Access Control and Authorization

You can install Flask-Security and other required dependencies using pip:

```
pip install Flask-Security Flask-SQLAlchemy Flask
```

Create a Python file for your plugin, which we'll call `access_control.py`:

```
from flask import Flask, render_template, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
from flask_security import Security, SQLAlchemyUserDatastore, UserMixin, RoleMixin, login_required

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///your_database.db'
app.config['SECRET_KEY'] = 'your_secret_key'

db = SQLAlchemy(app)

# Define User and Role models
class Role(db.Model, RoleMixin):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(80), unique=True)
    description = db.Column(db.String(255))

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True)
    email = db.Column(db.String(120), unique=True)
    password = db.Column(db.String(255))
    active = db.Column(db.Boolean)
    roles = db.relationship('Role', secondary='roles_users',
                            backref=db.backref('users', lazy='dynamic'))

roles_users = db.Table('roles_users',
                       db.Column('user_id', db.Integer(), db.ForeignKey('user.id')),
                       db.Column('role_id', db.Integer(), db.ForeignKey('role.id')))

# Create the database tables
db.create_all()

# Initialize Flask-Security
user_datastore = SQLAlchemyUserDatastore(db, User, Role)
security = Security(app, user_datastore)

# Define routes and permissions
@app.route('/')
```

```
@login_required
def home():
    # Only authenticated users with the 'user' role can access this route
    return 'Welcome to the protected area!'

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

1. In this code:
    - We import Flask and the required extensions (Flask-Security, Flask-SQLAlchemy).
    - We define the database models for User and Role.
    - We create the database tables using SQLAlchemy.
    - We initialize Flask-Security with a database-backed user data store and set up permissions for the `home` route using the `@login_required` decorator.

2. Customize the code:
    - Modify the `User` and `Role` models to include additional fields and properties as needed.
    - Define more routes and permissions based on your application's requirements.

3. Run the application:
   Execute the script, and your Flask web application with access control and authorization will be up and running.