



Client/Server Single Sign On (SSO) with OpenID Connect

Bruce Armstrong, Integrated Data Services
October 2026

DISCLAIMER

This presentation was authored by volunteer(s) in the Appeon community. This is not a work for hire by Appeon. The views and opinions expressed in this presentation are those of the author(s).

Its contents are protected by US copyright law and may not be reproduced, distributed, transmitted, displayed, published or broadcast without the prior written permission of Appeon. All rights belong to their respective owners.

Any reference to third-party materials, including but not limited to Websites, content, services, or software, has not been reviewed or endorsed by Appeon. YOUR USE OF THIRD-PARTY MATERIALS SHALL BE AT YOUR OWN RISK.

Appeon makes no warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. Appeon assumes no responsibility for errors or omissions.

Session Agenda

- OAuth2 vs OpenID Connect (OIDC)
- Setting Up Authentik
- PowerBuilder application changes
- Demo

Presenter Profile



Bruce Armstrong



<https://www.linkedin.com/in/brucearmstrong/>

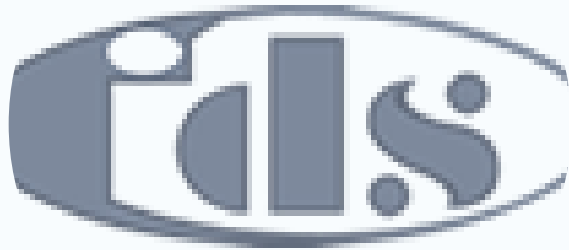
Key Skills

- PowerBuilder
- Oracle PL/SQL
- Java
- SonarQube
- C#
- C++
- AI
 - Claude Code
 - Codex
- Qlik Sense
- HTML5/JavaScript/TypeScript/Angular

Recent Projects

- [OpenSourcePFCLibraries](#) - The open-source PowerBuilder Foundation Class libraries

Company Profile



Integrated Data Services

- IDS has delivered innovative program and financial management information system solutions to the Federal Government for nearly three decades. Our flagship Enterprise Requirements Management product, CCaR™ is used by the DoD and Federal Agencies including the Air Force, Space Force, Army, Navy, USSOCOM, DSCA, DHA, JSF and DOE.

OAuth2 vs OIDC

- OAuth2 is an Authorization rather than an Authentication framework.
- OAuth2 relies on the identity server to perform authentication before it issues an Auth Token
- Open ID Connect (OIDC) is a protocol built on top of OAuth2 that does provide for both Authentication and Authorization

Scopes and claims

- The application asks for scopes; the identity provider answers with claims
 - Scopes: the categories of information the client requests (scope=openid profile)
 - Claims: facts about the user that the provider fills in (sub, name, email, groups)
- The application can't assert a claim: the provider reports the groups the user is in
- A scope decides which claims come back, not who may use the application

Who decides what

- Authentik (the identity provider)
 - Who the user is
 - Which groups the user belongs to (the groups claim)
 - Whether the user may use the application at all (group bindings)
- The application (PFC security service)
 - Which groups exist, and their priority
 - What each group may do with windows, controls and menu items
- One source of truth for each: group membership is never stored in PFC

Setting Up Authentik

Setting Up Authentik

- Install and configure Authentik via Docker Compose
- Log in and set the admin password
- Create a new OAuth2/OIDC application and provider
- Create groups and users, and assign them
- Restrict the application to those groups
- Test authentication and read the groups claim

Setting Up Authentik

Step 1: Install & Configure Authentik

- Download the latest docker-compose.yml from Authentik
 - wget <https://docs.goauthentik.io/compose.yml>
 - Invoke-WebRequest <https://docs.goauthentik.io/compose.yml> -OutFile "docker-compose.yml"
- Create an .env file with passwords in that same directory
 - PG_PASS=dGhpcy1pcy1hLXNlY3...[36-byte Base64 String]
 - AUTHENTIK_SECRET_KEY=dGhpcy1pcy1hLXZlcnktdG...[60-byte Base64 String]
 - AUTHENTIK_ERROR_REPORTING__ENABLED=true
- Optionally, add custom ports to the .env file (defaults are 9000 and 9443)
 - COMPOSE_PORT_HTTP=80
 - COMPOSE_PORT_HTTPS=443

Setting Up Authentik

Step 1 (cont.): Run Authentik via Docker Compose

- Optionally, configure the docker socket
- Optionally, configure email
- Run docker compose in that same directory
 - docker compose pull
 - docker compose up -d

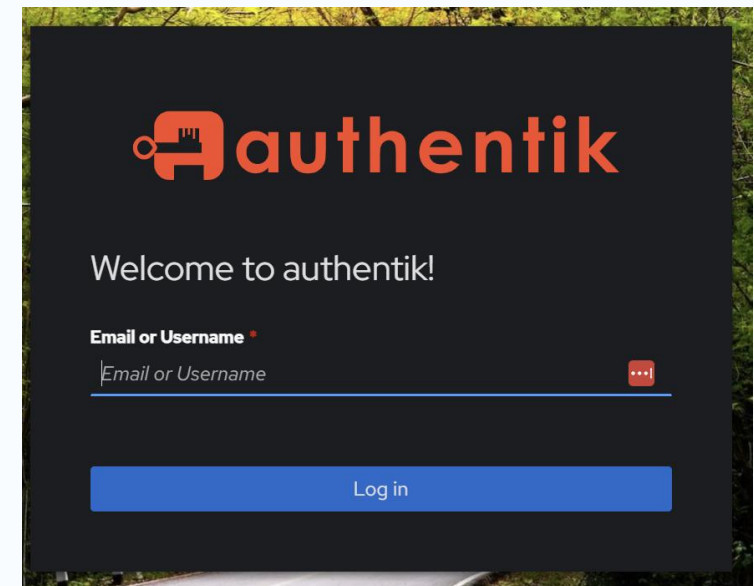
```
PS C:\Users\me\Downloads\docker\authentik> docker compose pull
[+] pull 43/43
✓Image docker.io/library/postgres:16-alpine Pulled
✓Image ghcr.io/goauthentik/server:2026.5.6 Pulled
PS C:\Users\me\Downloads\docker\authentik> |
```

```
PS C:\Users\me\Downloads\docker\authentik> docker compose up -d
[+] up 4/4
✓Network authentik_default Created
✓Container authentik-postgresql-1 Healthy
✓Container authentik-worker-1 Started
✓Container authentik-server-1 Started
PS C:\Users\me\Downloads\docker\authentik> |
```

Setting Up Authentik

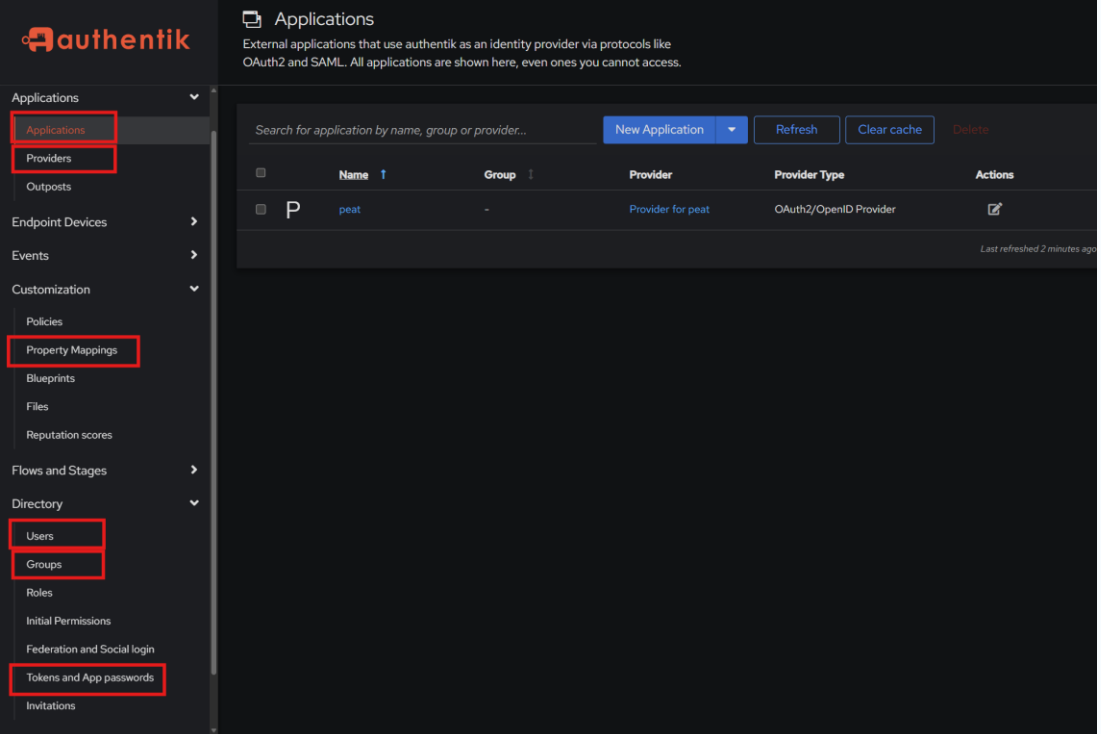
Step 2: Log In & Set the Admin Password

- Access authentic at <http://localhost:9000> and set the password for the “akadmin” user



Setting Up Authentik

Step 2 (cont.): View the Admin Dashboard



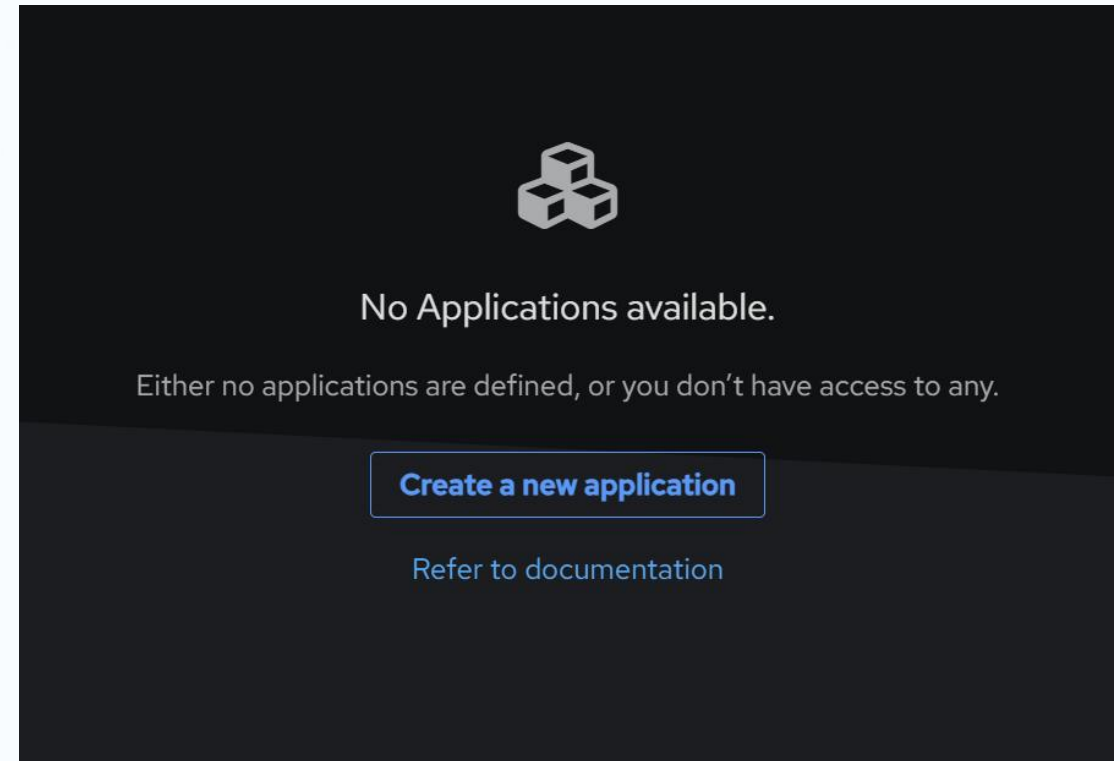
The screenshot displays the Authentik Admin Dashboard. On the left is a dark sidebar with a navigation menu. The 'Applications' menu item is highlighted with a red box. Other menu items include Providers, Outposts, Endpoint Devices, Events, Customization (with sub-items Policies, Property Mappings, Blueprints, Files, and Reputation scores), Flows and Stages, Directory (with sub-items Users, Groups, Roles, Initial Permissions, Federation and Social login, and Tokens and App passwords), and Invitations. The main content area is titled 'Applications' and includes a description: 'External applications that use authentik as an identity provider via protocols like OAuth2 and SAML. All applications are shown here, even ones you cannot access.' Below this is a search bar and buttons for 'New Application', 'Refresh', 'Clear cache', and 'Delete'. A table lists applications with columns for Name, Group, Provider, Provider Type, and Actions. One application named 'peat' is listed with a provider 'Provider for peat' and type 'OAuth2/OpenID Provider'. A 'Last refreshed 2 minutes ago' status is shown at the bottom right of the table.

Name	Group	Provider	Provider Type	Actions
peat	-	Provider for peat	OAuth2/OpenID Provider	[Edit]

Setting Up Authentik

Step 3: Create a New Application

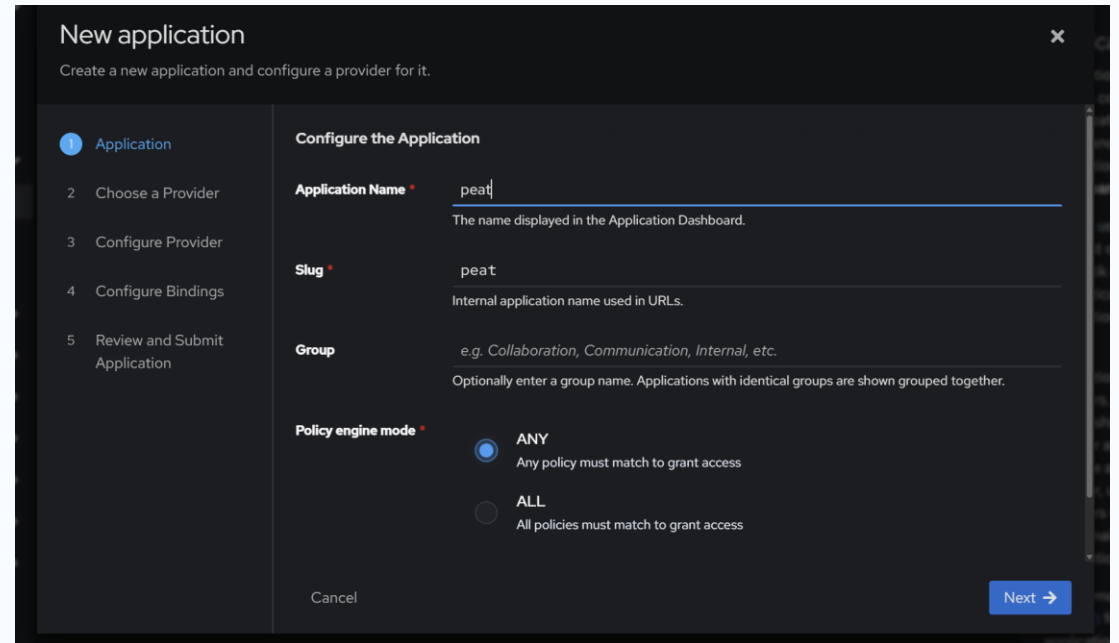
- Create a new application



Setting Up Authentik

Step 3 (cont.): Name the Application

- Provide the name and policy engine mode for the new application
- Set the policy engine mode to any: a user needs to pass only one of the group bindings (Step 5)



The screenshot shows the 'New application' configuration window in Authentik. The window has a dark theme and a sidebar on the left with five steps: 1. Application (selected), 2. Choose a Provider, 3. Configure Provider, 4. Configure Bindings, and 5. Review and Submit Application. The main area is titled 'Configure the Application' and contains the following fields:

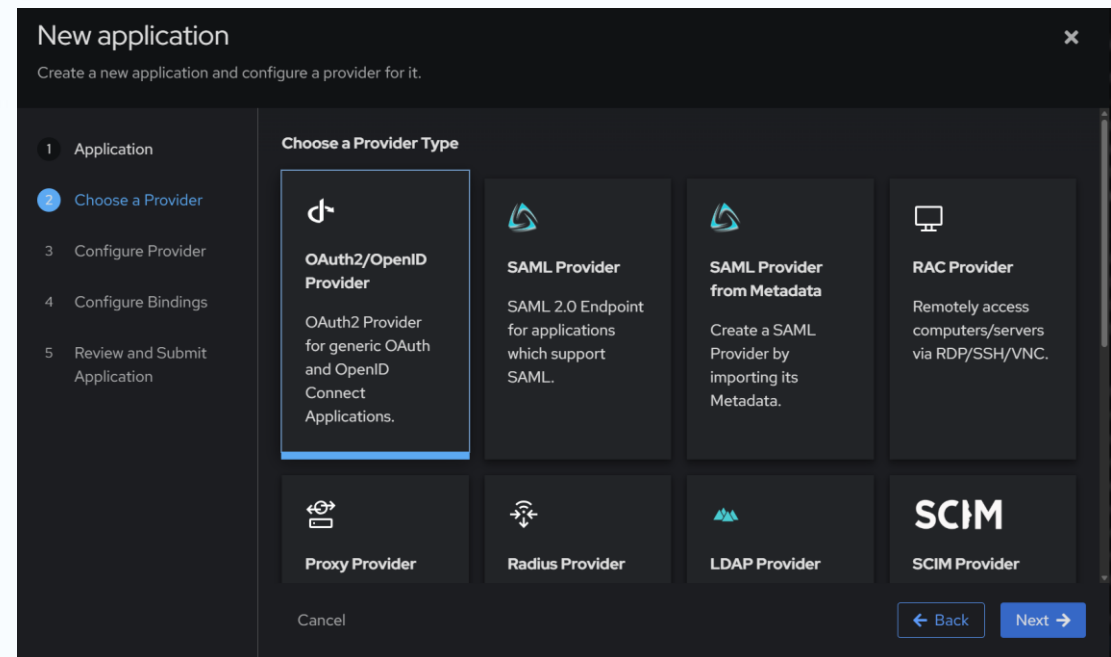
- Application Name ***: A text input field containing 'peat'. Below it, a small text label reads 'The name displayed in the Application Dashboard.'
- Slug ***: A text input field containing 'peat'. Below it, a small text label reads 'Internal application name used in URLs.'
- Group**: A text input field containing 'e.g. Collaboration, Communication, Internal, etc.'. Below it, a small text label reads 'Optionally enter a group name. Applications with identical groups are shown grouped together.'
- Policy engine mode ***: Two radio button options. The first is 'ANY' with the description 'Any policy must match to grant access' and is selected. The second is 'ALL' with the description 'All policies must match to grant access' and is unselected.

At the bottom of the main area, there are two buttons: 'Cancel' on the left and 'Next →' on the right.

Setting Up Authentik

Step 3 (cont.): Select an Authentication Provider

- Select an authentication provider for the new application



Setting Up Authentik

Step 3 (cont.): Configure the Provider

- Configure the provider
 - Set the Authorization Flow to implicit
 - Set the Client Type to Public (not Confidential as shown here).
 - If you set the client type to Confidential then you need to add Client Secret to the auth request
- Leave the scopes at the defaults (openid, email, profile): profile includes the groups claim

New application

Create a new application and configure a provider for it.

1 Application

Configure OAuth2 Provider

Provider Name * Provider for peat

Choose a Provider

Configure Provider

Authorization Flow * default-provider-authorization-implicit-consent (Authorize Application)

Flow used when authorizing this provider.

Client Type *

☒ Confidential

Confidential clients are capable of maintaining the confidentiality of their credentials such as client secrets

☐ Public

Public clients are incapable of maintaining the confidentiality and should use methods like PKCE.

Client ID * yp2EWP80FaRFqIIPnYYa4Pvawd4bRt6rqLjEKjh7

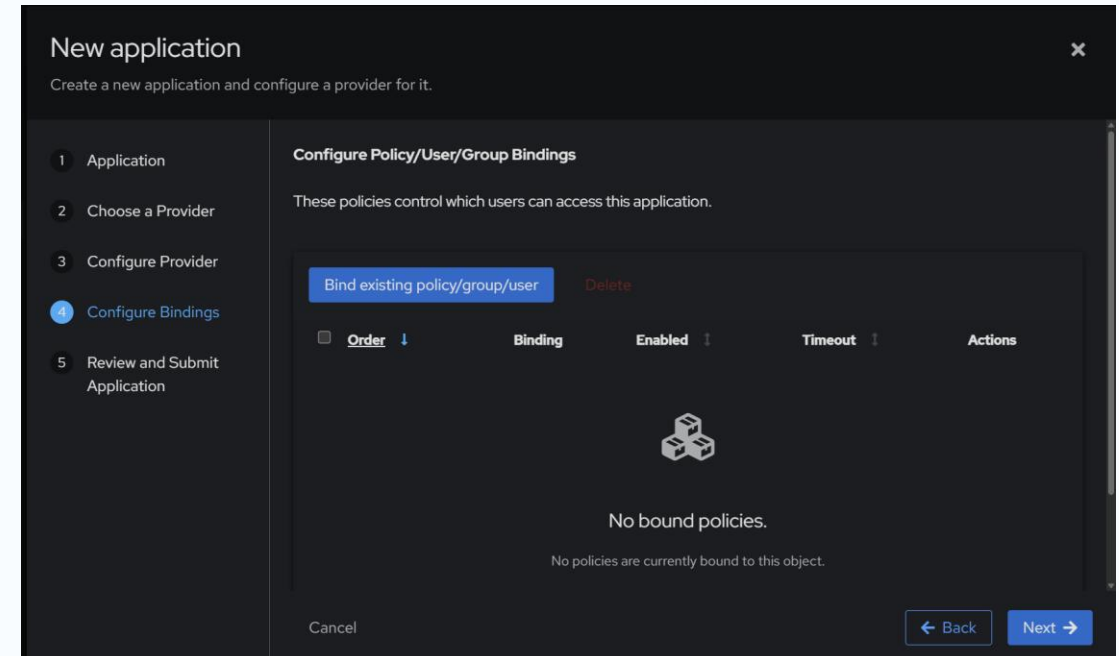
Cancel

Back Next

Setting Up Authentik

Step 3 (cont.): Skip Policy/User/Group Binding

- Skip the bindings for now: the groups don't exist yet. They are bound in Step 5



Setting Up Authentik

Step 3 (cont.): Finish the Application Wizard

- Finished

New application

Create a new application and configure a provider for it.

1 Application

2 Choose a Provider

3 Configure Provider

4 Configure Bindings

5 Review and Submit Application

Review the Application and Provider

Application Details

Application Name

peat

Group

-

Policy engine mode

ANY

Provider Details

Type	Name	Client Type
OAuth2	Provider for peat	Confidential

Client ID

Redirect URIs

Cancel

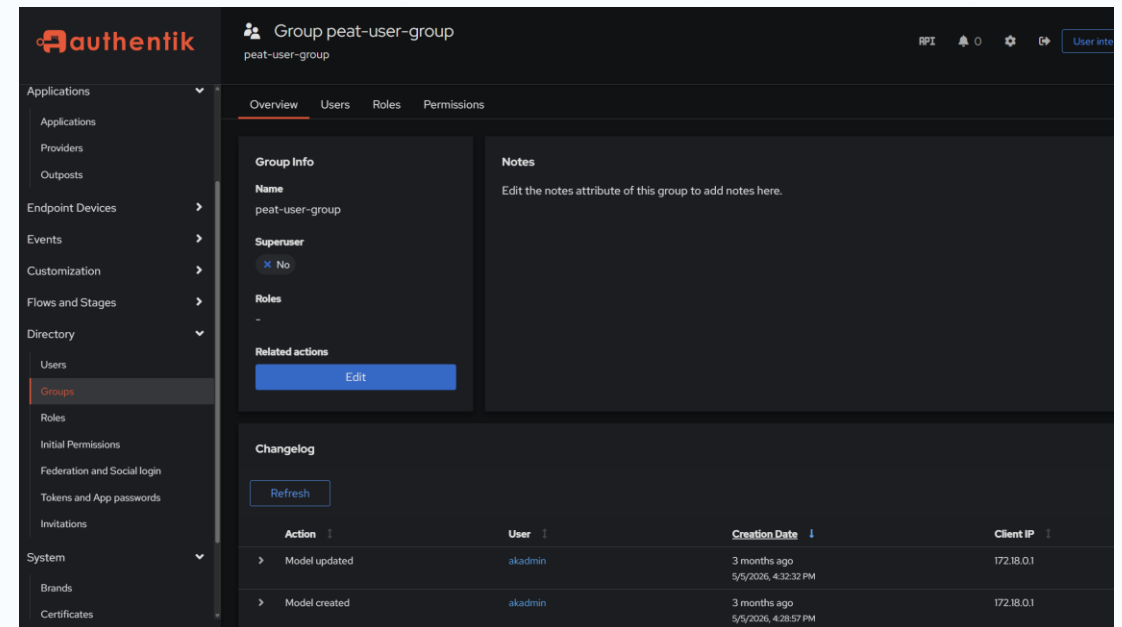
← Back

Create Application

Setting Up Authentik

Step 4: Create Groups

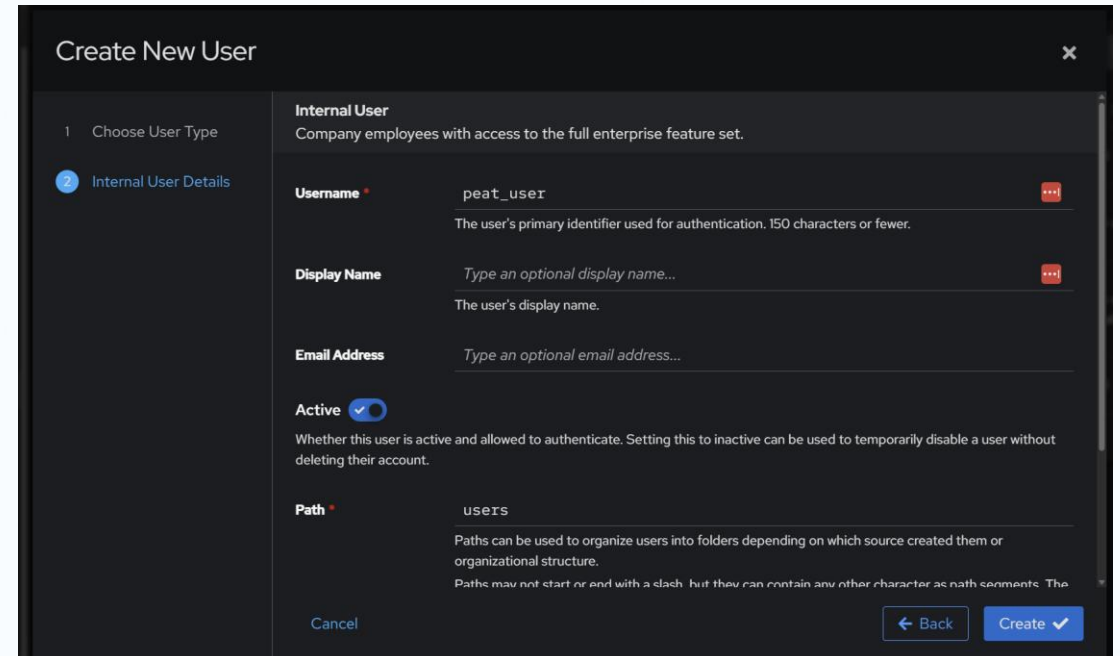
- Under Directory -> Groups, create two groups, one for admins and one for users



Setting Up Authentik

Step 4 (cont.): Create Users

- Under Directory -> Users, create two users, one a normal user and one an admin



The screenshot shows the 'Create New User' dialog box in Authentik. The dialog has a dark theme and a close button (X) in the top right corner. On the left, there is a sidebar with two steps: '1 Choose User Type' and '2 Internal User Details' (which is currently selected). The main area is titled 'Internal User' and contains the following fields and options:

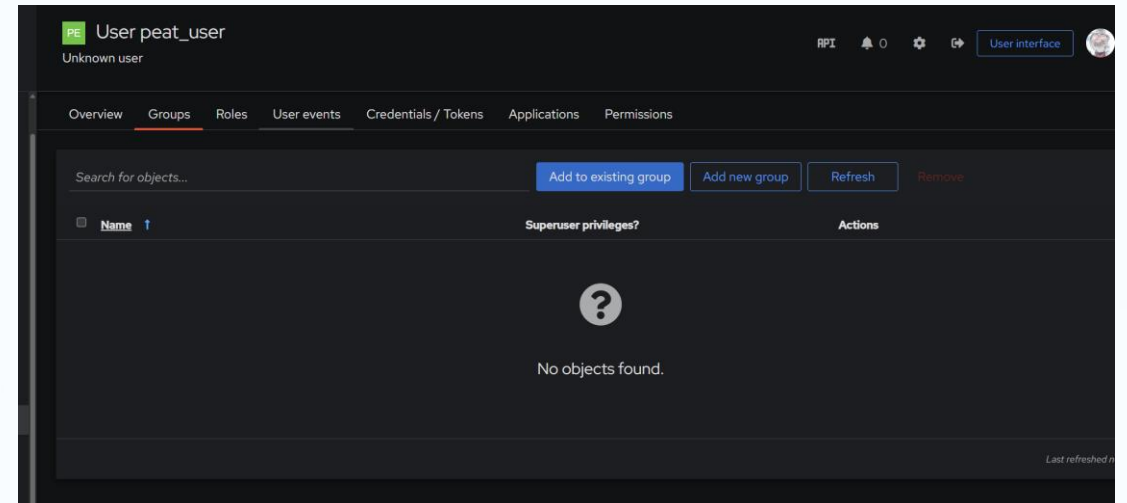
- Username ***: A text input field containing 'peat_user'. Below it, a description reads: 'The user's primary identifier used for authentication. 150 characters or fewer.'
- Display Name**: A text input field with a placeholder 'Type an optional display name...'. Below it, a description reads: 'The user's display name.'
- Email Address**: A text input field with a placeholder 'Type an optional email address...'.
- Active**: A toggle switch that is currently turned on (blue). Below it, a description reads: 'Whether this user is active and allowed to authenticate. Setting this to inactive can be used to temporarily disable a user without deleting their account.'
- Path ***: A text input field containing 'users'. Below it, a description reads: 'Paths can be used to organize users into folders depending on which source created them or organizational structure. Paths may not start or end with a slash, but they can contain any other character as path segments. The'.

At the bottom of the dialog, there are three buttons: 'Cancel', 'Back' (with a left arrow), and 'Create' (with a checkmark).

Setting Up Authentik

Step 4 (cont.): Assign Users to Groups

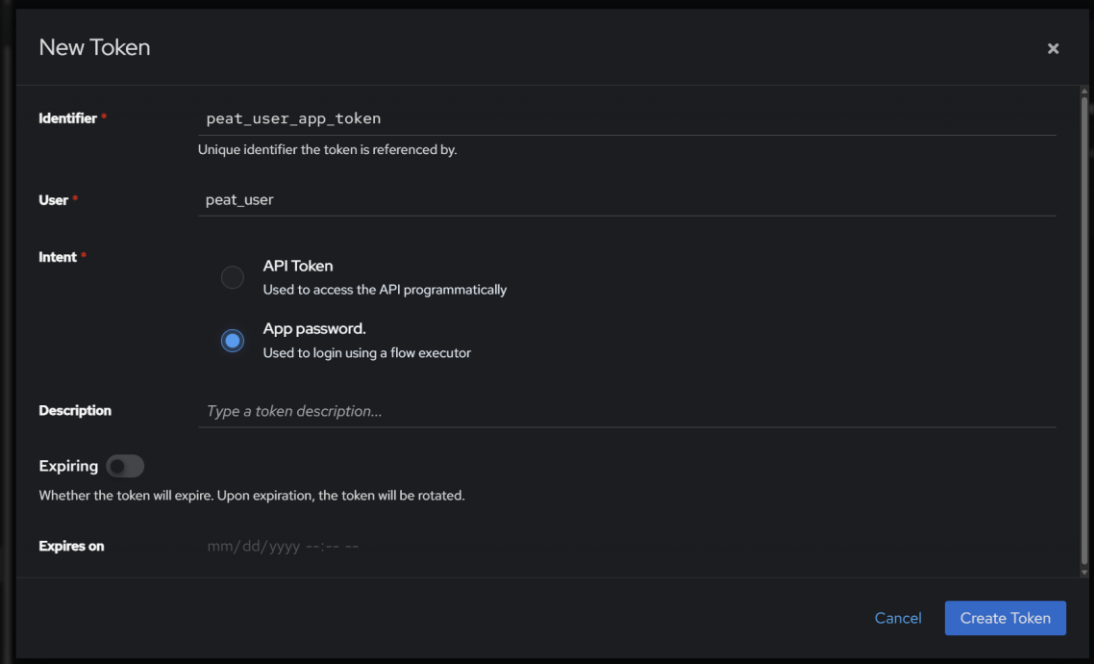
- Assign each user to their respective group



Setting Up Authentik

Step 4 (cont.): Create an App Password

- Create an app password for each user



The screenshot shows the 'New Token' dialog box in Authentik. The dialog has a dark theme and a close button (X) in the top right corner. It contains the following fields and options:

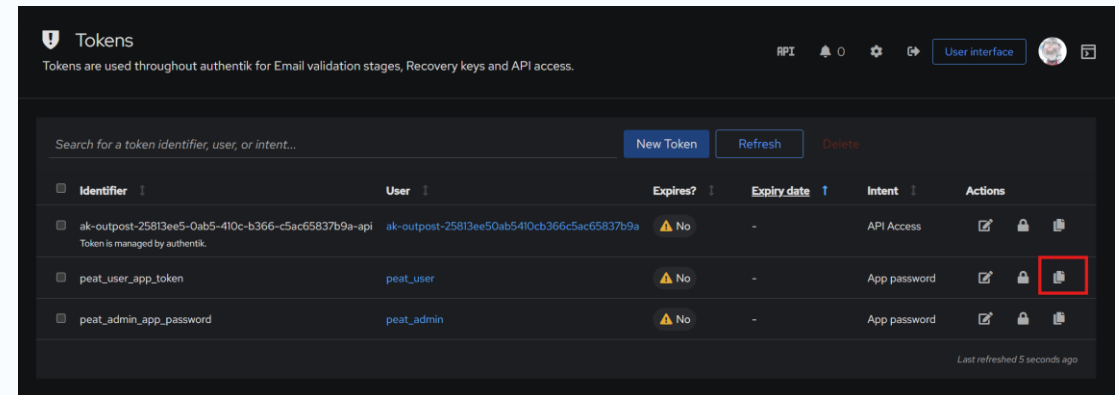
- Identifier ***: A text input field containing 'peat_user_app_token'. Below it, a small text label reads 'Unique identifier the token is referenced by.'
- User ***: A text input field containing 'peat_user'.
- Intent ***: Two radio button options:
 - ☐ **API Token**: Used to access the API programmatically.
 - ☒ **App password.**: Used to login using a flow executor.
- Description**: A text input field with the placeholder text 'Type a token description...'.
- Expiring**: A toggle switch that is currently turned off. Below it, a small text label reads 'Whether the token will expire. Upon expiration, the token will be rotated.'
- Expires on**: A text input field with the placeholder text 'mm/dd/yyyy --:-- --'.

At the bottom right of the dialog, there are two buttons: 'Cancel' and 'Create Token'.

Setting Up Authentik

Step 4 (cont.): Copy the App Password

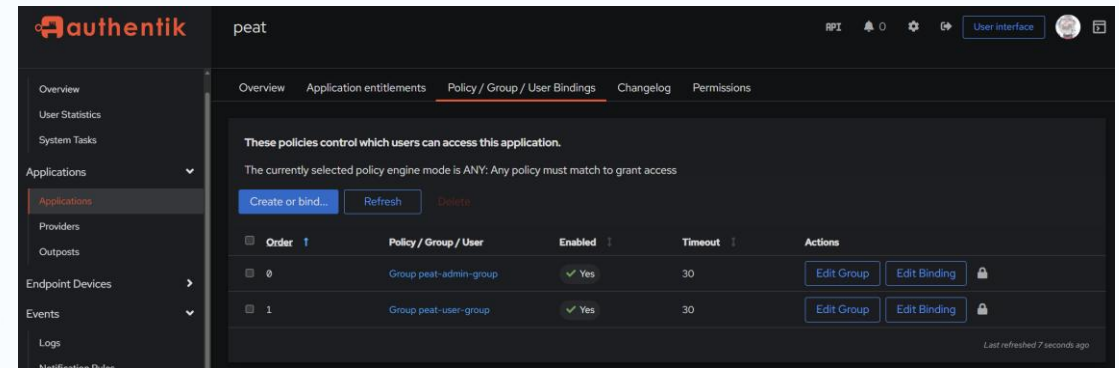
- Copy the app password to use for the auth request



Setting Up Authentik

Step 5: Bind the Groups to the Application

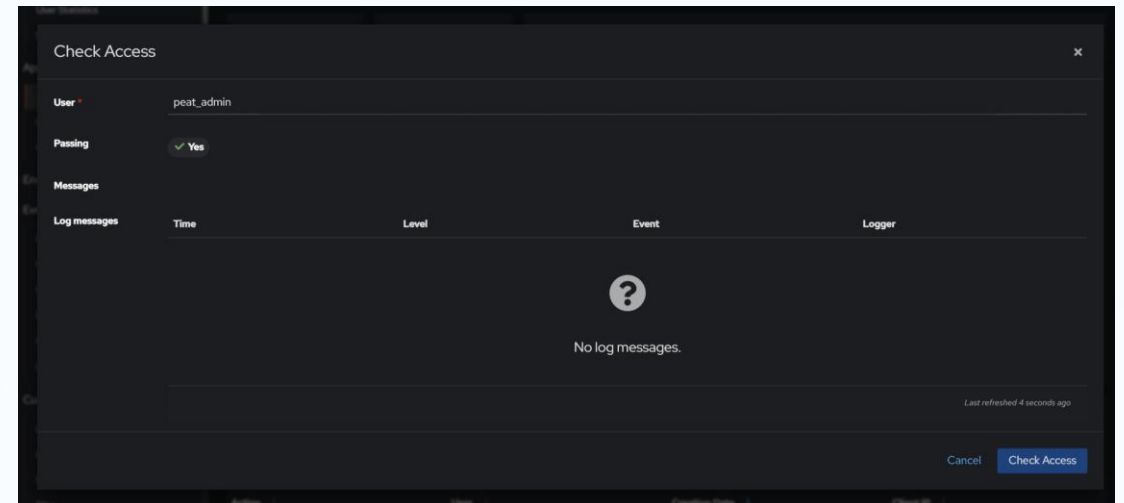
- Under Applications, open the application and select Policy / Group / User Bindings
- Bind the admin group and the user group
- Users in neither group are refused



Setting Up Authentik

Step 5 (cont.): Check Application Access

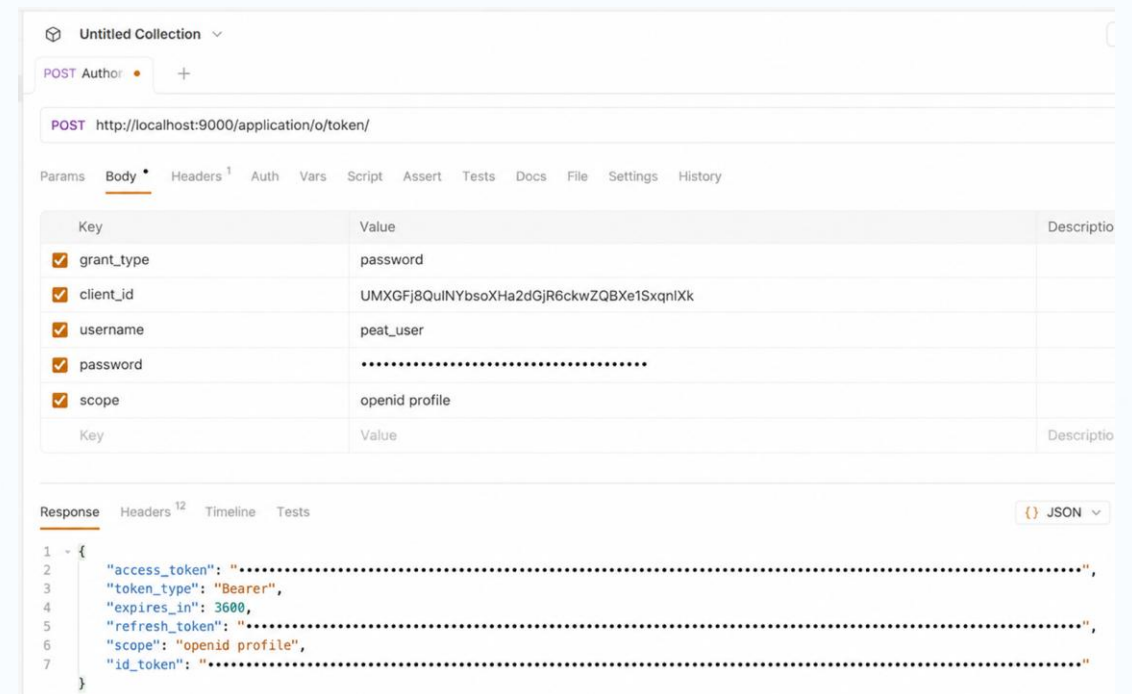
- Test a user against the bindings with the application's Check access page
- peat_admin and peat_user pass; any other user is denied



Setting Up Authentik

Step 6: Test Authentication

- Send a token request from Bruno (password grant, scope=openid profile)
- Only users in a bound group get a token



Setting Up Authentik

Step 6 (cont.): Read the groups claim

Call the userinfo endpoint with the access token. The groups claim lists the user's Authentik groups (response abridged)

GET <http://localhost:9000/application/o/userinfo/>
Authorization: Bearer <access_token>

```
{  
  "sub": "...",  
  "preferred_username": "peat_admin",  
  "groups": [  
    "peat-admin-group"  
  ]  
}
```

Scopes are not permissions

- A scope mapping adds claims to the response; it can't refuse a scope
 - Its expression must return a dictionary of claims: True or False is ignored
 - Testing a mapping shows what the expression returns, not what is enforced
- Any client can request any scope the provider offers
- To control who may use the application, bind groups (or a policy) to the application
- Let the identity provider report group membership, and let the application decide what each group may do

Using Entra ID instead of Authentik

- Entra ID needs more set up to give the same result
 - Define app roles in the app registration (or turn on the groups claim) and assign users or groups to them
 - Set Assignment required on the enterprise application, so only assigned users can sign in (like Authentik's group bindings)
 - The groups claim holds group object IDs, not names: map those IDs in [GroupMap]
- Entra's userinfo endpoint doesn't return roles or groups; PFC 2025 reads them from the ID token instead (set GroupsClaim=roles for app roles)
- Entra scopes are delegated permissions checked by a resource API (SMTP.Send for Exchange): they authorize the application, not the user's role
- Entra discourages the password grant (no MFA): use the authorization code flow with PKCE



PowerBuilder application changes

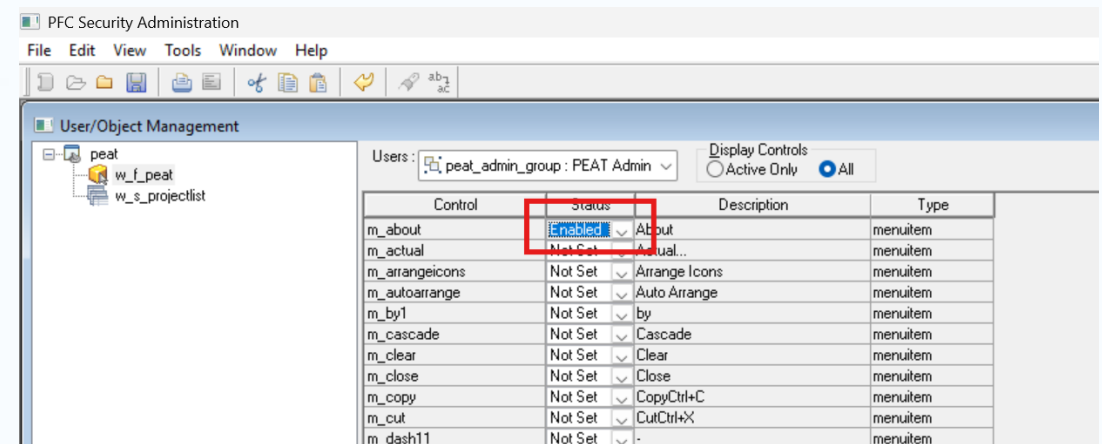
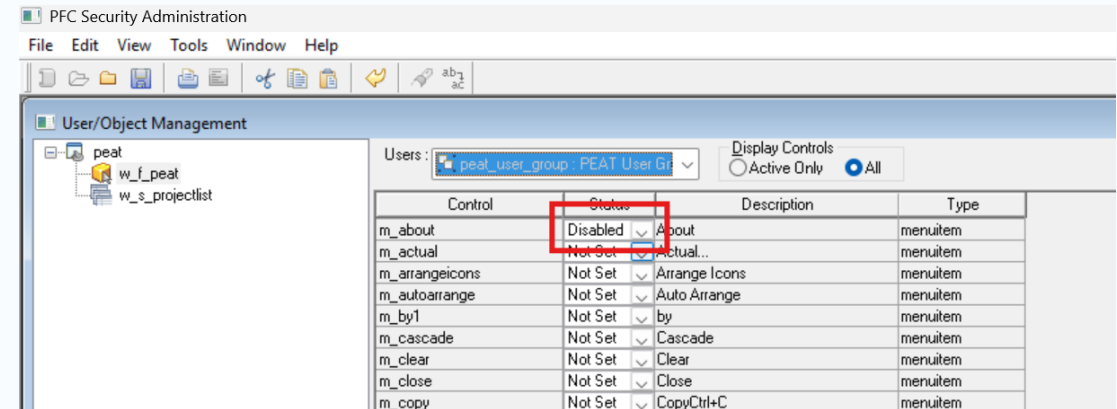
PowerBuilder application changes

- Scan the controls and configure PFC group security
- Put the database, OpenID and group mapping settings in an INI file
- Turn on the PFC security service at startup
- Connect to the security database and open the frame
- Sign the user in with PFC's OpenID Connect logon window
- Map the Authentik groups and initialize security with them
- Apply security to each window as it opens

PowerBuilder application changes

Step 1: Scan controls & configure PFC group security

- Ran the PFC Security Scanner on the application to capture all of the control information
- Using the PFC Security Administrator set security for the different groups
 - PFC Users – About window menu option disabled
 - PFC Admins – About window menu option enabled



PowerBuilder application changes

Step 2: Settings in an INI file

peat_security.ini: the security database, the identity provider, and Authentik group = PFC group

[Security]

; connection to the PFC security database

DefaultGroup=

[OpenID]

TokenEndpoint=http://localhost:9000/application/o/token/

UserInfoEndpoint=http://localhost:9000/application/o/userinfo/

ClientID=<client ID of the provider>

Scope=openid profile

[GroupMap]

peat-user-group=peat_user_group

peat-admin-group=peat_admin_group

PowerBuilder application changes

Step 3: Turn on the PFC security service

In the pfc_open event of the application manager (n_cst_peat)

```
// Security is on when peat_security.ini is present
ls_securityinfile = Left(ls_inifile, &
    LastPos(ls_inifile, "\")) + "peat_security.ini"
lb_secured = FileExists(ls_securityinfile)

If lb_secured Then this.of_SetSecurity(TRUE)

// ... connect to the application database ...

If lb_secured Then
    this.of_OpenSecured(ls_securityinfile)
Else
    Open(w_f_peat)
End If
```

PowerBuilder application changes

Step 4: Connect and sign the user in

of_OpenSecured connects to the security database, opens the frame, then shows PFC's OpenID Connect logon window

```
gnv_trans_security = CREATE n_tr  
gnv_trans_security.of_Init(as_securityinifile, "Security")  
IF gnv_trans_security.of_Connect() = -1 THEN Return -1
```

```
Open(w_f_peat)
```

```
// PFC's OpenID Connect logon window (w_openid_logon)
```

```
li_rc = this.of_OpenIDLogonDlg(lnv_openid, &  
    as_securityinifile, "OpenID")
```

```
IF li_rc <> 1 THEN
```

```
    Close(w_f_peat)
```

```
    Return li_rc
```

```
END IF
```

PowerBuilder application changes

Step 5: Map the groups and initialize security

of_MapGroups translates the groups claim;
of_InitSecurity uses the result instead of memberships in the database

```
inv_security.of_MapGroups(Inv_openid.is_groups, &  
    as_securityinfile, "GroupMap", ls_groups)
```

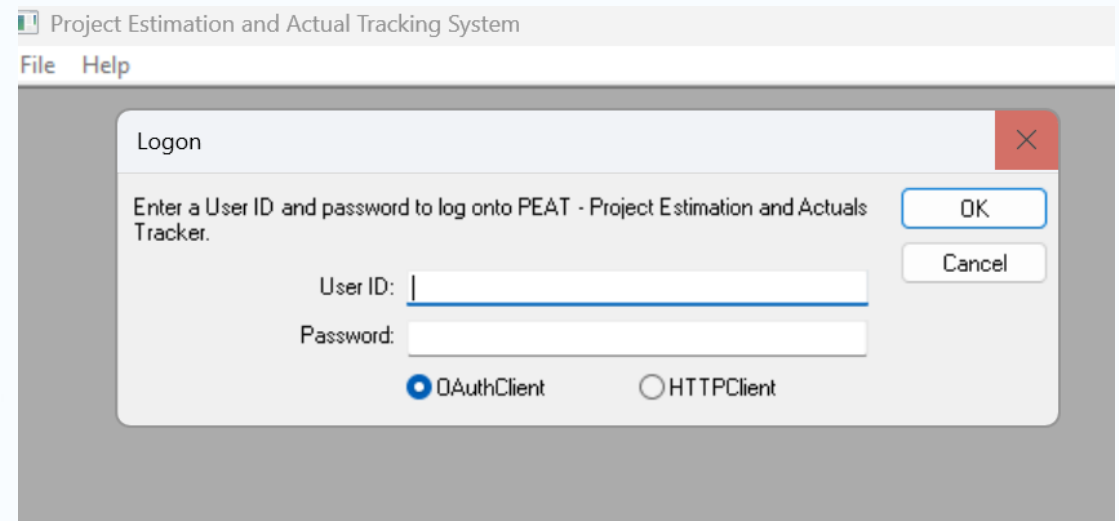
```
li_rc = inv_security.of_InitSecurity(gnv_trans_security, &  
    "peat", this.of_GetUserID(), ls_groups, &  
    ProfileString(as_securityinfile, "Security", &  
        "DefaultGroup", ""))
```

```
// The frame is already open, so secure it now  
inv_security.of_SetSecurity(w_f_peat)  
OpenSheet(w_s_projectlist, w_f_peat, 0, Layered!)
```

PowerBuilder application changes

Step 6: PFC's OpenID Connect logon window

- w_openid_logon replaces the custom login window
- Requests a token (password grant) with OAuthClient or HTTPClient
- Reads the groups from the userinfo endpoint
- ShowResponses=1 shows Authentik's responses



PowerBuilder application changes

Step 6 (cont.): The token request

Inside w_openid_logon,
with OAuthClient
(simplified)

```
Inv_tokenrequest.TokenLocation = inv_openidattrib.is_tokenendpoint  
Inv_tokenrequest.Method = "POST"  
Inv_tokenrequest.AppendParam("grant_type", "password")  
Inv_tokenrequest.AppendParam("client_id", inv_openidattrib.is_clientid)  
Inv_tokenrequest.AppendParam("username", as_userid)  
Inv_tokenrequest.AppendParam("password", as_password)  
Inv_tokenrequest.AppendParam("scope", inv_openidattrib.is_scope)
```

```
ll_rc = Inv_oauthclient.AccessToken(Inv_tokenrequest, &  
    Inv_tokenresponse)  
If ll_rc = 1 And Inv_tokenresponse.GetStatusCode() = 200 Then  
    inv_openidattrib.is_accesstoken = &  
        Inv_tokenresponse.GetAccessToken()  
End If
```

PowerBuilder application changes

Step 7: Apply security to each window

In each sheet's pfc_preopen event, or once in an ancestor window

```
// w_s_projectlist: pfc_preopen  
// PFC security is only on when peat_security.ini is present  
If IsValid(gnv_app.inv_security) Then &  
    gnv_app.inv_security.of_SetSecurity(This)
```

PowerBuilder application changes

Step 8: Open the project list after sign-in

The frame only opens it when security is off; with security on, of_OpenSecured opens it after the logon

```
// w_f_peat: pfc_postopen
```

```
// With PFC security on, n_cst_peat opens the project list
```

```
// after the logon
```

```
If Not IsValid(gnv_app.inv_security) Then &
```

```
    OpenSheet(w_s_projectlist, this, 0, layered!)
```

PowerBuilder application changes

Optional: Deny by default

PFC 2025: disable (or hide) every control and menu item that has no setting, so the security database only needs to grant access

```
li_rc = inv_security.of_InitSecurity(gnv_trans_security, &  
    "peat", this.of_GetUserID(), ls_groups, "")
```

```
// Only controls and menu items granted 'E' stay enabled  
inv_security.of_SetDefaultStatus(inv_security.DISABLE)
```

```
inv_security.of_SetSecurity(w_f_peat)
```

Demo



Final thoughts

- The sample is a PFC application using the PFC security service
- PFC 2025 removes the two issues this approach used to have
 - Group membership comes from Authentik: of_InitSecurity takes the groups, and of_MapGroups maps their names to PFC groups
 - Deny by default (the principle of least privilege) is available: of_SetDefaultStatus
 - PoLP was introduced in 1975 (Saltzer and Schroeder); PFC dates from 1995 and PowerBuilder 5.0
- PFC 2025 also adds the OpenID Connect logon window (of_OpenIDLogonDlg) and keeps restrictions in place when code enables controls (of_ApplyRestrictions)
 - Not in the 2019 and 2022 versions of PFC
- Next: the authorization code flow with PKCE (the password grant is deprecated in OAuth 2.1, with no MFA or browser SSO)

Session Agenda (Recap)

- OAuth2 vs OpenID Connect (OIDC)
- Setting Up Authentik
- PowerBuilder application changes
- Demo

Thank you

Connect with Us



community.appeon.com

Discussions, tech articles and videos, free online training, and more.



facebook.com/AppeonPB

Encourage us with a “like”, see cool pics, and get notified of upcoming events.



x.com/AppeonPB

Follow Appeon and community members to get the latest tech news.



linkedin.com/company/appeon

Build up your career profile, and stay in contact with other professionals.



youtube.com/Appeon

Share important Appeon videos with others; no account registration required.

Q&A Time

