Index

Securing Django REST APIs with Auth0 OAuth

This guide explains how to secure Django REST Framework (DRF) APIs using Auth0 OAuth, including JWT validation, custom authentication, and DRF permission enforcement.

⚠️ Note: This setup is intended for DRF-based API endpoints, not Django template-based views. For traditional Django authentication, refer to **Auth0 Django guide**

Configure Auth0

Configure api in auth0

Steps:

1. Create a new API in Auth0:
    - Name: AssignMate API
    - Identifier (Audience):*http://localhost:8000/users*
2. Create an Application (machine-to-machine or SPA).
    - Note Client ID and Client Secret
    - Associate your application with the API.

Purpose: Only tokens issued for your API audience will be valid. Auth0 acts as the trusted authentication server.

Install Dependencies

```
pip install authlib python-dotenv djangorestframework
```

authlib → JWT parsing and validation

python-dotenv → Load environment variables

djangorestframework → API framework

Django Settings *settings.py*

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'users.authentication.Auth0Authentication',
        'rest_framework.authentication.SessionAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
}

AUTH0_DOMAIN = os.environ.get("AUTH0_DOMAIN")
AUTH0_AUDIENCE = os.environ.get("AUTH0_AUDIENCE")
```

Purpose: Configures DRF to use custom authentication and ensures only authenticated users can access endpoints.

JWT Validator *users/validator.py*

```
import json
from urllib.request import urlopen
from authlib.oauth2.rfc7523 import JWTBearerTokenValidator
from authlib.jose.rfc7517 import JsonWebKey

class Auth0JWTBearerTokenValidator(JWTBearerTokenValidator):
    def __init__(self, domain, audience):
        self.issuer = f"https://{domain}/"
        self.audience = audience
        jwks_url = f"https://{domain}/.well-known/jwks.json"
        with urlopen(jwks_url) as response:
            jwks_data = json.load(response)
        self.jwks = JsonWebKey.import_key_set(jwks_data)
        super().__init__(self.jwks)
        self.claims_options = {
            "iss": {"essential": True, "value": self.issuer},
            "aud": {"essential": True, "value": self.audience},
            "exp": {"essential": True},
        }
```

Purpose:

- Fetch Auth0 JWKS (public keys)
- Validate JWT signature and essential claims (iss, aud, exp)

Custom Authentication Class *users/authentication.py*

```
import os
from rest_framework.authentication import BaseAuthentication
from rest_framework.exceptions import AuthenticationFailed
from .validator import Auth0JWTBearerTokenValidator

class Auth0Authentication(BaseAuthentication):
    def __init__(self):
        self.validator = Auth0JWTBearerTokenValidator(
            os.environ.get("AUTH0_DOMAIN"),
            os.environ.get("AUTH0_AUDIENCE"),
        )

    def authenticate(self, request):
        auth = request.headers.get("Authorization")
        if not auth or not auth.startswith("Bearer "):
            return None
        token = auth.split(" ")[1]
        try:
            claims = self.validator.authenticate_token(token)
            if not claims:
                raise AuthenticationFailed("Invalid token")

            # Map Auth0 'sub' claim to Django User
            from django.contrib.auth import get_user_model
            User = get_user_model()
            user, _ = User.objects.get_or_create(username=claims['sub'])
            return (user, token)
        except Exception as e:
            raise AuthenticationFailed(f"Token validation failed: {e}")
```

Purpose:

- Extracts JWT from `Authorization` header
- Validates token with Auth0 JWKS
- Returns Django `User` object to satisfy DRF permission checks

Protected DRF Viewsets *users/views.py*

```
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated
from .serializers import UserSerializer
from django.contrib.auth import get_user_model
from .authentication import Auth0Authentication

User = get_user_model()

class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    authentication_classes = [Auth0Authentication]
    permission_classes = [IsAuthenticated]
```

Purpose:

- Ensures all requests to this viewset are authenticated using Auth0 JWT
- `IsAuthenticated` checks the `user` object returned by the authentication class

---

How the flow works

1. Client requests token from Auth0 using your API audience.
2. Token is sent in Authorization: Bearer *token* header to Django API.
3. DRF calls Auth0Authentication:
    - Validates token signature
    - Confirms `aud` claim matches your API
    - Returns a Django `User` object
4. DRF permission classes `IsAuthenticated` allow or deny access based on the returned `user`.

Purpose: Only authenticated users from Auth0 can access the API, and DRF framework contracts are respected.

Key lessons

- Audience `aud` claim ensures tokens are intended for your API.
- DRF expects `user, token` from `authenticate()`; permissions rely on the `user` object.

✅ Outcome: DRF APIs are secured with Auth0 OAuth; only authenticated users can access endpoints. Next step could be implementing scope-based authorization for role-specific access.