

Securing Application Access

Microsoft Entra ID • OAuth 2.0 • JWT Inspection • Microsoft Graph

NOCTURNE SYSTEMS | IAM LAB 001 | Technical implementation and verification

Portfolio case study — sanitized. The screenshots are intended to be inserted after capture and redaction.

1. Project overview

In the Wayne Enterprise Systems personal lab, I configured a single-tenant Microsoft Entra ID app registration and built a Python public-client application using Microsoft Authentication Library (MSAL). The application authenticated a test user, obtained a delegated Microsoft Graph access token, and retrieved the signed-in user's profile. I then inspected selected access-token claims locally and tested an API operation that the delegated permissions did not authorize.

Key outcome: GET /me returned HTTP 200 OK; GET /users returned HTTP 403 Forbidden with Authorization_RequestDenied and an insufficient-privileges message.

2. Tools and scope

Identity platform	Microsoft Entra ID (Wayne Enterprise Systems test tenant; Microsoft 365 Business Premium)
App model	Single-tenant public/native client, interactive user sign-in
Development	Python 3, MSAL, Requests, PowerShell, Windows virtual environment
API	Microsoft Graph v1.0
Permission	Delegated User.Read (plus OIDC scopes returned during sign-in)
Out of scope	Production deployment, application-only tokens, consent phishing simulation, cryptographic token validation

3. Architecture and authorization

Local Python app → Microsoft Entra ID (interactive sign-in and consent) → delegated Graph access token → Microsoft Graph API. The app registration specifies the client's identity and sign-in configuration; its service principal represents the app within the tenant. Graph evaluates the presented token and applicable authorization rules for each endpoint.

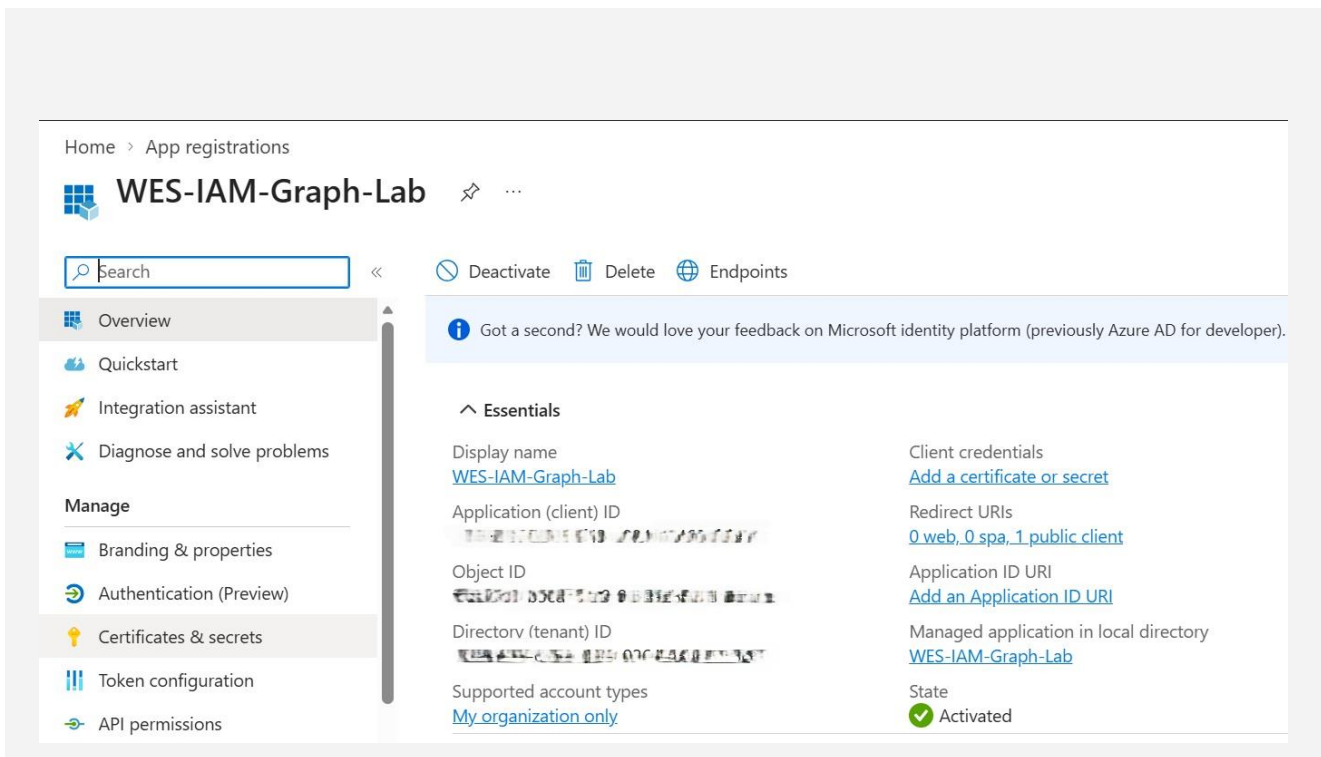
4. Implementation steps

Step 1 — Register the application

Created WES-IAM-Graph-Lab as a single-tenant app registration. Configured the Mobile and desktop applications platform with http://localhost as the redirect URI. Used a public-client interactive browser flow; no client secret was needed.

SCREENSHOT 1 — App registration overview

Insert directly below this cue: Entra admin center → Entra ID → App registrations → WES-IAM-Graph-Lab → Overview.
Show the application name and overview context.



Before publishing: Cover application ID, tenant ID, account name, domain, and browser personal details.

Step 2 — Configure delegated API permission

Selected Microsoft Graph → Delegated permissions → User.Read. The consent screen indicated profile-read and continued access requested as part of authentication; the lab used user-specific consent, not organization-wide consent.

IMPORTANT — Least privilege: no directory-wide read or write permission was granted for this experiment.

SCREENSHOT 2 — API permissions

Insert directly below this cue: Entra admin center → WES-IAM-Graph-Lab → API permissions. Show Microsoft Graph User.Read with Type = Delegated.

Search



- Overview
- Quickstart
- Integration assistant
- Diagnose and solve problems

Manage

- Branding & properties
- Authentication (Preview)
- Certificates & secrets
- Token configuration
- API permissions**
- Expose an API
- App roles

The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. [Learn more](#)

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

+ Add a permission ✓ Grant admin consent for Wayne Enterprise

API / Permissions name	Type	Description	Admin cons
Microsoft Graph (1)			
User.Read	Delegated	Sign in and read user profile	No

Before publishing: Hide tenant-specific navigation and identifying information; show only relevant permission rows.

Step 3 — Prepare and run the Python project

```
mkdir "$HOME\WES-IAM-Lab"
cd "$HOME\WES-IAM-Lab"
python -m venv .venv
.\.venv\Scripts\python.exe -m pip install msal requests
# Start after saving the script below:
.\.venv\Scripts\python.exe .\iam_lab.py
```

The code below is a publication-oriented, privacy-safe reconstruction of the successfully tested lab logic. It reads identifiers from environment variables instead of embedding tenant-specific values. Test this exact cleaned-up version before labeling the published code independently verified.

Python script — iam_lab.py

```
import os
import base64
import json
from datetime import datetime, timezone

import msal
import requests

CLIENT_ID = os.environ["WES_CLIENT_ID"]
TENANT_ID = os.environ["WES_TENANT_ID"]
AUTHORITY = f"https://login.microsoftonline.com/{TENANT_ID}"

app = msal.PublicClientApplication(CLIENT_ID, authority=AUTHORITY)
result = app.acquire_token_interactive(scopes=["User.Read"], port=8400)
if "access_token" not in result:
    raise RuntimeError("Interactive authentication failed: " +
                      str(result.get("error", "unknown")))

token = result["access_token"]
headers = {"Authorization": f"Bearer {token}"}
```

```
def graph_get(path):
    return requests.get("https://graph.microsoft.com/v1.0" + path,
                        headers=headers, timeout=20)

profile_response = graph_get("/me")
print("GET /me:", profile_response.status_code)
if profile_response.ok:
    print("Profile retrieved successfully (details omitted).")

parts = token.split(".")
if len(parts) != 3:
    raise RuntimeError("Expected JWT-format access token")
payload = parts[1] + "=" * (-len(parts[1]) % 4)
claims = json.loads(base64.urlsafe_b64decode(payload))
print("Audience:", claims.get("aud"))
print("Scopes:", claims.get("scp"))
print("Roles:", claims.get("roles"))
exp = claims.get("exp")
if isinstance(exp, (int, float)):
    print("Expires (UTC):", datetime.fromtimestamp(
        exp, timezone.utc).isoformat())

users_response = graph_get("/users")
print("GET /users:", users_response.status_code)
if not users_response.ok:
    err = users_response.json().get("error", {})
    print("Graph error code:", err.get("code"))
    # Avoid logging raw response bodies or sensitive context.
```

For local testing in the current PowerShell session, first define \$env:WES_CLIENT_ID and \$env:WES_TENANT_ID with your own values. Never put passwords, client secrets, refresh tokens, or bearer access tokens in published code.

Step 4 — Verify authenticated API access

The delegated access token enabled GET https://graph.microsoft.com/v1.0/me and Microsoft Graph returned HTTP 200 OK along with the signed-in test account's profile. This shows that the request completed successfully for the authorized operation.

SCREENSHOT 3 — Successful authenticated Graph call

Insert directly below this cue: PowerShell after running iam_lab.py. Capture "Authentication successful", "Access token acquired", and "HTTP Status: 200", or the corresponding success output from the cleaned-up version.

```
PS C:\Users\domin\WES-IAM-Lab> .\venv\Scripts\python.exe .\iam_lab.py
Authentication successful!
Access token acquired.
HTTP Status: 200
Display Name: Christopher Frye
User Principal Name: brucewayne@WayneEnterprise921.onmicrosoft.com
```

Before publishing: Black out display name, UPN/email, machine username, and any other personal/tenant detail. Never include a token.

Step 5 — Inspect JWT claims and test insufficient authorization

Decoded the JWT payload locally for learning. The observed audience was Microsoft Graph’s app ID (00000003-0000-0000-c000-000000000000); scopes included openid, profile, User.Read, and email; the roles claim was absent. The exp value represented token expiration. Decoding is not cryptographic signature validation.

The same delegated token was sent to GET https://graph.microsoft.com/v1.0/users. Graph returned HTTP 403 Forbidden with error code Authorization_RequestDenied and an insufficient-privileges message because the token did not carry permission to list directory users.

IMPORTANT — A successful sign-in or HTTP 200 from one API endpoint does not grant access to unrelated API resources.

SCREENSHOT 4 — Token scope + HTTP 403 authorization test

Insert directly below this cue: PowerShell showing the sanitized “Scopes (scp)” line and the “AUTHORIZATION TEST” block containing HTTP 403, Authorization_RequestDenied, and the insufficient-privileges error.

```
--- AUTHORIZATION TEST ---
HTTP Status: 403
Authorization denied (403 Forbidden).
Error code: Authorization_RequestDenied
Error message: Insufficient privileges to complete the operation.
PS C:\Users\domin\WES-IAM-Lab> |
```

Before publishing: Crop out the issuer URL/tenant ID, timestamps if desired, Windows username, profile data, and all bearer tokens.

5. Verified outcomes

Microsoft Entra authentication	Successful interactive sign-in; delegated access token acquired
Graph user profile query	GET /me — HTTP 200 OK
JWT payload inspection	Observed delegated User.Read and other OIDC scopes; roles absent
Directory-wide user-list attempt	GET /users — HTTP 403 Forbidden
Graph error	Authorization_RequestDenied; insufficient privileges

6. Security interpretation

This exercise demonstrated how application authentication and API authorization are separate checks. Delegated User.Read was sufficient for the signed-in user’s profile but not for a tenant-wide directory listing. The scope claim indicated delegated authorization; application-only access using the client-credentials flow was not implemented in this lab.

Publication safeguard: publish only sanitized screenshots and code with placeholders/environment variables. Do not include real access/refresh tokens, authorization redirect URLs, client secrets, private keys, tenant IDs, application IDs, or personal account information.

7. Lessons learned and next steps

I practiced registering an Entra application, configuring delegated Graph permissions, implementing interactive OAuth sign-in using MSAL, making authenticated REST requests, inspecting JWT claims, and interpreting HTTP 200 versus HTTP 403. A logical next project is a separately controlled application-only client-credentials lab to compare application roles and delegated scopes.

Documentation note: four screenshot cues in this document are intentionally blank. Insert your sanitized evidence at the matching locations; do not replace it with fabricated portal or terminal screenshots.