pymunge Documentation

Release 0.1.3

nomadictype

Feb 18, 2018

Contents:

CHAPTER 1

API reference

1.1 Basic encoding and decoding

pymunge.**encode**(*payload=None*)

Create a MUNGE credential using the default context. Optionally, a payload (byte string) can be encapsulated as well.

If successful, returns the credential (a byte string), otherwise raises a *MungeError*.

pymunge.**decode**(*cred*)

Validate a MUNGE credential using the default context.

If successful, returns `(payload, uid, gid, ctx)`, where `payload` is the payload encapsulated in the credential, `uid`, `gid` are the UID/GID of the process that created the credential, and `ctx` is a *MungeContext* set to the one used to create the credential.

If unsuccessful, a *MungeError* is raised. For certain errors (i.e. *EMUNGE_CRED_EXPIRED*, *EMUNGE_CRED_REWOUND*, *EMUNGE_CRED_REPLAYED*), the `payload`, `uid` and `gid` can still be obtained via the `result` property of the raised *MungeError*. Note that the context cannot be obtained from the *MungeError*; if you need it, manually create a *MungeContext* and use its decode() method.

1.2 MUNGE contexts

class pymunge.**MungeContext**(*ctx=None*)

A MUNGE context. Encapsulates a collection of options used when creating a credential, or obtained from decoding a credential.

MungeContext() creates a new context with default attributes. As contexts are mutable, the context's attributes can subsequently be modified by assigning values to them.

If `ctx != None`, `MungeContext(ctx)` creates a copy of the context `ctx`. (For `ctx == None`, `MungeContext(ctx)` is equivalent to *MungeContext()*.) Modifying attributes in the copy does not affect the attributes of the original context.

A *MungeContext* should be closed when it is no longer used. The easiest way to do this is to use the *MungeContext* as a context manager for a 'with' statement, which automatically closes the context when the 'with' scope ends, e.g.:

```
>>> with MungeContext() as ctx:
>>>     do stuff with ctx
>>> # ctx is now closed
```

Typical *MungeContext* usage patterns:

- For encoding:

```
>>> with MungeContext() as ctx:
>>>     (set attributes of ctx, if needed)
>>>     cred = ctx.encode(payload)
```

- For decoding:

```
>>> with MungeContext() as ctx:
>>>     payload, uid, gid = ctx.decode(cred)
>>>     (check attributes of ctx, if needed)
```

close()
Close this context, releasing any resources associated with it. Once a context is closed, it cannot be reopened. It also cannot be used to encode or decode credentials, nor can its attributes (other than *closed*) be read or set (in each case, a *MungeError* is raised). Calling *close()* on an already closed context has no effect.

decode(*cred*)
Validate a MUNGE credential. The attributes of this context will be set to those used to encode the credential.

If successful, returns `(payload, uid, gid)`, where `payload` is the payload encapsulated in the credential, and `uid`, `gid` are the UID/GID of the process that created the credential. Otherwise a *MungeError* is raised. For certain errors (i.e. *EMUNGE_CRED_EXPIRED*, *EMUNGE_CRED_REWOUND*, *EMUNGE_CRED_REPLAYED*), the `payload`, `uid` and `gid` can still be obtained via the `result` property of the raised *MungeError*.

encode(*payload=None*)
Create a MUNGE credential using the options defined in this context. Optionally, a payload (byte string) can be encapsulated as well.

If successful, returns the credential (a byte string), otherwise raises a *MungeError*.

addr4
The IPv4 address of the host where the credential was encoded, in dotted-quad notation (e.g. '127.0.0.1'). This property cannot be explicitly set.

cipher_type
Symmetric cipher type (a *CipherType*).

closed
True if this context is closed, False otherwise. This property cannot be explicitly set, instead use the *close()* method to close the context.

decode_time
The time (in seconds since the epoch) at which the credential was decoded. This property cannot be explicitly set.

encode_time
The time (in seconds since the epoch) at which the credential was encoded. This property cannot be explicitly set.

gid_restriction
Numeric GID allowed to decode the credential. This value will be matched against the effective group ID of the process requesting the credential decode. Default is the special value *GID_ANY*, which means no GID restriction is set.

mac_type
Message authentication code type (a *MACType*).

realm
Security realm (a str). Not currently supported.

socket
Path of the local domain socket for connecting with munged, a str.

ttl
Time-to-live (in seconds). This value controls how long the credential is valid once it has been encoded.

When encoding a credential, two special values can be used:

- *TTL_DEFAULT*, which specifies the default according to the munged configuration. This is the default value of this property.
- *TTL_MAXIMUM*, which specifies the maximum allowed by the munged configuration.

uid_restriction
Numeric UID allowed to decode the credential. This value will be matched against the effective user ID of the process requesting the credential decode. Default is the special value *UID_ANY*, which means no UID restriction is set.

zip_type
Compression type (a *ZipType*).

1.3 Enumerations and constants

class pymunge.**CipherType**
Bases: enum.Enum

MUNGE symmetric cipher types

AES128 = 4
AES CBC with 128b-block/128b-key

AES256 = 5
AES CBC with 128b-block/256b-key

Blowfish = 2
Blowfish CBC with 64b-block/128b-key

CAST5 = 3
CAST5 CBC with 64b-block/128b-key

Default = 1
default cipher specified by daemon

Disabled = 0
encryption disabled

class pymunge.**MACType**
Bases: enum.Enum

MUNGE message authentication code types

Default = 1
default MAC specified by daemon

Disabled = 0
MAC disabled – invalid, btw

MD5 = 2
MD5 with 128b-digest

RIPEMD160 = 4
RIPEMD-160 with 160b-digest

SHA1 = 3
SHA-1 with 160b-digest

SHA256 = 5
SHA-256 with 256b-digest

SHA512 = 6
SHA-512 with 512b-digest

class pymunge.**ZipType**
Bases: enum.Enum

MUNGE compression types

Default = 1
default zip specified by daemon

Disabled = 0
compression disabled

bzlib = 2
bzip2 by Julian Seward

zlib = 3
zlib "deflate" by Gailly & Adler

pymunge.**TTL_MAXIMUM**
Use the maximum TTL allowed by the daemon.

pymunge.**TTL_DEFAULT**
Use the default TTL specified by the daemon.

pymunge.**UID_ANY**
Do not restrict decode to a specific UID.

pymunge.**GID_ANY**
Do not restrict decode to a specific GID.

1.4 Exceptions

class pymunge.**MungeError**(*code*, *message*, *result=None*)
Bases: Exception

Generic MUNGE exception. Generally raised when an underlying libmunge function returns an error code, or in a few cases when a pymunge wrapper detects an invalid argument.

MungeError instances have the following attributes:

- code: The error code (a *MungeErrorCode*, which is NOT an integer). To retrieve the raw error code as an integer, use code.value.
- message: The message string from libmunge. This is only the raw message without the exception type or the error code.
- result: Partial result, in most cases None. If a decode fails with one of certain errors (i.e. *EMUNGE_CRED_EXPIRED*, *EMUNGE_CRED_REWOUND*, *EMUNGE_CRED_REPLAYED*), result is a 3-tuple (payload, uid, gid) containing the results that would have been returned by the decode function or method.

class pymunge.**MungeErrorCode**

Bases: enum.Enum

MUNGE error codes.

EMUNGE_BAD_ARG = 2
Invalid argument

EMUNGE_BAD_CIPHER = 10
Bad credential cipher type

EMUNGE_BAD_CRED = 8
Bad credential format

EMUNGE_BAD_LENGTH = 3
Exceeded maximum message length

EMUNGE_BAD_MAC = 11
Bad credential message authentication code type

EMUNGE_BAD_REALM = 13
Bad credential security realm

EMUNGE_BAD_VERSION = 9
Bad credential version

EMUNGE_BAD_ZIP = 12
Bad credential compression type

EMUNGE_CRED_EXPIRED = 15
Credential expired

EMUNGE_CRED_INVALID = 14
Credential invalid

EMUNGE_CRED_REPLAYED = 17
Credential replayed

EMUNGE_CRED_REWOUND = 16
Credential created in the future

EMUNGE_CRED_UNAUTHORIZED = 18
Credential decode unauthorized

EMUNGE_NO_MEMORY = 5
Out of memory

EMUNGE_OVERFLOW = 4
Buffer overflow

EMUNGE_SNAFU = 1
Doh!

EMUNGE_SOCKET = 6
Munged communication error

EMUNGE_SUCCESS = 0
Whoohoo!

EMUNGE_TIMEOUT = 7
Munged timeout

1.5 Low-level API

1.5.1 pymunge.raw - low-level API

This module contains declarations of raw libmunge C functions and constants.

Importing this module causes the libmunge shared library to be loaded.

Note that most function prototypes differ slightly from their C counterparts, as follows:

- For all C functions that originally return an error code (*munge_err_t*), the corresponding Python wrapper instead checks the return value and raises a *MungeError* if the wrapped function returns anything other than *EMUNGE_SUCCESS*.
- Some functions originally return multiple values via pointer-based output arguments (e.g. uid and gid in *munge_decode*). The Python wrapper does not take these arguments and instead returns the multiple values as a tuple.

pymunge.raw.**uid_t**
The *uid_t* POSIX type. Specifies a numeric user ID.

alias of c_uint

pymunge.raw.**gid_t**
The *gid_t* POSIX type. Specifies a numeric group ID.

alias of c_uint

pymunge.raw.**time_t**
The *time_t* C type. Specifies a timestamp.

alias of c_long

pymunge.raw.**munge_ctx_t**
The *munge_ctx_t* C type, an opaque handle to a MUNGE context. The low-level version of *MungeContext*.

alias of c_void_p

pymunge.raw.**munge_err_t**
The *munge_err_t* C enumeration type. Specifies a MUNGE error code. The low-level version of *MungeErrorCode*.

alias of c_int

pymunge.raw.**munge_opt_t**
The *munge_opt_t* C enumeration type. Specifies a MUNGE context option.

alias of c_int

pymunge.raw.**munge_enum_t**
The *munge_enum_t* C enumeration type. Specifies a MUNGE enumeration.

alias of c_int

pymunge.raw.**munge_encode**(*ctx*, *buf*, *len*)
C prototype: `munge_err_t munge_encode(char **cred, munge_ctx_t ctx, const void *buf, int len);`

Note: when called from Python, returns `cred` instead of the *munge_err_t*.

Creates a credential contained in a base64 string. A payload specified by a buffer `buf` (a byte string) of length `len` can be encapsulated in as well. If the munge context `ctx` is None, the default context will be used. Returns the credential `cred` if the credential is successfully created; otherwise, raises a *MungeError* containing the error code and message. The error message may be more detailed if a `ctx` was specified.

pymunge.raw.**munge_decode**(*cred*, *ctx*)
C prototype: `munge_err_t munge_decode(const char *cred, munge_ctx_t ctx, void **buf, int *len, uid_t *uid, gid_t *gid);`

Note: when called from Python, returns `(payload, uid, gid)` instead of the *munge_err_t*, where `payload` is the contents of `buf` of length `len`. Example usage:

```
>>> payload, uid, gid = munge_decode(cred, ctx)
```

Validates the credential `cred`. If the munge context `ctx` is not None, it will be set to that used to encode the credential. If the credential is valid, returns the encapsulated payload byte string `payload` as well as the numeric UID `uid` and GID `gid` of the process that created the credential. If the credential is not valid, raises a *MungeError* containing the error code and message. The error message may be more detailed if a `ctx` was specified. For certain errors (ie, *EMUNGE_CRED_EXPIRED*, *EMUNGE_CRED_REWOUND*, *EMUNGE_CRED_REPLAYED*), the raised *MungeError* will contain the result `(payload, uid, gid)` which would have been returned if the credential were still valid.

pymunge.raw.**munge_strerror**(*e*)
C prototype: `const char * munge_strerror(munge_err_t e);`

Returns a descriptive string describing the munge errno `e`.

pymunge.raw.**munge_ctx_create**()
C prototype: `munge_ctx_t munge_ctx_create(void);`

Creates and returns a new munge context or None on error. Abandoning a context without calling *munge_ctx_destroy()* will result in a memory leak.

pymunge.raw.**munge_ctx_copy**(*ctx*)
C prototype: `munge_ctx_t munge_ctx_copy(munge_ctx_t ctx);`

Copies the context `ctx`, returning a new munge context or None on error. Abandoning a context without calling *munge_ctx_destroy()* will result in a memory leak.

pymunge.raw.**munge_ctx_destroy**(*ctx*)
C prototype: `void munge_ctx_destroy(munge_ctx_t ctx);`

Destroys the context `ctx`.

pymunge.raw.**munge_ctx_strerror**(*ctx*)
C prototype: `const char * munge_ctx_strerror(munge_ctx_t ctx);`

Returns a descriptive text string describing the munge error number according to the context `ctx`, or None if no error condition exists. This message may be more detailed than that returned by *munge_strerror()*.

pymunge.raw.**munge_ctx_get**(*ctx*, *opt*, *ptr*)
C prototype: `munge_err_t munge_ctx_get(munge_ctx_t ctx, munge_opt_t opt, ...);`

Note: when called from Python, returns nothing.

Gets the value for the option `opt` associated with the munge context `ctx`, storing the result in the subsequent pointer argument. Refer to the *munge_opt_t* enum comments for argument types. If the result is a string, that string should not be freed or modified by the caller. Raises a *MungeError* upon failure.

pymunge.raw.**munge_ctx_set**(*ctx*, *opt*, *val*)
C prototype: `munge_err_t munge_ctx_set(munge_ctx_t ctx, munge_opt_t opt, ...);`

Note: when called from Python, returns nothing.

Sets the value for the option `opt` associated with the munge context `ctx`, using the value of the subsequent argument. Refer to the *munge_opt_t* enum comments for argument types. Raises a *MungeError* upon failure.

pymunge.raw.**munge_enum_is_valid**(*type*, *val*)
C prototype: `int munge_enum_is_valid(munge_enum_t type, int val);`

Note: when called from Python, the returned int is converted to a boolean.

Returns True if the given value `val` is a valid enumeration of the specified type `type` in the software configuration as currently compiled; otherwise returns False. Some enumerations correspond to options that can only be enabled at compile-time.

pymunge.raw.**munge_enum_int_to_str**(*type*, *val*)
C prototype: `const char * munge_enum_int_to_str(munge_enum_t type, int val);`

Converts the munge enumeration `val` of the specified type `type` into a text string. Returns the text string, or None on error.

pymunge.raw.**munge_enum_str_to_int**(*type*, *str*)
C prototype: `int munge_enum_str_to_int(munge_enum_t type, const char *str);`

Converts the case-insensitive byte string `str` into the corresponding munge enumeration of the specified type `type`. Returns a munge enumeration on success (>= 0), or -1 on error.

pymunge.raw.**libmunge_filename = None**
Name of the libmunge shared object.

pymunge.raw.**libmunge = None**
Handle to the loaded libmunge shared object (a `ctypes.CDLL` object).

pymunge.raw.**MUNGE_OPT_CIPHER_TYPE = 0**
symmetric cipher type (int)

pymunge.raw.**MUNGE_OPT_MAC_TYPE = 1**
message auth code type (int)

pymunge.raw.**MUNGE_OPT_ZIP_TYPE = 2**
compression type (int)

pymunge.raw.**MUNGE_OPT_REALM = 3**
security realm (str)

pymunge.raw.**MUNGE_OPT_TTL = 4**
time-to-live (int)

pymunge.raw.**MUNGE_OPT_ADDR4 = 5**
src IPv4 addr (struct in_addr)

pymunge.raw.**MUNGE_OPT_ENCODE_TIME = 6**
time when cred encoded (time_t)

pymunge.raw.**MUNGE_OPT_DECODE_TIME = 7**
time when cred decoded (time_t)

pymunge.raw.**MUNGE_OPT_SOCKET = 8**
socket for comm w/ daemon (str)

pymunge.raw.**MUNGE_OPT_UID_RESTRICTION = 9**
UID able to decode cred (uid_t)

pymunge.raw.**MUNGE_OPT_GID_RESTRICTION = 10**
GID able to decode cred (gid_t)

pymunge.raw.**MUNGE_ENUM_CIPHER = 0**
cipher enum type

pymunge.raw.**MUNGE_ENUM_MAC = 1**
mac enum type

pymunge.raw.**MUNGE_ENUM_ZIP = 2**
zip enum type

The *pymunge.raw* module provides access to the low-level C API of libmunge. See the *module documentation*.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

M

P

R

S

T

U

Z