack*
- For Python 3, there is no new line in the output when *pack* command fails

18.55 4.6.1

- Fix license issue in 64-bit embedded platform

18.56 4.6.0

- Fix crash issue for special code object in Python 3.6

18.57 4.5.5

- Fix stack overflow issue

18.58 4.5.4

- Refine platform name to search dynamic library *_pytransform*

18.59 4.5.3

- Print the exact message when checking license failed to run obfuscated scripts.

18.60 4.5.2

- Add documentation https://pyarmor.readthedocs.io/en/latest/
- Exclude *dist*, *build* folder when executing *pyarmor obfuscate –recursive*

18.61 4.5.1

- Fix #41: can not find dynamic library *_pytransform*

18.62 4.5.0

- Add anti-debug code for dynamic library *_pytransform*

18.63 4.4.2

- Change default capsule to user home other than the source path of *pyarmor*

18.64 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

18.65 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*

18.66 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

18.67 4.3.4

- Fix start pyarmor issue for *pip install* in Python 2

18.68 4.3.3

- Fix issue: missing file in wheel

18.69 4.3.2

- Fix *pip* install issue in MacOS
- Refine sample scripts to make workaround for py2exe/cx_Freeze simple

18.70 4.3.1

- Fix typos in examples
- Fix bugs in sample scripts

18.71 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.

18.72 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)

18.73 4.2.2

- Fix webui import error.

18.74 4.2.1

- Add option *–recursive* for command *obfuscate*

18.75 4.1.4

- Rewrite project long description.

18.76 4.1.3

- Fix Python3 issue for *get_license_info*

18.77 4.1.2

- Add function *get_license_info* in *pytransform.py* to show license information

18.78 4.1.1

- Fix import *main* from *pyarmor* issue

18.79 4.0.3

- Add command *capsule*
- Find default capsule in the current path other than *–src* in command *obfuscate*
- Fix pip install issue #30

18.80 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*

18.81 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

18.82 3.9.9

- Fix segmentation fault issues for *asyncio*, *typing* modules

18.83 3.9.8

- Add documentation for examples (examples/README.md)

18.84 3.9.7

- Fix windows 10 issue: access violation reading 0x000001ED00000000

18.85 3.9.6

- Fix the generated license bind to fixed machine in webui is not correct
- Fix extra output path issue in webui

18.86 3.9.5

- Show registration code when printing version information

18.87 3.9.4

- Rewrite long description of package in pypi

18.88 3.9.3

- Fix issue: __*file*__ is not really path in main code of module when import obfuscated module

18.89 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*

18.90 3.9.1

- Add option *–type* for command *init*
- Refine document *user-guide.md* and *mechanism.md*

18.91 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

18.92 3.8.10

- Fix shared code object issue in *__wraparmor__*

18.93 3.8.9

- Clear frame as long as *tb* is not *Py_None* when call *__wraparmor__*
- Generator will not be obfucated in *__wraparmor__*

18.94 3.8.8

- Fix bug: the *frame.f_locals* still can be accessed in callback function

18.95 3.8.7

- The *frame.f_locals* of *wrapper* and wrapped function will return an empty dictionary once *__wraparmor__* is called.

18.96 3.8.6

- The *frame.f_locals* of *wrapper* and wrapped function return an empty dictionary, all the other frames still return original value.

18.97 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.

18.98 3.8.4

- Do not touch *frame.f_locals* when raise exception, let decorator *wraparmor* to control everything.

18.99 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*

18.100 3.8.2

- Change module filename to *<frozen modname>* in traceback, set attribute *__file__* to real filename when running obfuscated scripts.

18.101 3.8.1

- Try to access original func_code out of decorator *wraparmor* is forbidden.

18.102 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.

18.103 3.7.5

- Remove dotted name from filename in traceback, if it's not a package.

18.104 3.7.4

- Strip *__init__* from filename in traceback, replace it with package name.

18.105 3.7.3

- Remove brackets from filename in traceback, and add dotted prefix.

18.106 3.7.2

- Change filename in traceback to <*frozen [modname]*>, other than original filename

18.107 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*

18.108 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**

18.109 3.6.2

- Fix issue #8 (Linux): option –manifest broken in shell script

18.110 3.6.1

- Add option "Restrict Mode" in web ui
- Document restrict mode in details (user-guide.md)

18.111 3.6.0

- Introduce restrict mode to avoid obfuscated scripts observed from no obfuscated scripts
- Add option –disable-restrict-mode for command "config"

18.112 3.5.1

- Support pip install pyarmor

18.113 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

18.114 3.4.3

- Fix issue: raise exception when project entry isn't obfuscated

18.115 3.4.2

- Add webui to manage project

18.116 3.4.1

- Fix README.rst format error.
- Add title attribute to project
- Print new command help when option is -h, –help

18.117 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.

18.118 3.3.1

- Remove unused files in distribute package

18.119 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

18.120 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

18.121 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

18.122 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

18.123 3.1.6

- Fix serial number of harddisk doesn't work in mac osx.

18.124 3.1.5

- Support MACOS

18.125 3.1.4

- Fix issue: load _pytransfrom failed in linux x86_64 by subprocess.Popen
- Fix typo in error messge when load _pytransfrom failed.

18.126 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”

18.127 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

18.128 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

18.129 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

18.130 2.6.1

- Fix encrypted scripts don't work in multi-thread framework (Django).

18.131 2.5.5

- Add option '-i' for command 'encrypt' so that the encrypted scripts will be saved in the original path.

18.132 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.

18.133 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

18.134 2.5.2

- License file can mix expire-date with fix file or fix key.
- Fix log error: not enough arguments for format string

18.135 2.5.1

- License file can bind to ssh private key file or any other fixed file.

18.136 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.

18.137 2.3.4

- The trial license will never be expired (But in trial version, the key used to encrypt scripts is fixed).

18.138 2.3.3

- Refine the document

18.139 2.3.2

- Fix error data in examples of wizard

18.140 2.3.1

- Implement Run function in the GUI wizard
- Make license works in trial version

18.141 2.2.1

- Add a GUI wizard
- Add examples to show how to use pyarmor

18.142 2.1.2

- Fix syntax-error when run/import encrypted scripts in linux x86_64

18.143 2.1.1

- Support armv6

18.144 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

18.145 1.7.7

- Add option '–expired-date' for command 'license'
- Fix undefined 'tfm_desc' for arm-linux
- Enhance security level of scripts

18.146 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.

18.147 1.7.5

- Add missing extensions for linux x86_64.

18.148 1.7.4

- Add command "licene" to generate more "license.lic" by project capsule.

18.149 1.7.3

- Add information for using registration code

18.150 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"

18.151 1.7.1

- Encrypt pyshield.lic when distributing source code.

18.152 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 19

Indices and tables

- genindex
- modindex
- search

Index