obfuscator Documentation

Release 1.1.5

Timothy McFadden

January 02, 2015

Contents

Contents:

CHAPTER 1

Introduction

Obfuscator is a Python package used to obfuscate a set of data (e.g. bytes). It provides **no** encryption! It's strictly a "security through obscurity" tool, with limited usefulness. You have been warned!

The project is hosted on GitHub at https://github.com/mtik00/obfuscator

CHAPTER 2

Install

Download the latest release tarball and install with *pip install <package>*.

CHAPTER 3

Usage

See the unit tests for more in-depth examples. Here are the basics:

```
import obfuscator
original_bytes = map(ord, "testing")
_key, obfuscated_bytes = obfuscator.obfuscate_xor(original_bytes, key=0x66)
deobfuscated_bytes = obfuscator.deobfuscate_xor(key=0x66, data=obfuscated_bytes)
assert original_bytes == deobfuscated_bytes
```

CHAPTER 4

Auto Generated API Documentation

Contents:

4.1 obfuscator.file

This module contains a file-level interface for the XOR obfuscation methods.

class `obfuscator.file.`**`ObfuscatedFile`**(*filename*)

Parameters **filename** (*str*) – The path of the file you want to read/write.

This class represents an obfuscated data file. You can use this to store some-what sensitive information inside a file. The documentation for some of the functions is purposely light.

Example #1 - Storing a string:

```
>>> from obfuscator.file import ObfuscatedFile
>>> of = ObfuscatedFile('data.bin')
>>> bytes = map(ord, "my string")
>>> of.write(bytes)  # data.bin is 32 bytes long
86  # The random key used for encoding; stored in the file
>>> read_bytes = of.read()
>>> read_bytes
[109L, 121L, 32L, 115L, 116L, 114L, 105L, 110L, 103L]
>>> ''.join(map(chr, read_bytes))
'my string'
```

Example #2 - Storing a string with known key:

```
>>> from obfuscator.file import ObfuscatedFile
>>> of = ObfuscatedFile('data.bin')
>>> bytes = map(ord, "my string")
>>> my_key = 0xCF
>>> of.write(bytes, my_key)  # data.bin is 32 bytes long
207
>>> read_bytes = of.read()  # Notice how we didn't use a key
>>> ''.join(map(chr, read_bytes))
',8a253(/&'
>>> # The string is wrong because the key is not stored in the file
>>> read_bytes = of.read(key=my_key)
>>> ''.join(map(chr, read_bytes))
'my string'
>>>
```

read(*key=None*)

This function reads a file written by *write*, and returns the deobfuscated data.

Parameters **key** (*int*) – The key used during *write()*. NOTE: If you passed in a key during *write()*, you *must* use the same key here; the key will not be stored in the file. If you let the algorithm choose the key, it is stored in the file, and will be used during *read()*.

write(*data*, *key=None*, *minimum_length=32*)

Write the data to a file.

Parameters

- **data** (*iterable*) – The data you want to encode; the length of data must be less than 0xFF (header size limitation)
- **key** (*int*) – The key used during encoding
- **minimum_length** (*int*) – The minimum number of bytes to write. If the encoding operation produces fewer bytes that this, random bytes are appended to the end of the result so len(bytes) == minimum_length.

4.2 obfuscator

This module contains a simple mechanism for obfuscating a set of data. Consider this "security through obscurity". This module contains no encryption mechanisms!

Example:

```
>>> from obfuscator import obfuscate_xor, deobfuscate_xor
>>> data = [1, 2, 3, 4]
>>> key, odata = obfuscate_xor(data)
>>> key, odata
(162, [163, 160, 161, 166, 166, 154, 181, 60, 131, 24, 88, 35, 137, 240, 216, 161, 247, 218, 19, 116,
>>> assert data == deobfuscate_xor(key, odata)[:len(data)]
>>>
>>> key, odata = obfuscate_xor(data, minimum_length=0)
>>> assert data == deobfuscate_xor(key, odata)
>>>
```

obfuscator.**deobfuscate**(*key*, *data*, *encoder=1*)

This function obfuscates the data using the default operation.

obfuscator.**deobfuscate_offset**(*key*, *data*)

This function deobfuscates the data using an offset operation.

The formula used is: [x - key for x in data]

Parameters

- **key** (*int*) – The key used for the offset operation.
- **data** (*iterable*) – The data you want to obfuscate

obfuscator.**deobfuscate_rot13**(*key*, *data*)

This function performs a ROT13 decode of the data. *data* needs to be an iterable that contains a representation of str types. This can be either a string of type *str*, or a list of bytes from something like *ord*.

Parameters

- **key** (*int*) – This value is ignored; it only exists to conform to the other methods.
- **data** (*iterable*) – The data you want to deobfuscate

`obfuscator.deobfuscate_xor`(*key*, *data*)

This function deobfuscates the data using an byte-wise XOR operation.

The formula used is: [x ^ key for x in data]

Parameters

- **key** (*int*) – The key used for the XOR operation.
- **data** (*iterable*) – The data you want to obfuscate

`obfuscator.obfuscate`(*data*, *key=None*, *minimum_length=32*, *encoder=1*)

This function obfuscates the data using the default operation.

`obfuscator.obfuscate_offset`(*data*, *key=None*, *minimum_length=32*)

This function obfuscates the data using an offset operation.

The formula used is: [x + key for x in data]

Parameters

- **data** (*iterable*) – The data you want to obfuscate
- **key** (*int*) – The value used for the offset operation. By default, the value will be a random integer between 40 and 127.
- **minimum_length** (*int*) – The minimum number of bytes to return. If the encoding operation produces fewer bytes that this, random bytes are appended to the end of the result so len(bytes) == minimum_length.

`obfuscator.obfuscate_rot13`(*data*, *key=None*, *minimum_length=32*)

This function performs a ROT13 encode on the data. *data* needs to be an iterable that contains a representation of str types. This can be either a string of type *str*, or a list of bytes from something like *ord*.

Parameters

- **data** (*iterable*) – The data you want to obfuscate
- **key** (*int*) – This value is ignored; it only exists to conform to the other methods.
- **minimum_length** (*int*) – The minimum number of bytes to return. If the encoding operation produces fewer bytes that this, random bytes are appended to the end of the result so len(bytes) == minimum_length.

`obfuscator.obfuscate_xor`(*data*, *key=None*, *minimum_length=32*)

This function obfuscates the data using an byte-wise XOR operation.

The formula used is: [x ^ key for x in data]

Parameters

- **data** (*iterable*) – The data you want to obfuscate
- **key** (*int*) – The key used for the XOR operation. By default, the key will be a random integer between 1 and 255.
- **minimum_length** (*int*) – The minimum number of bytes to return. If the encoding operation produces fewer bytes that this, random bytes are appended to the end of the result so len(bytes) == minimum_length.

`obfuscator.rot13`(*data*, *minimum_length=32*)

This function performs a ROT13 encode/decode on the data. *data* needs to be an iterable that contains a representation of str types. This can be either a string of type *str*, or a list of bytes from something like *ord*.

Parameters

- **data** (*iterable*) – The data you want to obfuscate
- **minimum_length** (*int*) – The minimum number of bytes to return. If the encoding operation produces fewer bytes that this, random bytes are appended to the end of the result so len(bytes) == minimum_length.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

O

D

O

R

W