flask-ldap3-login Documentation

Release 0.0.0.dev0

Nick Whyte

Jul 03, 2017

Contents

1 Contents: **3**
1.1 Configuration 3
1.2 Quick Start 5
1.3 API 9

Python Module Index **15**

Flask LDAP3 Login allows you to easily integrate your flask app with an LDAP directory. It can be used as an extension to Flask-Login and can even be used with Flask-Principal for permission and privilege management.

Flask LDAP3 Login uses the ldap3 library, maintaining compatability with python 3.4 and backwards.

Flask LDAP3 Login Will:

- Allow you to query whether or not a user's credentials are correct
- Query the directory for users details
- Query the directory for group details
- Query the directory for users group memberships
- Provide a contextual `ldap_manager.connection` object (`ldap3.Connection`) which can be used in any flask request context. Useful for writing your own more advanced queries.

Flask LDAP3 Login Wont:

- Provide a login/logout mechanism. You need to provide this with something like flask-login
- Provide any extension to the application's session. User tracking and group tracking should be done via flask-login and flask-principal

View the Full Documentation at ReadTheDocs

CHAPTER 1

Contents:

Configuration

The following configuration values are used by Flask-Security:

Core

`LDAP_PORT`	Specifies the port to use when connecting to LDAP. Defaults to `389`.
`LDAP_HOST`	Speficies the address of the server to connect to by default. `None`. Additional servers can be added via the `add_server` method.
`LDAP_USE_SSL`	Specifies whether the default server connection should use SSL. Defaults to `False`.
`LDAP_ADD_SERVER`	Specifies whether the default server as specified in `LDAP_HOST` should be added to the server pool. Defaults to `True`. Servers can be added via the `add_server` method.
`LDAP_READONLY`	Specified if connections made to the server are readonly. Defaults to `True`
`LDAP_BIND_DIRECT_CREDENTIALS`	Instead of searching for a DN of a user you can instead bind directly to the directory. Setting this `True` will perform binds without formatting the username parameter. This is useful if you need to authenticate users with windows domain notation `myuser@ad.mydomain.com`. Using this method however limits the info you can get from the directory because we are unable to get the user's DN to look up their user info. You will only know if their credentials are correct or not. Defaults to `False`.
`LDAP_ALWAYS_SEARCH_BIND`	Specifies whether or not the library should perform direct binds. When the RDN attribute is the same as the login attribute, a direct bind will be performed automatically. However if the user is contained within a sub container of the `LDAP_USER_DN`, authentication will fail. Set this `True` to never perform a direct bind and instead perform a search to look up a user's DN. Defaults to `False`.
`LDAP_BIND_USER_DN`	Specifies the dn of the user to perform search requests with. Defaults to `None`. If None, Anonymous connections are used.
`LDAP_BIND_USER_PASSWORD`	Specifies the password to bind `LDAP_BIND_USER_DN` with. Defaults to `None`
`LDAP_SEARCH_FOR_GROUPS`	Specifies whether or not groups should be searched for when getting user details. Defaults to `True`.
`LDAP_FAIL_AUTH_ON_MULTIPLE_FOUND`	Specifies whether or not to fail authentication if multiple users are found when performing a `bind_search`. Defaults to `False`
`LDAP_BASE_DN`	Specifies the base DN for searching. Defaults to `''`
`LDAP_USER_DN`	Specifies the user DN for searching. Prepended to the base DN to limit the scope when searching for users. Defaults to `''`
`LDAP_GROUP_DN`	Specifies the group DN for searching. Prepended to the base DN to limit the scope when searching for groups. Defaults to `''`
`LDAP_BIND_AUTHENTICATION_TYPE`	Specifies the LDAP bind type to use when binding to LDAP. Defaults to `'AUTH_SIMPLE'`

Filters/Searching

`LDAP_USER_SEARCH_SCOPE`	Specifies what scope to search in when searching for a specific user. Defaults to `'LEVEL'`.
`LDAP_USER_OBJECT_FILTER`	Specifies what object filter to apply when searching for users. Defaults to `'(objectclass=inetorgperson)'`
`LDAP_USER_LOGIN_ATTR`	Declares what ldap attribute corresponds to the username passed to any login method when performing a bind. Defaults to `'uid'`
`LDAP_USER_RDN_ATTR`	Specifies the RDN attribute used in the directory. Defaults to `'uid'`
`LDAP_GET_USER_ATTRIBUTES`	Specifies which LDAP attributes to get when searching LDAP for a user/users. Defaults to `ldap3.ALL_ATTRIBUTES`
`LDAP_GROUP_SEARCH_SCOPE`	Specifies what scope to search in when searching for a specific group. Defaults to `'LEVEL'`.
`LDAP_GROUP_OBJECT_FILTER`	Specifies what object filter to apply when searching for groups. Defaults to `'(objectclass=group)'`
`LDAP_GROUP_MEMBERS_ATTR`	Specifies the LDAP attribute where group members are declared. Defaults to `'uniqueMember'`
`LDAP_GET_GROUP_ATTRIBUTES`	Specifies which LDAP attributes to get when searching LDAP for a group/groups. Defaults to `ldap3.ALL_ATTRIBUTES`

Quick Start

Install the Package

```
$ pip install flask-ldap3-login
```

Basic Application

This is a basic application which uses Flask-Login to handle user sessions. The application stores the users in the dictionary `users`.

```
from flask import Flask, url_for
from flask_ldap3_login import LDAP3LoginManager
from flask_login import LoginManager, login_user, UserMixin, current_user
from flask import render_template_string, redirect
from flask_ldap3_login.forms import LDAPLoginForm

app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret'
app.config['DEBUG'] = True

# Setup LDAP Configuration Variables. Change these to your own settings.
# All configuration directives can be found in the documentation.

# Hostname of your LDAP Server
app.config['LDAP_HOST'] = 'ad.mydomain.com'
```

```
# Base DN of your directory
app.config['LDAP_BASE_DN'] = 'dc=mydomain,dc=com'

# Users DN to be prepended to the Base DN
app.config['LDAP_USER_DN'] = 'ou=users'

# Groups DN to be prepended to the Base DN
app.config['LDAP_GROUP_DN'] = 'ou=groups'

# The RDN attribute for your user schema on LDAP
app.config['LDAP_USER_RDN_ATTR'] = 'cn'

# The Attribute you want users to authenticate to LDAP with.
app.config['LDAP_USER_LOGIN_ATTR'] = 'mail'

# The Username to bind to LDAP with
app.config['LDAP_BIND_USER_DN'] = None

# The Password to bind to LDAP with
app.config['LDAP_BIND_USER_PASSWORD'] = None

login_manager = LoginManager(app)               # Setup a Flask-Login Manager
ldap_manager = LDAP3LoginManager(app)           # Setup a LDAP3 Login Manager.

# Create a dictionary to store the users in when they authenticate
# This example stores users in memory.
users = {}


# Declare an Object Model for the user, and make it comply with the
# flask-login UserMixin mixin.
class User(UserMixin):
    def __init__(self, dn, username, data):
        self.dn = dn
        self.username = username
        self.data = data

    def __repr__(self):
        return self.dn

    def get_id(self):
        return self.dn


# Declare a User Loader for Flask-Login.
# Simply returns the User if it exists in our 'database', otherwise
# returns None.
@login_manager.user_loader
def load_user(id):
    if id in users:
        return users[id]
    return None


# Declare The User Saver for Flask-Ldap3-Login
# This method is called whenever a LDAPLoginForm() successfully validates.
# Here you have to save the user, and return it so it can be used in the
# login controller.
```

```
@ldap_manager.save_user
def save_user(dn, username, data, memberships):
    user = User(dn, username, data)
    users[dn] = user
    return user


# Declare some routes for usage to show the authentication process.
@app.route('/')
def home():
    # Redirect users who are not logged in.
    if not current_user or current_user.is_anonymous:
        return redirect(url_for('login'))

    # User is logged in, so show them a page with their cn and dn.
    template = """
    <h1>Welcome: {{ current_user.data.cn }}</h1>
    <h2>{{ current_user.dn }}</h2>
    """

    return render_template_string(template)


@app.route('/manual_login')
def manual_login():
    # Instead of using the form, you can alternatively authenticate
    # using the authenticate method.
    # This WILL NOT fire the save_user() callback defined above.
    # You are responsible for saving your users.
    app.ldap3_login_manager.authenticate('username', 'password')


@app.route('/login', methods=['GET', 'POST'])
def login():
    template = """
    {{ get_flashed_messages() }}
    {{ form.errors }}
    <form method="POST">
        <label>Username{{ form.username() }}</label>
        <label>Password{{ form.password() }}</label>
        {{ form.submit() }}
        {{ form.hidden_tag() }}
    </form>
    """

    # Instantiate a LDAPLoginForm which has a validator to check if the user
    # exists in LDAP.
    form = LDAPLoginForm()

    if form.validate_on_submit():
        # Successfully logged in, We can now access the saved user object
        # via form.user.
        login_user(form.user)  # Tell flask-login to log them in.
        return redirect('/')  # Send them home

    return render_template_string(template, form=form)
```

```
if __name__ == '__main__':
    app.run()
```

Basic Scripting Usage (Without a Flask App)

This is an example for if you wish to simply use the module, maybe for testing or for use in some other environment.

```
from flask_ldap3_login import LDAP3LoginManager

config = dict()

# Setup LDAP Configuration Variables. Change these to your own settings.
# All configuration directives can be found in the documentation.

# Hostname of your LDAP Server
config['LDAP_HOST'] = 'ad.mydomain.com'

# Base DN of your directory
config['LDAP_BASE_DN'] = 'dc=mydomain,dc=com'

# Users DN to be prepended to the Base DN
config['LDAP_USER_DN'] = 'ou=users'

# Groups DN to be prepended to the Base DN
config['LDAP_GROUP_DN'] = 'ou=groups'


# The RDN attribute for your user schema on LDAP
config['LDAP_USER_RDN_ATTR'] = 'cn'

# The Attribute you want users to authenticate to LDAP with.
config['LDAP_USER_LOGIN_ATTR'] = 'mail'

# The Username to bind to LDAP with
config['LDAP_BIND_USER_DN'] = None

# The Password to bind to LDAP with
config['LDAP_BIND_USER_PASSWORD'] = None

# Setup a LDAP3 Login Manager.
ldap_manager = LDAP3LoginManager()

# Init the mamager with the config since we aren't using an app
ldap_manager.init_config(config)

# Check if the credentials are correct
response = ldap_manager.authenticate('username', 'password')
print(response.status)
```

Custom TLS Context

This is an example that shows how to initialize a custom TLS context for securing communication between the module and a secure LDAP (LDAPS server.

```
from flask_ldap3_login import LDAP3LoginManager

config = dict()

# Setup LDAP Configuration Variables. Change these to your own settings.
# All configuration directives can be found in the documentation.

# Hostname of your LDAP Server
config['LDAP_HOST'] = 'ad.mydomain.com'

# Base DN of your directory
config['LDAP_BASE_DN'] = 'dc=mydomain,dc=com'

# Users DN to be prepended to the Base DN
config['LDAP_USER_DN'] = 'ou=users'

# Groups DN to be prepended to the Base DN
config['LDAP_GROUP_DN'] = 'ou=groups'


# The RDN attribute for your user schema on LDAP
config['LDAP_USER_RDN_ATTR'] = 'cn'

# The Attribute you want users to authenticate to LDAP with.
config['LDAP_USER_LOGIN_ATTR'] = 'mail'

# The Username to bind to LDAP with
config['LDAP_BIND_USER_DN'] = None

# The Password to bind to LDAP with
config['LDAP_BIND_USER_PASSWORD'] = None

# Setup a LDAP3 Login Manager.
ldap_manager = LDAP3LoginManager()

# Init the mamager with the config since we aren't using an app
ldap_manager.init_config(config)

# Check if the credentials are correct
response = ldap_manager.authenticate('username', 'password')
print(response.status)
```

API

Core

class `flask_ldap3_login.`**`AuthenticationResponse`**(*status=<AuthenticationResponseStatus.fail: 1>, user_info=None, user_id=None, user_dn=None, user_groups=[]*)

A response object when authenticating. Lets us pass status codes around and also user data.

Args: status (AuthenticationResponseStatus): The status of the result. user_info (dict): User info dictionary obtained from LDAP. user_id (str): User id used to authenticate to LDAP with. user_dn (str): User DN found from LDAP. user_groups (list): A list containing a dicts of group info.

class `flask_ldap3_login.`**`AuthenticationResponseStatus`**

An enumeration.

class `flask_ldap3_login.`**`LDAP3LoginManager`**(*app=None*)

Initialise a LDAP3LoginManager. If app is passed, init_app is called within this call.

Args: app (flask.Flask): The flask app to initialise with

`add_server`(*hostname*, *port*, *use_ssl*, *tls_ctx=None*)

Add an additional server to the server pool and return the freshly created server.

Args: hostname (str): Hostname of the server port (int): Port of the server use_ssl (bool): True if SSL is to be used when connecting. tls_ctx (ldap3.Tls): An optional TLS context object to use

when connecting.

Returns: ldap3.Server: The freshly created server object.

`authenticate`(*username*, *password*)

An abstracted authentication method. Decides whether to perform a direct bind or a search bind based upon the login attribute configured in the config.

Args: username (str): Username of the user to bind password (str): User's password to bind with.

Returns: AuthenticationResponse

`authenticate_direct_bind`(*username*, *password*)

Performs a direct bind. We can do this since the RDN is the same as the login attribute. Hence we just string together a dn to find this user with.

Args:

username (str): Username of the user to bind (the field specified as LDAP_BIND_RDN_ATTR)

password (str): User's password to bind with.

Returns: AuthenticationResponse

`authenticate_direct_credentials`(*username*, *password*)

Performs a direct bind, however using direct credentials. Can be used if interfacing with an Active Directory domain controller which authenticates using username@domain.com directly.

Performing this kind of lookup limits the information we can get from ldap. Instead we can only deduce whether or not their bind was successful. Do not use this method if you require more user info.

Args:

username (str): username for the user to bind with. LOGIN_SUFFIX will be appended.

password (str): User's password to bind with.

Returns: AuthenticationResponse

`authenticate_search_bind`(*username*, *password*)

Performs a search bind to authenticate a user. This is required when a the login attribute is not the same as the RDN, since we cannot string together their DN on the fly, instead we have to find it in the LDAP, then attempt to bind with their credentials.

Args:

username (str): Username of the user to bind (the field specified as LDAP_BIND_LOGIN_ATTR)

password (str): User's password to bind with when we find their dn.

Returns: AuthenticationResponse

compiled_sub_dn(*prepend*)

Returns: str: A DN with the DN Base appended to the end.

Args: prepend (str): The dn to prepend to the base.

connection

Convenience property for externally accessing an authenticated connection to the server. This connection is automatically handled by the appcontext, so you do not have to perform an unbind.

Returns: ldap3.Connection: A bound ldap3.Connection

Raises:

ldap3.core.exceptions.LDAPException: Since this method is performing a bind on behalf of the caller. You should handle this case occuring, such as invalid service credentials.

destroy_connection(*connection*)

Destroys a connection. Removes the connection from the appcontext, and unbinds it.

Args: connection (ldap3.Connection): The connnection to destroy

full_group_search_dn

Returns a the base search DN with the group search DN prepended.

Returns: str: Full group search dn

full_user_search_dn

Returns a the base search DN with the user search DN prepended.

Returns: str: Full user search dn

get_group_info(*dn*, *_connection=None*)

Gets info about a group specified at dn.

Args: dn (str): The dn of the group to find _connection (ldap3.Connection): A connection object to use when

searching. If not given, a temporary connection will be created, and destroyed after use.

Returns: dict: A dictionary of the group info from LDAP

get_object(*dn*, *filter*, *attributes*, *_connection=None*)

Gets an object at the specified dn and returns it.

Args: dn (str): The dn of the object to find. filter (str): The LDAP syntax search filter. attributes (list): A list of LDAP attributes to get when searching. _connection (ldap3.Connection): A connection object to use when

searching. If not given, a temporary connection will be created, and destroyed after use.

Returns: dict: A dictionary of the object info from LDAP

get_user_groups(*dn*, *group_search_dn=None*, *_connection=None*)

Gets a list of groups a user at dn is a member of

Args: dn (str): The dn of the user to find memberships for. _connection (ldap3.Connection): A connection object to use when

searching. If not given, a temporary connection will be created, and destroyed after use.

group_search_dn (str): The search dn for groups. Defaults to `'{LDAP_GROUP_DN},{LDAP_BASE_DN}'`.

Returns: list: A list of LDAP groups the user is a member of.

get_user_info(*dn*, *_connection=None*)

Gets info about a user specified at dn.

Args: dn (str): The dn of the user to find _connection (ldap3.Connection): A connection object to use when

searching. If not given, a temporary connection will be created, and destroyed after use.

Returns: dict: A dictionary of the user info from LDAP

get_user_info_for_username(*username*, *_connection=None*)

Gets info about a user at a specified username by searching the Users DN. Username attribute is the same as specified as LDAP_USER_LOGIN_ATTR.

Args: username (str): Username of the user to search for. _connection (ldap3.Connection): A connection object to use when

searching. If not given, a temporary connection will be created, and destroyed after use.

Returns: dict: A dictionary of the user info from LDAP

init_app(*app*)

Configures this extension with the given app. This registers an `teardown_appcontext` call, and attaches this `LDAP3LoginManager` to it as `app.ldap3_login_manager`.

Args: app (flask.Flask): The flask app to initialise with

init_config(*config*)

Configures this extension with a given configuration dictionary. This allows use of this extension without a flask app.

Args: config (dict): A dictionary with configuration keys

make_connection(*bind_user=None*, *bind_password=None*, ***kwargs*)

Make a connection to the LDAP Directory.

Args:

bind_user (str): User to bind with. If *None*, AUTH_ANONYMOUS is used, otherwise authentication specified with config['LDAP_BIND_AUTHENTICATION_TYPE'] is used.

bind_password (str): Password to bind to the directory with **kwargs (dict): Additional arguments to pass to the

`ldap3.Connection`

Returns:

ldap3.Connection: An unbound ldap3.Connection. You should handle exceptions upon bind if you use this internal method.

save_user(*callback*)

This sets the callback for saving a user that has been looked up from from ldap.

The function you set should take a user dn (unicode), username (unicode) and userdata (dict), and memberships (list).

```
@ldap3_manager.save_user
def save_user(dn, username, userdata, memberships):
    return User(username=username, data=userdata)
```

Your callback function MUST return the user object in your ORM (or similar). as this is used within the LoginForm and placed at `form.user`

Args:

callback (function): The function to be used as the save user callback.

teardown(*exception*)

Cleanup after a request. Close any open connections.

Forms

class flask_ldap3_login.forms.**LDAPLoginForm**(*formdata=<object object>*, ***kwargs*)

A basic loginform which can be subclassed by your application. Upon validation, the form will check against ldap for a valid username/password combination.

Once validiated will have a *form.user* object that contains a user object.

validate(**args*, ***kwargs*)

Validates the form by calling *validate* on each field, passing any extra *Form.validate_<fieldname>* validators to the field validator.

also calls *validate_ldap*

Python Module Index

f

flask_ldap3_login, 9
flask_ldap3_login.forms, 13

Index

A

add_server() (flask_ldap3_login.LDAP3LoginManager method), 10
authenticate() (flask_ldap3_login.LDAP3LoginManager method), 10
authenticate_direct_bind() (flask_ldap3_login.LDAP3LoginManager method), 10
authenticate_direct_credentials() (flask_ldap3_login.LDAP3LoginManager method), 10
authenticate_search_bind() (flask_ldap3_login.LDAP3LoginManager method), 10
AuthenticationResponse (class in flask_ldap3_login), 9
AuthenticationResponseStatus (class in flask_ldap3_login), 9

C

compiled_sub_dn() (flask_ldap3_login.LDAP3LoginManager method), 10
connection (flask_ldap3_login.LDAP3LoginManager attribute), 11

D

destroy_connection() (flask_ldap3_login.LDAP3LoginManager method), 11

F

flask_ldap3_login (module), 9
flask_ldap3_login.forms (module), 13
full_group_search_dn (flask_ldap3_login.LDAP3LoginManager attribute), 11
full_user_search_dn (flask_ldap3_login.LDAP3LoginManager attribute), 11

G

get_group_info() (flask_ldap3_login.LDAP3LoginManager method), 11
get_object() (flask_ldap3_login.LDAP3LoginManager method), 11
get_user_groups() (flask_ldap3_login.LDAP3LoginManager method), 11
get_user_info() (flask_ldap3_login.LDAP3LoginManager method), 11
get_user_info_for_username() (flask_ldap3_login.LDAP3LoginManager method), 12

I

init_app() (flask_ldap3_login.LDAP3LoginManager method), 12
init_config() (flask_ldap3_login.LDAP3LoginManager method), 12

L

LDAP3LoginManager (class in flask_ldap3_login), 10
LDAPLoginForm (class in flask_ldap3_login.forms), 13

M

make_connection() (flask_ldap3_login.LDAP3LoginManager method), 12

S

save_user() (flask_ldap3_login.LDAP3LoginManager method), 12

T

teardown() (flask_ldap3_login.LDAP3LoginManager method), 13

V

validate() (flask_ldap3_login.forms.LDAPLoginForm method), 13