wheezy.security documentation

Release latest

Andriy Kornatskyy

Apr 17, 2021

Contents

CHAPTER 1

Introduction

wheezy.security is a python package written in pure Python code. It is a lightweight security library that provides integration with:

- pycrypto - The Python Cryptography Toolkit.
- pycryptodome - PyCryptodome is a fork of PyCrypto. It brings several enhancements.
- pycryptodomex - PyCryptodomex is a library independent of the PyCrypto.
- cryptography - cryptography is a package which provides cryptographic recipes and primitives to Python developers.

It is optimized for performance, well tested and documented.

Resources:

- source code and issues tracker are available on github
- documentation

CHAPTER 2

Contents

2.1 Getting Started

2.1.1 Install

wheezy.security requires python version 3.6+. It is independent of operating system. You can install it from pypi site

```
$ pip install wheezy.security
```

2.2 Examples

We start with a simple example. Before we proceed let's setup a virtualenv environment:

```
$ pip install wheezy.security[pycryptodome]
```

2.2.1 Protecting Information

Let's assume we would like to protect some sensitive information, e.g. user id. We can encrypt it, add a hash to prove validity and finally say that this value is valid for 20 minutes only:

```
from wheezy.security.crypto import Ticket

ticket = Ticket(max_age=1200, salt='p5sArbHFZvxgeEJFrM9h')
```

Once you have ticket you can encode any string:

```
protected_value = ticket.encode('hello')
```

Decode `protected_value` this way:

```
value = ticket.decode(protected_value)
```

2.2.2 User Principal

Ticket can be used to protect user principal over network (e.g. in http cookie):

```
from wheezy.security import Principal

principal = Principal(
        id='125134788',
        roles=['user'],
        alias='John Smith')
secure_value = ticket.encode(principal.dump())
```

Server side now restores this information:

```
from wheezy.security import ANONYMOUS
from wheezy.security import Principal

principal_dump = ticket.decode(secure_value)
if principal_dump:
    principal = Principal.load(principal_dump)
else:
    principal = ANONYMOUS
```

2.3 User Guide

The objective of security is protection of information from theft or corruption, while allowing the information to remain accessible to its intended users.

2.3.1 Ticket

Ticket is a short packet of bytes generated by a network server for a client, which can be delivered to itself as a means of authentication or proof of authorization, and cannot easily be forged.

Ticket has the following characteristics:

- It is valid for certain period of time, in particular it has an explicitly set expiration time.
- Its value is signed to prove its authenticity.
- It is encrypted to protect sensitive information.
- It has noise to harden forgery.

Ticket can be instantiated by passing the following arguments:

- max_age - period of time (in seconds) this Ticket is considered valid.
- salt - a random sequence that hardens against ticket forgery. It is prepended to the validation key and the encryption key.
- digestmod - hash algorithm used with HMAC (Hash-based Message Authentication Code) to sign ticket. Defaults to SHA1.
- cypher - cryptography algorithm. Defaults to AES128.

- `options` - a dictionary that holds the following configuration values: `CRYPTO_VALIDATION_KEY` (used by signature) and `CRYPTO_ENCRYPTION_KEY` (used by encryption).

Validation and Encryption Keys

Keys used for validation and encryption are ensured to be at least of 320 bits length. The *`ensure_strong_key()`* function appends HMAC signature to the key.

If the cryptography library is not available you will see a warning message:

```
Ticket: cypher not available
```

Although Ticket continues to function even cryptography library is not installed it strongly recommended to use cryptography in a production environment.

Thread Safety

Ticket does not alter it state once initialized. It is guaranteed to be thread safe.

Typical Use Case

Here is typical use case when all possible configuration attributes are used:

```
from wheezy.security.crypto.comp import aes192
from wheezy.security.crypto.comp import sha1
from wheezy.security.crypto import Ticket

options = {
    'CRYPTO_VALIDATION_KEY': 'LkLlYR5WbTk54kaIgJOp',
    'CRYPTO_ENCRYPTION_KEY': 'rH64daeXBZdgrR7WNawf'
}

ticket = Ticket(
        max_age=1200,
        salt='CzQnV0KazDKElBYiIC2w',
        digestmod=sha1,
        cypher=aes192,
        options=options)
```

The `ticket` instance can be shared application wide. The `encode` / `decode` methods are used in the following way:

```
protected_value = ticket.encode('hello')

assert 'hello' == ticket.decode(protected_value)
```

In case the validity of a ticket cannot be confirmed, the `decode` method returns `None`.

Extensibility

Ticket `cypher` can be any callable that satisfies the following contract:

- Initialization is called with encryption key. Returned object must be a factory for the actual algorithm instance.
- Algorithm factory must return new algorithm via simple callable with no arguments.

- Algorithm implementation must support two methods: `encrypt(value)` and `decrypt(value)`.

2.3.2 Principal

Principal is a container of user specific security information. It includes the following attributes:

- `id` - user identity, e.g. number *755345*, UUID *f102a87b-ee36-4a2e-97de-8f803f470867* or whatever else is valid to look up a user quickly in your application.
- `roles` - a list of authorized user roles, e.g. *user*, *manager*, etc.
- `alias` - a user friendly name, display name, etc. This can be something like *John Smith*, etc.
- `extra` - any string you would like to hold in security context.

Here is a sample how to instantiate new Principal:

```
principal = Principal(
        id='125134788',
        roles=['user'],
        alias='John Smith')
```

Principal supports the following methods:

- `dump` - converts instance to a string.
- `load` - reverse operation to `dump`.

You can use `Ticket` to securely pass `Principal` across network boundaries. Combining them both you can introduce an authentication/authorization cookie to your application.

2.3.3 Authorization

Authorization specifies access rights to resources and provides access control in particular to your application.

You are able to request authorization by decorating your method with *authorized()*. Here is a typical use case:

```
from wheezy.security import authorized

class MyBusinessLogic(object):

     principal = None

     @authorized
     def cancel_transfer(self, id):
         return True

     @authorized(roles=('operator',))
     def approve_transfer(self):
         return True
```

Note that the *authorized()* decorator requires the object to supply a `principal` attribute of type *Principal*.

If a caller is not authorized to perform a requested operation, a *SecurityError* exception is raised. See *authorized()* for more details.

2.4 Modules

2.4.1 wheezy.security

wheezy.security.**authorized**(*wrapped=None*, *roles=None*)

Demand the user accessing protected resource is authenticated and optionally in one of allowed `roles`.

Requires wrapped object to provide attribute principal.

`roles` - a list of authorized roles.

Here is an example:

```
from wheezy.security.principal import Principal

class Context(object):
    principal = None

    @authorized
    def op_a(self):
        return True

    @authorized(roles=('operator',))
    def op_b(self):
        return True
```

exception wheezy.security.**SecurityError**(*message*)

Raised when a security error occurs. It is subclass of `RuntimeError`.

class wheezy.security.**Principal**(*id=”*, *roles=()*, *alias=”*, *extra=”*)

Container of user specific security information

dump()

Dump principal object.

classmethod load(*s*)

Load principal object from string.

2.4.2 wheezy.security.authorization

`authorization` module.

wheezy.security.authorization.**authorized**(*wrapped=None*, *roles=None*)

Demand the user accessing protected resource is authenticated and optionally in one of allowed `roles`.

Requires wrapped object to provide attribute principal.

`roles` - a list of authorized roles.

Here is an example:

```
from wheezy.security.principal import Principal

class Context(object):
    principal = None

    @authorized
    def op_a(self):
```

(continues on next page)

(continued from previous page)

```
        return True

    @authorized(roles=('operator',))
    def op_b(self):
        return True
```

2.4.3 wheezy.security.errors

`errors` module.

exception `wheezy.security.errors.`**SecurityError**(*message*)

Raised when a security error occurs. It is subclass of `RuntimeError`.

2.4.4 wheezy.security.principal

`principal` module.

class `wheezy.security.principal.`**Principal**(*id=”*, *roles=()*, *alias=”*, *extra=”*)

Container of user specific security information

dump()

Dump principal object.

classmethod load(*s*)

Load principal object from string.

2.4.5 wheezy.security.crypto

`crypto` package.

class `wheezy.security.crypto.`**Ticket**(*max_age=900*, *salt=”*, *digestmod=None*, *cypher=None*, *options=None*)

Protects sensitive information (e.g. user id).

Default policy applies verification and encryption. Verification is provided by `hmac` initialized with `sha1` digestmod. Encryption is provided if available, by default it attempts to use AES cypher.

decode(*value*, *encoding=’UTF-8’*)

Decode `value` according to ticket policy.

encode(*value*, *encoding=’UTF-8’*)

Encode `value` according to ticket policy.

sign(*value*)

Compute hmac digest.

2.4.6 wheezy.security.crypto.ticket

`crypto` module.

class `wheezy.security.crypto.ticket.`**Ticket**(*max_age=900*, *salt=”*, *digestmod=None*, *cypher=None*, *options=None*)

Protects sensitive information (e.g. user id).

Default policy applies verification and encryption. Verification is provided by `hmac` initialized with `sha1` digestmod. Encryption is provided if available, by default it attempts to use AES cypher.

decode(*value*, *encoding='UTF-8'*)
Decode `value` according to ticket policy.

encode(*value*, *encoding='UTF-8'*)
Encode `value` according to ticket policy.

sign(*value*)
Compute hmac digest.

`wheezy.security.crypto.ticket.`**ensure_strong_key**(*key*, *digestmod*)
Translates a given key to a computed strong key of length 3 * digestmode.digest_size suitable for encryption, e.g. with digestmod set to `sha1` returns 480 bit (60 bytes) key.

2.4.7 wheezy.security.crypto.padding

`padding` module.

see http://www.di-mgt.com.au/cryptopad.html

`wheezy.security.crypto.padding.`**pad**(*s*, *block_size*)
Pad with zeros except make the last byte equal to the number of padding bytes.

The convention with this method is usually always to add a padding string, even if the original plaintext was already an exact multiple of *block_size* bytes.

`s` - byte string.

`wheezy.security.crypto.padding.`**unpad**(*s*, *block_size*)
Strip right by the last byte number.

`s` - byte string.

Python Module Index

Index