lastpasslib Documentation

Release 1.1.0

Costas Tyfoxylos

Sep 11, 2023

CONTENTS

Contents:

CHAPTER

ONE

LASTPASSLIB

A library able to retrieve and decrypt all items in lastpass along with their change history and attachments.

- Documentation: https://lastpasslib.readthedocs.org/en/latest

1.1 Project Features

- Can completely decrypt all secrets, attachments, and all history of every field that supports it.
- Can save the blob locally.
- Can save attachments of secrets.
- Exposes share info to and from people.

1.2 Project Inspiration

Initial inspiration was taken from https://github.com/konomae/lastpass-python. More features were needed and I could not really follow the design of that project so well, so I ended up rewriting all of it with a new design that made sense to me and implemented all the required features on that. This project is now quite further that the original project feature wise.

During my reverse engineering efforts I also found https://github.com/cfbao/lastpass-vault-parser/blob/master/lastpass-vault-format.md sadly a little too late. Also extended my model further than the documentation of that project.

CHAPTER

TWO

INSTALLATION

At the command line:

```
$ pip install lastpasslib
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv lastpasslib
$ pip install lastpasslib
```

Or, if you are using pipenv:

```
$ pipenv install lastpasslib
```

CHAPTER

THREE

USAGE

To use lastpasslib in a project:

```
from lastpasslib import Lastpass
lastpass = Lastpass(USERNAME, PASSWORD, MFA)

# Just showing a fragment of info exposed.

# iterate through all secrets:
for secret in lastpass.get_secrets():
    print(secret.name)
    # if a secret is shared print the info
    if secret.shared_to_people:
        for share in secret.shared_to_people:
            print(share)
    # if the secret type is password print note, username and password history if any.
    if secret.type == 'Password':
        if secret.note_history:
            for history in secret.note_history:
                print(history)
        if secret.username_history:
            for history in secret.username_history:
                print(history)
        if secret.password_history:
            for history in secret.password_history:
                print(history)
    else:
        # else it is a secure note type so print any history it has
        if secret.history:
            for history in secret.history:
                print(history)
```

CHAPTER

FOUR

DEVELOP

4.1 Development Workflow

The workflow supports the following steps

- lint
- test
- build
- document
- upload
- graph

These actions are supported out of the box by the corresponding scripts under _CI/scripts directory with sane defaults based on best practices. Sourcing setup_aliases.ps1 for windows powershell or setup_aliases.sh in bash on Mac or Linux will provide with handy aliases for the shell of all those commands prepended with an underscore.

The bootstrap script creates a .venv directory inside the project directory hosting the virtual environment. It uses pipenv for that. It is called by all other scripts before they do anything. So one could simple start by calling _lint and that would set up everything before it tried to actually lint the project

Once the code is ready to be delivered the _tag script should be called accepting one of three arguments, patch, minor, major following the semantic versioning scheme. So for the initial delivery one would call

```
$ _tag –minor
```

which would bump the version of the project to 0.1.0 tag it in git and do a push and also ask for the change and automagically update HISTORY.rst with the version and the change provided.

So the full workflow after git is initialized is:

- repeat as necessary (of course it could be test - code - lint :))
  - code
  - lint
  - test
- commit and push
- develop more through the code-lint-test cycle
- tag (with the appropriate argument)
- build
- upload (if you want to host your package in pypi)

- document (of course this could be run at any point)

4.2 Important Information

This template is based on pipenv. In order to be compatible with requirements.txt so the actual created package can be used by any part of the existing python ecosystem some hacks were needed. So when building a package out of this **do not** simple call

    $ python setup.py sdist bdist_egg

as this will produce an unusable artifact with files missing. Instead use the provided build and upload scripts that create all the necessary files in the artifact.

To develop on lastpasslib:

```
# The following commands require pipenv as a dependency

# To lint the project
_CI/scripts/lint.py

# To execute the testing
_CI/scripts/test.py

# To create a graph of the package and dependency tree
_CI/scripts/graph.py

# To build a package of the project under the directory "dist/"
_CI/scripts/build.py

# To see the package version
_CI/scripts/tag.py

# To bump semantic versioning [--major|--minor|--patch]
_CI/scripts/tag.py --major|--minor|--patch

# To upload the project to a pypi repo if user and password are properly provided
_CI/scripts/upload.py

# To build the documentation of the project
_CI/scripts/document.py
```

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Submit Feedback

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.

5.1.1 Get Started!

Ready to contribute? Here's how to set up *lastpasslib* for local development. Using of pipenv is highly recommended.

1. Clone your fork locally:

```
$ git clone https://github.com/schubergphilis/lastpasslib
```

2. Install your local copy into a virtualenv. Assuming you have pipenv installed, this is how you set up your clone for local development:

```
$ cd lastpasslib/
$ pipenv install --ignore-pipfile
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally. Do your development while using the CI capabilities and making sure the code passes lint, test, build and document stages.

4. Commit your changes and push your branch to the server:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

5. Submit a merge request

CHAPTER

SIX

LASTPASSLIB

6.1 lastpasslib package

6.1.1 Submodules

6.1.2 lastpasslib.configuration module

Main code for configuration.

class lastpasslib.configuration.**Configurations**

Bases: object

default = {'aid': '0', 'ajax': '1', 'auto': '1', 'extjs': '1', 'localupdate': '1', 'method': 'cr', 'requestsrc': 'cr', 'source': 'vault'}

move_secrets_payload = {'cmd': 'uploadaccounts', 'hasplugin': '4.119.0', 'lpversion': '4.119.0', 'pwprotect0': '0', 'realm0': '', 'requestsrc': 'cr', 'sessonly': '0', 'type0': 'cr'}

secret_payload = {'aid': '0', 'ajax': '1', 'auto': '1', 'extjs': '1', 'folder': 'none', 'localupdate': '1', 'method': 'cr', 'requestsrc': 'cr', 'source': 'vault', 'urid': '0'}

secure_note_payload = {'aid': '0', 'ajax': '1', 'auto': '1', 'extjs': '1', 'localupdate': '1', 'method': 'cr', 'notetype': 'Generic', 'password': '', 'requestsrc': 'cr', 'source': 'vault', 'template': '', 'totp': '', 'u': '', 'url': '', 'username': ''}

6.1.3 lastpasslib.datamodels module

Main code for datamodels.

class lastpasslib.datamodels.**Chunk**(*id: bytes*, *payload_size: bytes*, *payload: bytes*)

Bases: object

Models data of an encrypted chunk of the vault blob.

id: bytes

payload: bytes

payload_size: bytes

class lastpasslib.datamodels.**CompanyUser**(*email: str*, *img: str*, *name: str*, *type: str*, *uid: str*)

Bases: object

Models data of a company user.

email: str

img: str

name: str

type: str

uid: str

class lastpasslib.datamodels.**EquivalentDomain**(*id: int*, *url: str*)

Bases: object

Models data of an equivalent domain.

id: int

url: str

class lastpasslib.datamodels.**Event**(*_name1: str*, *_name2: str*, *_name3: str*, *_name4: str*, *_name5: str*, *name: str*, *group: str*, *date: str*, *ip: str*, *reverse: str*, *action: str*, *ulid: str*, *share_id: str*)

Bases: object

Models data of an event on the server.

action: str

date: str

property datetime

Datetime object of the date.

group: str

ip: str

name: str

property name_alternative

The alternative name of the event. Concatenating attributes name_1 to name_5.

reverse: str

share_id: str

ulid: str

class lastpasslib.datamodels.**Folder**(*name: str, path: tuple, id: Union[str, NoneType], encryption_key: str, parent: 'Folder' = None, folders: list = <factory>, secrets: list = <factory>, is_personal: bool = False*)

Bases: object

add_folder(*folder*)

add_folders(*folders*)

add_secret(*secret*)

add_secrets(*secrets*)

encryption_key: str

folders: list

property full_path

get_secret(*secret_name*)

id: Optional[str]

property is_in_root

is_personal: bool = False

name: str

parent: *Folder* = None

path: tuple

secrets: list

class lastpasslib.datamodels.**FolderMetadata**(*path: tuple*, *id: Union[str, NoneType]*, *encryption_key: str*, *is_personal: bool*)

Bases: object

encryption_key: str

id: Optional[str]

is_personal: bool

path: tuple

class lastpasslib.datamodels.**NeverUrl**(*id: int*, *url: str*)

Bases: object

Models data of a never url.

id: int

url: str

class lastpasslib.datamodels.**SharedFolder**(*id: str*, *read_only: str*, *give: str*, *name: str*, *deleted: str*, *last_modified: str*, *association: str*, *can_administer: str*, *invisible: str*, *created: str*, *cgid: str*, *download: str*, *outside_enterprise: str*, *cid: str*, *share_data: str = ''*, *sharer: str = ''*, *shared_name: str = ''*)

Bases: object

Models data of a shared folder.

association: str

can_administer: str

cgid: str

cid: str

created: str

deleted: str

download: str

give: str

id: str

invisible: str

last_modified: str

property last_modified_datetime

Datetime object of the last modified date.

name: str

outside_enterprise: str

read_only: str

share_data: str = ''

shared_name: str = ''

sharer: str = ''

class lastpasslib.datamodels.**UrlRule**(*url: str*, *exact_host: bool*, *exact_port: bool*, *case_insensitive: bool*)

Bases: object

Models data of a url rule.

case_insensitive: bool

exact_host: bool

exact_port: bool

url: str

6.1.4 lastpasslib.dataschemas module

Main code for dataschemas.

class lastpasslib.dataschemas.**AttachmentSchema**

Bases: object

class lastpasslib.dataschemas.**SecretSchema**

Bases: object

class lastpasslib.dataschemas.**SharedFolderSchema**

Bases: object

6.1.5 lastpasslib.encryption module

Main code for encryption.

class lastpasslib.encryption.**Blob**(*blob*)

Bases: object

Models the encrypted blob and implements functionality to traverse it and split it into encrypted chunks.

property chunks

The chunks of the blob.

static is_complete(*chunks*)

If the blob is complete.

The last chunk of the encrypted blob should be an entry of "ENDM" with payload of "OK"

Parameters
chunks – The collection of chunks of the blob.

Returns
True is the blob is complete, False otherwise.

class lastpasslib.encryption.**EncryptManager**

Bases: object

Handles the decryption and decoding for all appropriate methods.

static create_random_iv(*byte_size: int = 16*) → bytes

Creates an Initialization Vector (IV) byte string for a given length.

Parameters
byte_size (*int*) – length of the byte string. Defaults to 16.

Returns
Byte string

Return type
bytes

static decode_hex(*data*)

Decodes a hex encoded string into raw bytes.

Parameters
data – The data to decode

Returns
The decoded data on success.

Raises
TypeError if the decoding is not possible. –

static decrypt_aes256_auto(*data*, *encryption_key*, *base64=False*)

Guesses AES cipher (ECB or CBD) from the length of the plain data.

Parameters

- **data** – The data to decrypt.
- **encryption_key** – The key to use to decrypt the data.
- **base64** – Flag of whether the payload is base64 encoded or plain text encrypted.

Returns
The decrypted data of the payload.

Raises
TypeError – The data is not of type bytes

static decrypt_aes256_cbc(*iv: bytes*, *data: bytes*, *encryption_key: bytes*) → bytes

Decrypt AES-256 bytes with CBC.

Parameters

- **iv** (*bytes*) – The initialization vector
- **data** (*bytes*) – The data to decrypt
- **encryption_key** (*bytes*) – The key used to decrypt

Returns
Byte string

Return type
bytes

static decrypt_aes256_ecb(*data: bytes*, *encryption_key: bytes*) → bytes

Decrypt AES-256 bytes with ECB.

Parameters

- **data** (*bytes*) – The data to decrypt
- **encryption_key** (*bytes*) – The key used to decrypt

Returns
Byte string

Return type
bytes

static decrypt_rsa_key(*payload*, *encryption_key*)

Parse PRIK chunk which contains a private RSA key and decrypt it.

Parameters

- **payload** – The payload that holds the encrypted rsa key.
- **encryption_key** – The key to use to decrypt the payload.

Returns
A decrypted RSA key.

static encode_hex(*data*)

Encodes a raw bytes string to a hex encoded string.

Parameters
data – The data to encode

Returns
The encoded data on success.

Raises
TypeError if the encoding is not possible. –

static encrypt_aes256_cbc(*iv: bytes*, *data: bytes*, *encryption_key: bytes*) → bytes

Encrypt AES-256 bytes with CBC.

Parameters

- **iv** (*bytes*) – The initialization vector
- **data** (*bytes*) – The data to encrypt
- **encryption_key** (*bytes*) – The key used to encrypt

Returns
Byte string of hex

Return type
bytes

static encrypt_and_encode_payload(*encryption_key: str*, *payload: str*) → str

aes256_cbc encrypting and encoding a payload.

Parameters

- **encryption_key** (*str*) – _description_
- **payload** (*str*) – _description_

Returns
description

Return type
str

class lastpasslib.encryption.**Stream**(*data*)

Bases: object

Models a stream of encrypted data and implements appropriate data retrieval capabilities.

An item in an itemized chunk is made up of the # big endian size and the payload of that size. # # Example: # 0000: 4 # 0004: 0xDE 0xAD 0xBE 0xEF # 0008: — Next item —

get_payload_by_size(*payload_size*)

Reads a payload from a stream by the provided size and returns it as bytes.

Parameters
payload_size (*bytes*) – The payload size to retrieve.

Returns
The payload.

Return type
bytes

property position

The current position of the stream.

read_byte_size(*size*)

Reads the next size provided bytes from a stream and returns it as bytes.

Parameters
size – An integer for the size to retrieve.

Returns
The bytes for the provided size.

Return type
bytes

skip_item(*times=1*)

Skips an item in a stream as many times as provided.

Parameters
times (*int*) – The times to skip the payload.

Returns
None

6.1.6 lastpasslib.lastpasslib module

Main code for lastpasslib.

class lastpasslib.lastpasslib.**Lastpass**(*username*, *password*, *mfa*, *domain='lastpass.com'*)

Bases: object

Models the main service and exposes the vault object and helper methods to interact and retrieve data.

property attachments

The attachments of the vault.

create_password(*name: str*, *url: Optional[str] = None*, *folder_path: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, *totp: Optional[str] = None*, *notes: Optional[str] = None*, *pwprotect: bool = False*, *auto_login: bool = False*, *autofill: bool = False*, *favorite: bool = False*) → bool

Creates a password.

Parameters

- **name** (*str*) – name
- **url** (*str, optional*) – url. Defaults to None.
- **folder_path** (*str, optional*) – folder path. Defaults to None.
- **username** (*str, optional*) – username. Defaults to None.
- **password** (*str, optional*) – password. Defaults to None.
- **totp** (*str, optional*) – totp. Defaults to None.
- **notes** (*str, optional*) – notes. Defaults to None.
- **pwprotect** (*bool, optional*) – pwprotect. Defaults to False.
- **auto_login** (*bool, optional*) – auto_login. Defaults to False.
- **autofill** (*bool, optional*) – autofill. Defaults to False.
- **favorite** (*bool, optional*) – favorite. Defaults to False.

Returns
True at success, False at failure.

Return type
bool

create_secure_note(*name: str*, *folder_path: Optional[str] = None*, *notes: Optional[str] = None*, *favorite: bool = False*) → bool

Creates a secure note.

Parameters

- **name** (*str*) – name.
- **folder_path** (*str, optional*) – folder path. Defaults to None.
- **notes** (*str, optional*) – notes. Defaults to None.
- **favorite** (*bool, optional*) – favorite. Defaults to False.

Returns
True at success, False at failure.

Return type
bool

property csrf_token

The csrf token required for some calls.

decrypt_blob(*blob*)

Decrypts a provided blob of a vault back up and returns the decrypted blob.

Parameters
blob – The blob to decrypt.

Returns
The decrypted blob.

Return type
DecryptedBlob

property decrypted_vault

delete_password_by_id(*id_*)

Deletes a password from the vault by id.

Parameters
id – The id to match on

Returns
True on success, False otherwise.

Return type
bool

Raises
MultipleInstances – If more than one password is found with the same name.

delete_password_by_name(*name*)

Deletes a password from the vault by name.

Parameters
name – The name to match on, case-sensitive.

Returns
True on success, False otherwise.

Return type
bool

Raises
MultipleInstances – If more than one password is found with the same name.

`delete_secret_by_id`(*id_*)

Deletes a secret from the vault by id.

Parameters
id – The id to match on

Returns
True on success, False otherwise.

Return type
bool

Raises
MultipleInstances – If more than one password is found with the same name.

`delete_secret_by_name`(*name*)

Deletes a secret from the vault by name.

Parameters
name – The name to match on, case-sensitive.

Returns
True on success, False otherwise.

Return type
bool

Raises
MultipleInstances – If more than one password is found with the same name.

`delete_secure_note_by_id`(*id_*)

Deletes a secure notes from the vault by id.

Parameters
id – The id to match on

Returns
True on success, False otherwise.

Return type
bool

Raises
MultipleInstances – If more than one secure note is found with the same name.

`delete_secure_note_by_name`(*name*)

Deletes a secure note from the vault by name.

Parameters
name – The name to match on, case-sensitive.

Returns
True on success, False otherwise.

Return type
bool

Raises
MultipleInstances – If more than one secure note is found with the same name.

property encrypted_username

The encrypted username of the user.

property equivalent_domains

The equivalent domains of the vault.

property folders

All the folders of the vault.

Returns
A list of all the folders of the vault.

get_attachments()

Gets all attachments from all secrets in the vault.

Returns
A list of attachment objects from all secrets of the vault.

Return type
list

get_company_user_by_email(*email*)

Gets a company user that exactly match a provided email.

Parameters
email – The email to match to.

Returns
A company user object if a match is found, else None.

get_company_users_by_email(*email_part*)

Gets a list of company users that match a fragment of the email.

Parameters
email_part – The fragment of the email to match to.

Returns
A list of company users that match the fragment provided.

get_event_history_by_date(*start_date=None*, *end_date=None*)

Get generic history events by a range of dates.

Parameters

- **start_date** – The start date of the range. Defaults to today if not provided.
- **end_date** – The end date of the range. Defaults to today if not provided.

Returns
A list of generic history events by the provided date range.

get_folder_by_name(*name*)

Gets a folder by name.

Parameters
name – The name of the folder to match.

Returns
The folder it matched on if there is one match only, None if no match found.

Raises
MultipleInstances – If there is more than one match with the same name.

`get_folder_by_path`(*path: str*) → *Folder*

Gets a folder by path.

Parameters
path (*str*) – A string with '\' as seperator.

Returns
The first folder it matched on based on path. None if no match found.

Return type
Folder

`get_login_history_by_date`(*start_date=None*, *end_date=None*)

Get login history events by a range of dates.

Parameters

- **start_date** – The start date of the range. Defaults to today if not provided.
- **end_date** – The end date of the range. Defaults to today if not provided.

Returns
A list of login history events by the provided date range.

`get_password_by_id`(*id_*)

Gets a password from the vault by id.

Parameters
id – The id to match on.

Returns
The password if a match is found, else None.

`get_password_by_name`(*name*)

Gets password from the vault matching a name.

Parameters

- **name** – The name to match on, case-sensitive.
- **filter** – The type of secret to filter on.

Returns
A list of passwords if they match the name, an empty list otherwise.

Return type
list

`get_passwords`()

Gets only the passwords from the vault.

Returns
A list of password type secrets.

`get_passwords_by_group`(*group_name*)

Gets passwords from the vault for the specified group.

Parameters
group_name – The name to match on, case-sensitive.

Returns
A list of passwords if they match the group name, an empty list otherwise.

Return type
list

`get_passwords_by_name`(*name*)

Gets passwords from the vault matching a name.

Parameters

- **name** – The name to match on, case-sensitive.
- **filter** – The type of secret to filter on.

Returns
A list of passwords if they match the name, an empty list otherwise.

Return type
list

`get_passwords_by_shared_folder`(*folder_name*)

Gets passwords from the vault for the specified shared folder.

Parameters
folder_name – The name to match on, case-sensitive.

Returns
A list of passwords of the shared folder, an empty list otherwise.

Return type
list

`get_passwords_with_attachments`()

Gets passwords with attachments.

Returns
A list of passwords with attachments.

Return type
list

`get_passwords_with_password_updated_before_date`(*date*)

Gets passwords with passwords updates before the given date.

Parameters
date – The date to match with. Parsing is applied on the date so any sane format will work. example: '22 sep 2022' or '22-09-2002' or '22/09/2022' should all work fine. To avoid ambiguity between US and EU date format a format with a named month is preferred.

Returns
A list of passwords that their password field had been updated before the given date.

`get_secret_by_full_path`(*path*, *name*)

Gets a secret from the vault by name.

Parameters

- **path** – The full path to the secret.
- **name** – The name to match on, case-sensitive.

Returns
The secret if a match is found, else None.

`get_secret_by_id`(*id_*)

Gets a secret from the vault by id.

Parameters
id – The id to match on.

Returns
The secret if a match is found, else None.

`get_secret_by_name`(*name*)

Gets a secret from the vault by name.

Parameters
name – The name to match on, case-sensitive.

Returns
The secret if a match is found, else None.

Raises
MultipleInstances – If more than one password is found with the same name.

`get_secrets`(*filter_=None*)

Gets secrets from the vault.

Parameters
filter – The secret type or types to filter.

Returns
A list of secrets matching the filter or all secrets of the vault.

Return type
list

`get_secrets_by_group`(*group_name*, *filter_=None*)

Gets secrets from the vault for the specified group.

Parameters

- **group_name** – The name to match on, case-sensitive.
- **filter** – The type of secret to filter on.

Returns
A list of secrets if they match the group name, an empty list otherwise.

Return type
list

`get_secrets_by_name`(*name*, *filter_=None*)

Gets secrets from the vault matching a name.

Parameters

- **name** – The name to match on, case-sensitive.
- **filter** – The type of secret to filter on.

Returns
A list of secrets if they match the name, an empty list otherwise.

Return type
list

`get_secrets_by_shared_folder`(*folder_name*, *filter_=None*)

Gets secrets from the vault for the specified shared folder.

Parameters

- `folder_name` – The name to match on, case-sensitive.
- `filter` – The type of secret to filter on.

Returns
A list of secrets of the shared folder, an empty list otherwise.

Return type
list

`get_secrets_shared_directly`()

Gets secrets that have been shared directly and not as part of a shared folder.

Returns
A list of secrets that have been shared directly.

Return type
list

`get_secrets_with_attachments`()

Gets secrets with attachments.

Returns
A list of secrets with attachments.

Return type
list

`get_secure_note_by_id`(*id_*)

Gets a secure note from the vault by id.

Parameters
`id` – The id to match on.

Returns
The secure note if a match is found, else None.

`get_secure_note_by_name`(*name*)

Gets secure note from the vault matching a name.

Parameters
`name` – The name to match on, case-sensitive.

Returns
A list of secure note if they match the name, an empty list otherwise.

Return type
list

`get_secure_notes`()

Gets only secure notes for the vault.

Returns
A list of secure note type secrets.

get_secure_notes_by_group(*group_name*)

Gets secure notes from the vault for the specified group.

Parameters
group_name – The name to match on, case-sensitive.

Returns
A list of secure notes if they match the group name, an empty list otherwise.

Return type
list

get_secure_notes_by_name(*name*)

Gets secure notes from the vault matching a name.

Parameters

- name – The name to match on, case-sensitive.
- filter – The type of secret to filter on.

Returns
A list of secure notes if they match the name, an empty list otherwise.

Return type
list

get_secure_notes_by_shared_folder(*folder_name*)

Gets secure notes from the vault for the specified shared folder.

Parameters
folder_name – The name to match on, case-sensitive.

Returns
A list of secure notes of the shared folder, an empty list otherwise.

Return type
list

get_secure_notes_updated_before_date(*date*)

Gets secure notes with updates before the given date.

Parameters
date – The date to match with. Parsing is applied on the date so any sane format will work. example: '22 sep 2022' or '22-09-2002' or '22/09/2022' should all work fine. To avoid ambiguity between US and EU date format a format with a named month is preferred.

Returns
A list of secure notes that have been updated before the given date.

get_secure_notes_with_attachments()

Gets secure notes with attachments.

Returns
A list of secure notes with attachments.

Return type
list

property iteration_count

The iteration count of the encryption for the vault.

logout()

Logs out of the session.

move_secret_to_folder(*secret_full_path: str*, *folder_path: str*) → bool

Moving a secret from a folder to another folder.

Parameters

- **secret_full_path** (*str*) – the path where the current secret is stored.
- **folder_path** (*str*) – location to the folder where the secret should move to.

Returns
True at success, False at failure.

Return type
bool

property never_urls

The never urls of the vault.

property personal_folders

Retrieves all folders of the vault that are personal and not shared.

Returns
A list of personal folders.

Return type
list

refresh()

Refreshes the vault by getting the blob again and decrypting everything.

Returns
True on success, False otherwise.

refresh_session(*mfa=None*, *client_id=None*)

property root_folder

The root folder of the lastpass vault.

Holds all sub folders and secrets saved in.

Returns
The root folder.

Return type
Folder

save_vault_blob(*path='.'*, *name='vault.blob'*)

Can save the downloaded blob.

Parameters

- **path** – The path to save the blob to, defaults to local directory.
- **name** – The name to save the blob as, defaults to "vault.blob".

Returns
None.

property session_id

The session ID.

property shared_folders

Retrieves all shared folders of the vault.

Returns
A list of shared folders.

Return type
list

property token

The token returned to be used for api calls.

property uid

The uid of lastpass.

property url_rules

The url rules of the vault.

6.1.7 lastpasslib.lastpasslibexceptions module

Custom exception code for lastpasslib.

exception lastpasslib.lastpasslibexceptions.**ApiLimitReached**

Bases: Exception

Server responded with a 429 status.

exception lastpasslib.lastpasslibexceptions.**InvalidMfa**

Bases: Exception

The mfa token provided is invalid.

exception lastpasslib.lastpasslibexceptions.**InvalidPassword**

Bases: Exception

The password provided is invalid.

exception lastpasslib.lastpasslibexceptions.**InvalidSecretType**

Bases: Exception

The secret type provided is not a valid one.

exception lastpasslib.lastpasslibexceptions.**InvalidYubiKey**

Bases: Exception

The yubikey token provided is invalid.

exception lastpasslib.lastpasslibexceptions.**MfaRequired**

Bases: Exception

A mfa token is required but not provided.

exception lastpasslib.lastpasslibexceptions.**MissingResult**

Bases: Exception

Server response does not contain a result.

exception lastpasslib.lastpasslibexceptions.**MobileDevicesRestricted**

Bases: Exception

Mobile devices are restricted on the Account settings of lastpass.

exception lastpasslib.lastpasslibexceptions.**MultipleInstances**

Bases: Exception

There is more than one item returned.

exception lastpasslib.lastpasslibexceptions.**RemoteCommandInvalidResult**

Bases: Exception

The result of the Remote Command is not valid.

exception lastpasslib.lastpasslibexceptions.**ServerError**

Bases: Exception

Server responded with some error.

exception lastpasslib.lastpasslibexceptions.**UnexpectedResponse**

Bases: Exception

The response provided does not follow the expected format.

exception lastpasslib.lastpasslibexceptions.**UnknownAccountID**

Bases: Exception

No Account ID is found.

exception lastpasslib.lastpasslibexceptions.**UnknownFolder**

Bases: Exception

No folder is found.

exception lastpasslib.lastpasslibexceptions.**UnknownIP**

Bases: Exception

The ip of the connection is not know to the service.

exception lastpasslib.lastpasslibexceptions.**UnknownSecret**

Bases: Exception

No secret is found.

exception lastpasslib.lastpasslibexceptions.**UnknownUsername**

Bases: Exception

The username provided is not known to the server.

6.1.8 lastpasslib.secrets module

Main code for secrets.

class lastpasslib.secrets.**Address**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models an Address secure note.

attribute_mapping = {'Address 1': 'address_1', 'Address 2': 'address_2', 'Address 3': 'address_3', 'Birthday': 'birthday', 'City / Town': 'city_town', 'Company': 'company', 'Country': 'country', 'County': 'country', 'Email Address': 'email_address', 'Evening Phone': 'evening_phone', 'Fax': 'fax', 'First Name': 'first_name', 'Gender': 'gender', 'Language': 'language', 'Last Name': 'last_name', 'Middle Name': 'middle_name', 'Mobile Phone': 'mobile_phone', 'Notes': 'notes', 'Phone': 'phone', 'State': 'state', 'Timezone': 'timezone', 'Title': 'title', 'Username': 'username', 'Zip / Postal Code': 'zip_postal_code'}

class lastpasslib.secrets.**Attachment**(*lastpass_instance*, *data*)

Bases: object

Models an attachment of a secret.

property content

The content of the attachment.

property filename

The filename of the attachment.

property id

ID of the attachment.

property mimetype

The mimetype of the attachment.

property parent_id

ID of the parent secret of the attachment.

property parent_secret

save(*path='.'*)

Saves the attachment on a given path, current working directory if not provided.

Parameters
path – The path to save the attachment to, defaults to current working directory.

Returns
None.

property uuid

The uuid of the attachment.

class lastpasslib.secrets.**BankAccount**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Bank Account secure note.

attribute_mapping = {'Account Number': 'accounting_number', 'Account Type': 'account_type', 'Bank Name': 'bank_name', 'Branch Address': 'branch_address', 'Branch Phone': 'branch_phone', 'IBAN Number': 'iban_number', 'Language': 'language', 'Notes': 'notes', 'Pin': 'pin', 'Routing Number': 'routing_number', 'SWIFT Code': 'swift_code'}

class lastpasslib.secrets.**CreditCard**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Credit Card secure note.

attribute_mapping = {'Expiration Date': 'expiration_date', 'Language': 'language', 'Name on Card': 'name_on_card', 'Notes': 'notes', 'Number': 'number', 'Security Code': 'security_code', 'Start Date': 'start_date', 'Type': 'type'}

class lastpasslib.secrets.**Custom**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Custom secure note.

property attribute_mapping

Attribute mapping.

class lastpasslib.secrets.Database(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Database secure note.

attribute_mapping = {'Alias': 'alias', 'Database': 'database', 'Hostname': 'hostname', 'Language': 'language', 'Notes': 'notes', 'Password': 'password', 'Port': 'port', 'SID': 'sid', 'Type': 'type', 'Username': 'username'}

class lastpasslib.secrets.DriverLicense(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Driver license secure note.

attribute_mapping = {'Address': 'address', 'City / Town': 'city_town', 'Country': 'country', 'Date of Birth': 'date_of_birth', 'Expiration Date': 'expiration_date', 'Height': 'height', 'Language': 'language', 'License Class': 'license_class', 'Name': 'name', 'Notes': 'notes', 'Number': 'number', 'Sex': 'sex', 'State': 'state', 'ZIP / Postal Code': 'zip_postal_code'}

class lastpasslib.secrets.EmailAccount(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Email Account secure note.

attribute_mapping = {'Language': 'language', 'Notes': 'notes', 'Password': 'password', 'Port': 'port', 'SMTP Port': 'smtp_port', 'SMTP Server': 'smtp_server', 'Server': 'server', 'Type': 'type', 'Username': 'username'}

class lastpasslib.secrets.FolderEntry(*lastpass_instance*, *data*, *shared_folder=None*)

Bases: *Secret*

property name

Name.

class lastpasslib.secrets.Generic(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Generic secure note.

attribute_mapping = {}

property notes

class lastpasslib.secrets.HealthInsurance(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Health Insurance secure note.

attribute_mapping = {'Co-pay': 'co_pay', 'Company': 'company', 'Company Phone': 'company_phone', 'Group ID': 'insurance_group_id', 'Language': 'language', 'Member ID': 'member_id', 'Member Name': 'member_name', 'Notes': 'notes', 'Physician Address': 'physician_address', 'Physician Name': 'physician_name', 'Physician Phone': 'physician_phone', 'Policy Number': 'policy_number', 'Policy Type': 'policy_type'}

class lastpasslib.secrets.**History**(*date: str*, *value: str*, *person: str*)

Bases: object

Models data of a history event on the server.

date: str

property datetime

Datetime object of the date.

person: str

value: str

class lastpasslib.secrets.**InstantMessenger**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Instant Messenger secure note.

attribute_mapping = {'Language': 'language', 'Notes': 'notes', 'Password': 'password', 'Port': 'port', 'Server': 'server', 'Type': 'type', 'Username': 'username'}

class lastpasslib.secrets.**Membership**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Membership secure note.

attribute_mapping = {'Expiration Date': 'expiration_date', 'Language': 'language', 'Member Name': 'member_name', 'Membership Number': 'membership_number', 'Notes': 'notes', 'Organization': 'organization', 'Password': 'password', 'Start Date': 'start_date', 'Telephone': 'telephone', 'Website': 'website'}

class lastpasslib.secrets.**Passport**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Passport secure note.

attribute_mapping = {'Country': 'country', 'Date of Birth': 'date_of_birth', 'Expiration Date': 'expiration_date', 'Issued Date': 'issued_date', 'Issuing Authority': 'issuing_authority', 'Language': 'language', 'Name': 'name', 'Nationality': 'nationality', 'Notes': 'notes', 'Number': 'number', 'Sex': 'sex', 'Type': 'type'}

class lastpasslib.secrets.**Password**(*lastpass_instance*, *data*, *shared_folder=None*)

Bases: *Secret*

Models a password and exposes appropriate attributes.

property action

Action of the password if any.

property auto_login

Flag set if auto login is set.

get_latest_password_update_person()

The email of the last person that updated the password if any, else None.

property is_generated_password

Flag if this is an auto generated password.

property mfa_seed
The mfa seed of the password if set.

property never_autofill
Flag whether the autofill is set.

property note_history
The note history objects of the password if any.

property notes
The notes of the password.

property password
The password field of the password.

property password_history
The note password objects of the password if any.

property secret_updated_datetime

property username
The username field of the password.

property username_history
The note username objects of the password if any.

class lastpasslib.secrets.Secret(*lastpass_instance*, *data*, *shared_folder=None*)
Bases: object

Models the secret and exposes the main attributes that are shared across Passwords and Secure Notes.

add_attachment(*attachment*)
Adds an attachment to the list of attachments on the secret.

Used as part of the secret decryption process by the vault object adding all relevant attachments to the appropriate secret.

Parameters
attachment – The attachment to add to the secret.

Returns
None

property attachment_encryption_key
The attachment encryption key if any.

property attachments
The attachments of the secret if any.

property created_datetime
A datetime object of the created at date of the secret.

delete()
Deletes the secret from Lastpass.

property encryption_key
The encryption key that is used on the encrypted data of the secret.

property full_path
The full path of where the secret is stored.

property group
: Group name of the secret.

property group_id
: Group id of the secret.

property has_attachment
: Flag of whether the secret has attachments.

property has_been_shared
: Flag of whether the secret has been shared with people.

property id
: ID.

property is_deleted
: Flag of the deletion state of the secret.

property is_favorite
: Is favorite flag.

property is_individual_share
: Flag of whether the secret is an individual share or a share as part of a shared folder.

property is_password_protected
: Flag of whether the secret is password protected.

property is_secure_note
: Flag of whether the secret is a secure note.

property last_modified_datetime
: A datetime object of the last modified date of the secret.

property last_password_change_datetime
: A datetime object of the last password change of the secret, relevant for Passwords.

property last_touch_datetime
: A datetime object of the last touch date of the secret.

move_to_folder(*folder_path: str*)
: Move the secret to another folder.

  Parameters
  folder_path (*str*) – folder path.

  Returns
  True at success, False at failure.

  Return type
  bool

property name
: Name.

property shared_folder
: A shared folder object of the parent share folder if any else None.

property shared_from_id
: The id of the user sharing the secret if it is an individual share.

property shared_to_people

List of people the secret has been shared with.

property type

The type of the secret.

property url

The url of the secret.

class lastpasslib.secrets.**SecureNote**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *Secret*

Models a secure note.

attribute_mapping = {}

property history

History of the secure note edits if any.

property secret_updated_datetime

class lastpasslib.secrets.**Server**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a Server secure note.

attribute_mapping = {'Hostname': 'hostname', 'Language': 'language', 'Notes': 'notes', 'Password': 'password', 'Username': 'username'}

class lastpasslib.secrets.**ShareAction**(*company_username: str*, *date: str*, *email: str*, *give: str*, *share_date: str*, *state: str*, *_uid: str*)

Bases: `object`

Models data of a share action of a secret.

property accepted

Boolean of the accepted status of the share.

company_username: str

date: str

property datetime

Datetime object of the date.

email: str

give: str

property given

Boolean of the given status of the share.

property id

ID of the share action, correlates with the ID of the user part of the share.

share_date: str

property share_datetime

Datetime object of the share date.

state: str

class lastpasslib.secrets.**SocialSecurity**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a SocialSecurity secure note.

attribute_mapping = {'Language': 'language', 'Name': 'name', 'Notes': 'notes', 'Number': 'number'}

class lastpasslib.secrets.**SoftwareLicense**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a SoftwareLicense secure note.

attribute_mapping = {'Language': 'language', 'License Key': 'license_key', 'Licensee': 'licensee', 'Notes': 'notes', 'Number of Licenses': 'number_of_licenses', 'Order Number': 'order_number', 'Order Total': 'order_total', 'Price': 'price', 'Publisher': 'publisher', 'Purchase Date': 'purchase_date', 'Support Email': 'support_email', 'Version': 'version', 'Website': 'website'}

class lastpasslib.secrets.**SshKey**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a SshKey secure note.

attribute_mapping = {'Bit Strength': 'bit_strength', 'Date': 'date', 'Format': 'format', 'Hostname': 'hostname', 'Language': 'language', 'Notes': 'notes', 'Passphrase': 'passphrase', 'Private Key': 'private_key', 'Public Key': 'public_key'}

class lastpasslib.secrets.**WifiPassword**(*lastpass_instance*, *data*, *shared_folder*)

Bases: *SecureNote*

Models a WifiPassword secure note.

attribute_mapping = {'Authentication': 'authentication', 'Connection Mode': 'connection_mode', 'Connection Type': 'connection_type', 'Encryption': 'encryption', 'FIPS Mode': 'fips_mode', 'Key Index': 'key_index', 'Key Type': 'key_type', 'Language': 'language', 'Notes': 'notes', 'Password': 'password', 'Protected': 'protected', 'SSID': 'ssid', 'Use 802.1X': 'use_8021x'}

6.1.9 lastpasslib.utils module

class lastpasslib.utils.**LastpassMock**(*username*, *domain='lastpass.com'*)

Bases: `object`

get_shared_folder_by_id(*id_*)

6.1.10 lastpasslib.vault module

Main code for vault.

class lastpasslib.vault.**DecryptedVault**(*lastpass_instance*, *encrypted_username*, *attachments*, *never_urls*, *equivalent_domains*, *url_rules*, *secrets*, *encryption_key*, *folder_entries*, *shared_folders*)

Bases: object

clear_folders()

create_secret(*secret_type*, *data*)

delete_secret_by_id(*id_*)

property folders

All the folders of the vault.

Returns
A list of all the folders of the vault.

class lastpasslib.vault.**Vault**(*lastpass_instance*, *password*)

Bases: object

Models the encrypted vault and implements decryption of all items and connection everything appropriately.

property blob

decrypt_blob(*data*)

property hash

The hash of the vault.

property key

The encryption key of the vault.

refresh()

Refreshes the vault by cleaning up the encrypted blob and the decrypted secrets and forcing the retrieval.

save(*path='.'*, *name='vault.blob'*, *timestamp=True*)

Can save the downloaded blob.

Parameters

- **path** – The path to save the blob to, defaults to local directory.
- **name** – The name to save the blob as, defaults to "vault.blob".

Returns
None.

6.1.11 Module contents

lastpasslib package.

Import all parts from lastpasslib here

CHAPTER

SEVEN

CREDITS

7.1 Development Lead

- Costas Tyfoxylos <ctyfoxylos@schubergphilis.com>
- Yorick Hoorneman <yhoorneman@schubergphilis.com>

7.2 Contributors

None yet. Why not be the first?

CHAPTER

EIGHT

HISTORY

CHAPTER

NINE

0.0.1 (08-02-2023)

- First code creation

CHAPTER

TEN

0.1.0 (11-02-2023)

- Initial release

CHAPTER

ELEVEN

0.2.0 (17-02-2023)

- Implement retrieving secrets by group and shared folder.

CHAPTER

TWELVE

0.3.0 (17-02-2023)

- Implement friendly interface to retrieve passwords and secure notes by group and shared folder.

CHAPTER

THIRTEEN

0.4.0 (19-02-2023)

- Implement folder grouping and retrieval.

CHAPTER

FOURTEEN

0.5.0 (24-02-2023)

- Implement folder filtering out on secret parsing.

CHAPTER

FIFTEEN

0.6.0 (24-02-2023)

- Implement a consistent interface for update datetime reporting.

CHAPTER

SIXTEEN

0.7.0 (01-03-2023)

- Implement root folder, personal folders and shared folders. Report on password change for secure notes that support it.

CHAPTER

SEVENTEEN

0.7.1 (01-03-2023)

- Expose only one level of personal folders.

CHAPTER

EIGHTEEN

0.7.2 (01-03-2023)

- Refactor to match on full share name.

CHAPTER

NINETEEN

0.7.3 (08-03-2023)

- Decouple decrypted vault from vault functionality.

CHAPTER

TWENTY

0.7.4 (08-03-2023)

- Decouple decrypted vault from vault functionality.

CHAPTER

TWENTYONE

0.7.5 (13-03-2023)

- Hide possible logout error.

CHAPTER

TWENTYTWO

0.7.6 (20-03-2023)

- Testing release.

CHAPTER

TWENTYTHREE

0.7.7 (21-03-2023)

- Implement better error messages on some faults.

CHAPTER

TWENTYFOUR

0.8.0 (08-06-2023)

- Implement support for yubikey MFA

CHAPTER

TWENTYFIVE

1.0.0 (04-09-2023)

- Implementing creating and moving of secrets.

CHAPTER

TWENTYSIX

1.0.1 (07-09-2023)

- Fix typo error.

CHAPTER

TWENTYSEVEN

1.1.0 (11-09-2023)

- Implement session refresh. Fix attachment retrieval for attachments in shared folders. Properly identify the attachment mode (text, binary)

CHAPTER

TWENTYEIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W