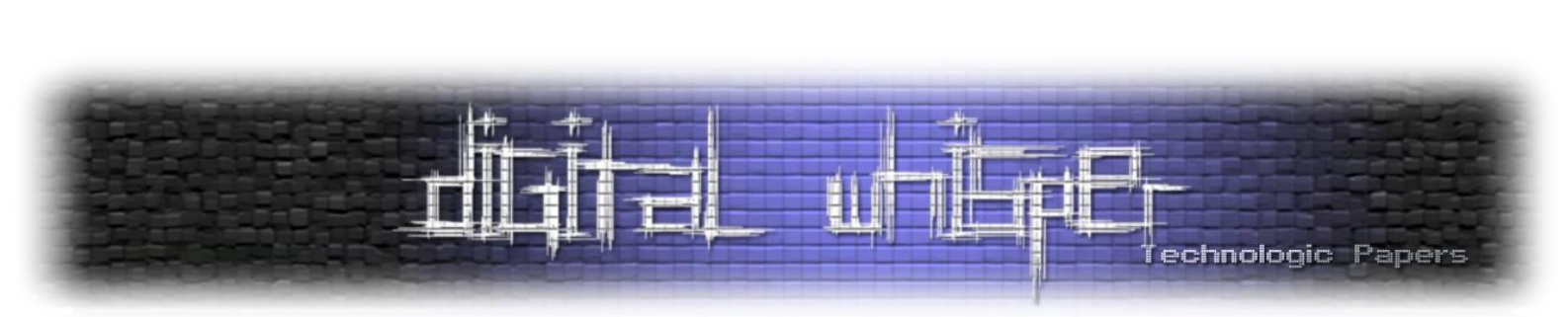


Managed Identities Azure

מאת ספיר פדרובסקי

טוב, איפה מתחילים?

היה הייתה ארץ מופלאה בשם Azure. הארץ הזו הכילה עשרות אם לא מאות שירותים שונים. כמו למשל Cosmos DB, Key Vaults, Virtual Machines, ובין כל השירותים האלו, שירות אחד קטן ונפלא בשם Azure Active Directory (או בשמו הכבר-לא-כל-כך-חדש והמזעזע: Entra ID).

בתור מכורה קשה לתחום הזהויות, עבודתי התמקדה בעיקר בספקי זהות שונים כמו AWS IAM,Entra ID, Okta ו-AD. ואני אוהבת את זה! אבל - תמיד חלמתי לצאת מגבולות עולם הזהויות, ולהכנס לעולם המחקר של הדבר העצום הזה שנקרא: ענן. בעודי משוטטת ביער אין סופי של מידע, נתקלתי במפתח שלי לעולם הגדול, וזה הוא נושא המאמר היום. אני נרגשת להכיר לכם: Managed Identities!

בואו נתחיל.

מה זה?

מדובר בעצם בתת-סוג של Service Principal. זאת אומרת, אם נריץ ב-Graph API את השאילתה הבאה:

```
https://graph.microsoft.com/beta/servicePrincipals?$filter=servicePrincipalType eq
'ManagedIdentity'
```

נקבל את כל ה-MI שיש לנו ב-Tenant. המטרה של MI (וההבדל בין ישות כזו לבין ServicePrincipal "רגיל") היא לאפשר לאפליקציה לקבל Token, מבלי שתצטרך לתחזק לה סיסמא. בעיניי זה הזכיר מאוד GMSA ב-AD, או AssumeRole קצת אם במקרה יצא לכם (משתתפת בצערכם) לעבוד עם AWS...

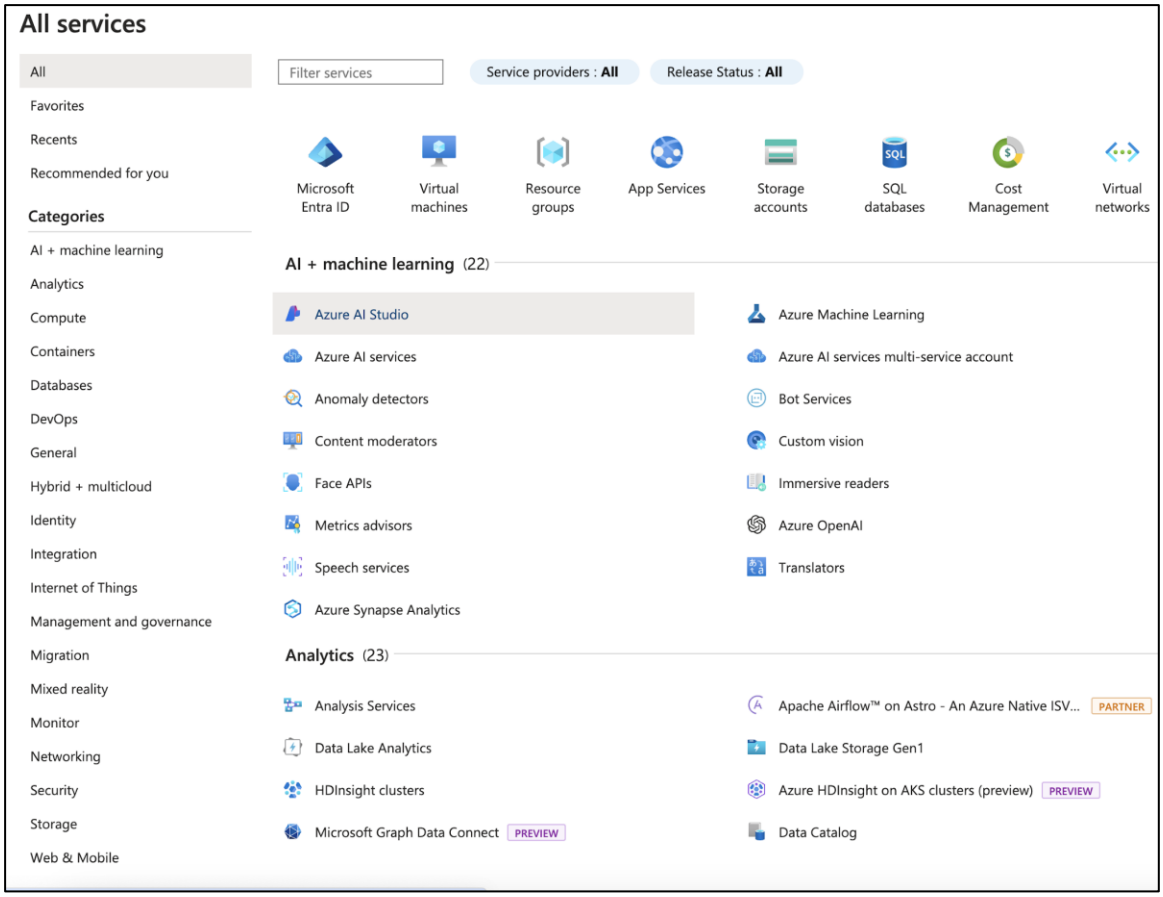
כדי להבין באמת מה זה אומר, מה ההשלכות של זה וכיצד ניתן לנצל את זה, נצטרך לקחת מאה צעדים אחורה, ולהכיר קצת יותר לעומק את העולם המופלא של Azure.

קדימון (באורך של מאמר בפני עצמו...)

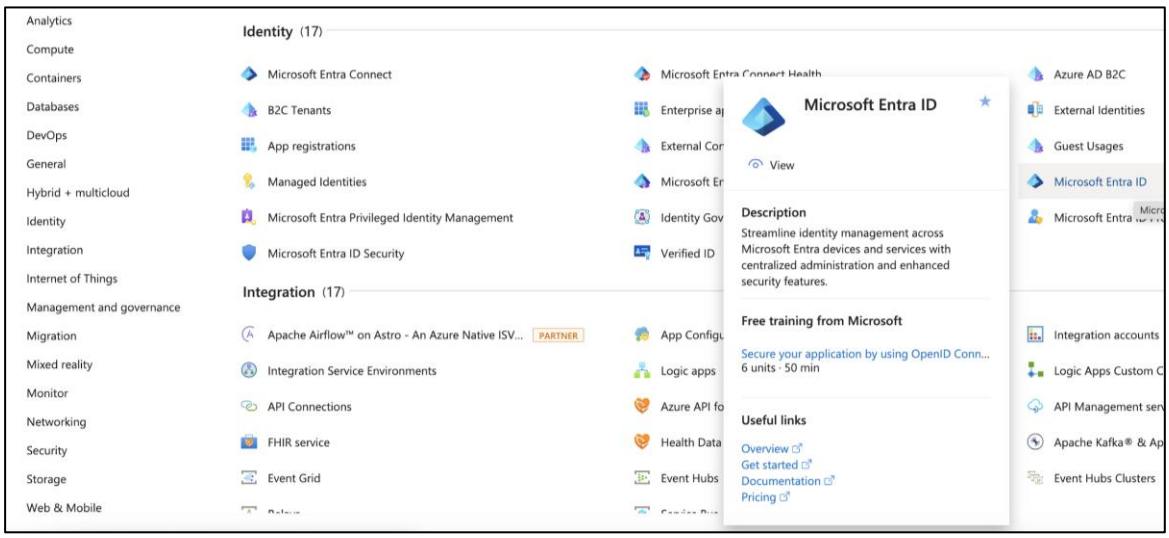
אז כפי שהסברתי במבוא, Entra ID הוא רק שירות אחד שמסופק על ידי Azure. באמצעות Entra נוכל לנהל את המשתמשים בענן שלנו, ההרשאות המנהליות שלהם וכו.

אבל אם אי פעם הסתכלתם על Role-ים ב-Entra, אני בטוחה ששמתם לב, שאין שם שום Role שמדבר על גישה למכונות וירטואליות, שרתי DB ועוד. זאת משום, שכל השירותים האלו בכלל מנוהלים במקום אחר!

זה יהיה הרבה יותר ברור עם תמונה, הנה כל השירותים המסופקים על ידי Azure:



הנה אחד מהם, שנקרא Entra ID:

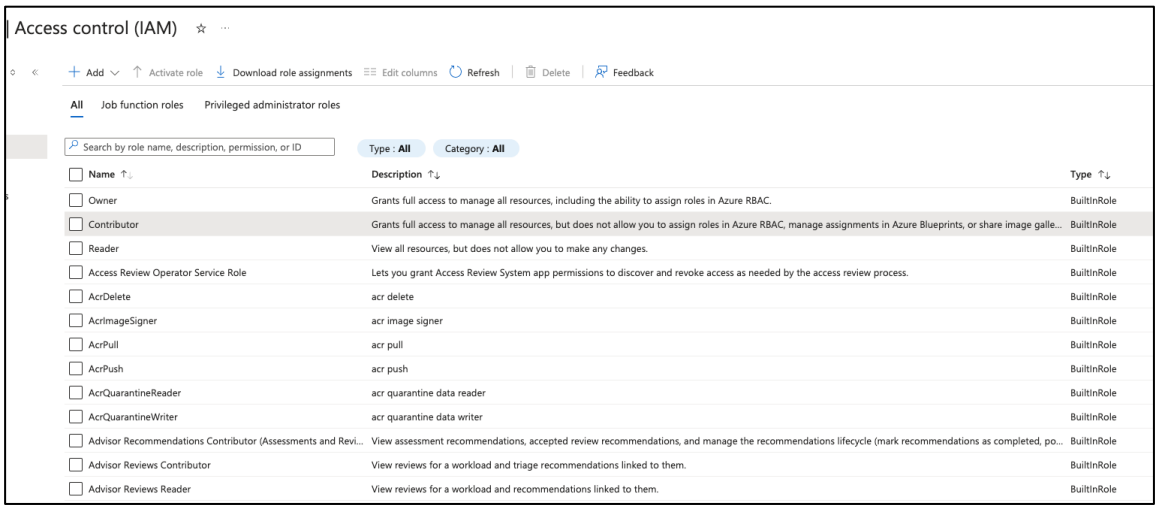


בואו נכנס לשירות Virtual Machines, אנחנו יכולים לראות שיש פה מכונות וירטואליות. נניח ועכשיו אנחנו רוצים להחליט לאיזה משתמש ב-Tenant שלנו יש גישה למכונה הוירטואלית X. ומעבר לגישה, אילו הרשאות בדיוק יש לו? לנהל את המכונה? להדליק אותה? למחוק אותה?

את כל הדברים האלו אנחנו לא מנהלים ב-Entra, הם מנוהלים במודל אחר בשם RBAC (קיצור של: Resource Based Access Controls). המשתמשים הם אותם משתמשים שאנחנו יוצרים מוחקים ומנהלים ב-Entra, אבל ספציפית את ההרשאות על השירותים השונים בענן, אנו מנהלים במקום אחר. ואם נשתמש במונחים קצת יותר טכניים (כי מה זה אומר "מקום אחר"?), הכוונה היא פשוט host אחר שמספק API אחרים. אם אנחנו יודעים שהפעולות של Entra מנוהלות ברובן על ידי GraphAPI. אז הרבה מהפעולות של RBAC מנוהלות תחת:

https://management.azure.com/

אז בואו נשים בצד רגע את Global Administrator, Application Administrator ושאר החברים שלנו. ונכיר את Contributor, Reader ועוד:



נוכל גם ממש לראות את ההגדרות של כל Role ולהבין מה היכולות שהוא מספק.

בואו ניקח לדוגמא אחד:

Virtual Machine User Login

BuiltInRole

Permission Type

(•) Actions ( ) DataActions

Showing 26 of 26 permissions

Type	Permissions	Description
Microsoft.Compute		
Read	Get Virtual Machine	Get the properties of a virtual machine
Read	Summarizes latest patch assessment operation results	Retrieves the summary of the latest patch assessment operation
Read	Lists all patches assessed in patch assessment operation	Retrieves list of patches assessed during the last patch assessment operation
Read	Summarizes latest patch installation operation results	Retrieves the summary of the latest patch installation operation
Read	Lists all patches considered in patch installation operation	Retrieves list of patches attempted to be installed during the last patch installation operation
Read	Get Virtual Machine log definitions	Gets the available logs for Virtual Machine.
Read	Read diagnostic setting	Gets the diagnostic setting for the Virtual Machine.
Read	Get Virtual Machine Extension	Get the properties of a virtual machine extension
Read	Get Virtual Machine Instance View	Gets the detailed runtime status of the virtual machine and its resources
Read	Get Virtual Machine Metric Definitions	Reads Virtual Machine Metric Definitions
Read	Get Virtual Machine run command	Get the properties of a virtual machine run command
Read	Lists Available Virtual Machine Sizes	Lists available sizes the virtual machine can be updated to
Microsoft.Network		
Read	Get Load Balancer	Gets a load balancer definition
Read	Get Network Interface	Gets a network interface definition.
Read	Get Public Ip Address	Gets a public ip address definition.
Read	Get Virtual Network	Get the virtual network definition
Microsoft.HybridCompute		
Read	Read Azure Arc machines	Read any Azure Arc machines
Read	Read Azure Arc patchInstallationResults	Reads any Azure Arc patchInstallationResults
Read	Read Azure Arc patchInstallationResults/softwarePatches	Reads any Azure Arc patchInstallationResults/softwarePatches
Read	Read Azure Arc extensions	Reads any Azure Arc extensions
Read	Read Azure Arc licenseProfiles	Reads any Azure Arc licenseProfiles
Read	Read Azure Arc machines's Hybrid Identity Metadata	Read any Azure Arc machines's Hybrid Identity Metadata
Read	Read Azure Arc patchAssessmentResults	Reads any Azure Arc patchAssessmentResults
Read	Read Azure Arc patchAssessmentResults/softwarePatches	Reads any Azure Arc patchAssessmentResults/softwarePatches
Read	Read Azure Arc runcommands	Reads any Azure Arc runcommands
Microsoft.HybridConnectivity		
Other	Endpoints_ListCredentials	Gets the endpoint access credentials to the resource.

אז אם עד כה הכרנו השמת Role באמצעות פקודה מהסוג:

```
get https://graph.microsoft.com/user/assignedroles
```

או משהו בסגנון, כשאנחנו מדברים על RBAC זה יראה יותר ככה:

```
POST /batch?api-version=2020-06-01 HTTP/2
Host: management.azure.com
Content-Length: 895
X-Ms-Client-Session-Id: c7e11a7ec4ae47178a6f5d6c9b2aa247
X-Ms-Command-Name: { Microsoft_Azure_AD.Batch:0,AddRoleAssignments.batch:1}
Accept-Language: en

{
  "requests":[
    {
      "content":{
        "Id":"9a26649e-4da7-43db-acdd-885d0d89050a",
        "Properties":{
          "PrincipalId":GUID,
          "PrincipalType":"User",
          "RoleDefinitionId":"/providers/Microsoft.Authorization/roleDefinitions/acdd72a7-3385-48ef-bd42-f606fba81ae7"=>Reader,
          "Scope":"/subscriptions/GUID/resourceGroups/resourceGroupName/providers/Microsoft.Compute/virtualMachines/VM-Name",
          "Condition":null,
          "ConditionVersion":null
        }
      },
      "httpMethod":"PUT",
      "name":"7395c1b4-0be6-447f-bae3-4a23ebf5c005",
      "requestHeaderDetails":{
        "commandName":"Microsoft_Azure_AD.AddRoleAssignments.batch"
      },
      "url":
      "https://management.azure.com/subscriptions/GUID/resourceGroups/resourceGroupName/providers/Microsoft.Compute/virtualMachines/VM-Name/providers/
      Microsoft.Authorization/roleAssignments/9a26649e-4da7-43db-acdd-885d0d89050a?api-version=2020-04-01-preview"
    }
  ]
}
```

נעשה קצת סדר - מה הבנו עד כה:

1. יש רק מקום אחד לתחזוקת משתמשים ב-Azure והוא Entra ID
2. יש 2 מקומות לניהול הרשאות של משתמש ב-Azure:
   - Entra ID שאחראי על הרשאות בעולם הזהויות (Application Administrator)
   - RBAC שאחראי על הרשאות למשאבים השונים שמסופקים על ידי Azure (כגון DB Reader).
3. יש דבר כזה Managed Identities ולא ברור מה הוא קשור

מה קשור Managed Identities?

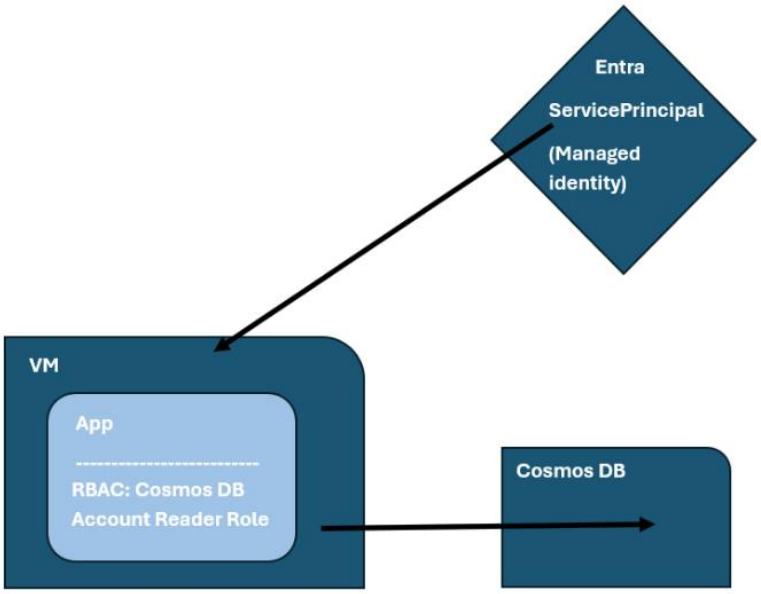
הרעיון מאחורי הישות הזו הוא מאוד ברור ודי מתבקש למען האמת. בואו נניח שלחברה שלי יש אפליקציה שרצה על מכונה וירטואלית ב-Azure. נגיד שזו אפליקציה של חנות אינטרנטית (טריגר לפרויקט סיום בבגרות במדמ"ח...). יש לי גם שרת DB שמחזיק את הפריטים שניתן לקנות בחנות.

כאשר משתמש מתחבר לאפליקציה, ורוצה לראות מה הפריטים שיש ב-DB, האפליקציה צריכה לתשאל אותו. אך מה היא אותה "אפליקציה"? באילו הרשאות היא משתמשת כדי לתשאל את ה-DB? איך היא מתאמתת?

אז נכון שלמדנו בעבר על Applications ו-Service Principals ואכן מדובר על אותו הקונספט, רק עם שינוי אחד קטנטן.

כדי שלא נצטרך להחזיק בצורה לא מאובטחת את הסיסמא של האפליקציה (אהמ אהמ Hard Coded), היא כבר תהיה built-in בתוך המכונה הווירטואלית עליה האפליקציה רצה! ויותר מזה, Azure ידאג לנהל לנו את תחלופת הסיסמא!

זאת אומרת, הבעיה שהקונספט הזה מנסה לפתור, היא אחסון וניהול של סיסמאות אפליקטיביות. הביטו על התרשים הבא:



בחלק העליון שלו, אנו רואים את Entra, כפי שניתן לראות, היא מכילה ישות Service Principal שבמקרה שלנו היא מסוג Managed Identity. לאחר מכן, בצד ימין למטה אנחנו רואים שרת DB ששייך ל- Azure Tenant שלנו. אליו נצטרך לגשת ולשלוף ממנו מידע (כמובן עם משתמש בעל הרשאות מתאימות).

אחת הדרכים לעשות זאת היא לאפשר לאפליקציה שלנו לפעול עם הרשאות delegated שזה בעקרון אומר שהמשתמש לוחץ על "אני מסכים לאפליקציה לרוץ בהרשאות שלי" .

בתרחיש הזה אנחנו מאפשרים לאפליקציה לפעול בהרשאות של המשתמש, שזה לא תמיד אידיאלי ועלול לתת לאפליקציה הרשאות הרבה יותר גבוהות משהיא צריכה בפועל. במידה ולא שמעתם על המונח, אנו שואפים תמיד לפעול בגישת least-privilege, שזה בגדול אומר, לתת לישות את ההרשאות המינימליות שהיא צריכה כדי לבצע את הפעולות הנחוצות.

לכן, הדרך הראשונה לא אידיאלית ולא מתאימה לגישה הזו. הדרך היותר טובה, היא באמצעות MI. עכשיו נבין למה ואיך!

ראשית, אנחנו לא משתמשים בהרשאות של המשתמש כדי לבצע פעולות, אנחנו נשתמש בהרשאות של ה-MI, ככה שנוכל לתת לו בדיוק את ההרשאות שהאפליקציה שלנו תצטרך ולא יותר מזה. (נניח במקרה שלנו זה יהיה משהו כמו DB Reader).

כעת אנחנו בבעיה אחרת, יש לנו אפליקציה עם הרשאות שהולכת לבצע פעולות, מה שאומר שהאפליקציה הזו צריכה Credentials, וגם להתאמת. ואם יש לה Credentials, הם צריכים להישמר בצורה כזו שהאפליקציה תוכל באופן אוטומטי לבצע הזדהות.

ופה נכנס ה-MI. כפי שתיארתי, Azure אחראי על MI, הוא יוצר להם סיסמא אוטומטית וגם אחראי על החלפתה פעם בכמה זמן.

בתוספת על כל הסיפור הזה, יש עוד עניין, MI תמיד יהיה מקושר (bound) למשאב מסוים, ולא יהיה ניתן להזדהות איתו מחוץ למשאב הזה.

בואו נתעכב רגע על המשפט האחרון: דיברנו עד כה על אפליקציה, אז מה פתאום אני אומרת ש-MI מקושר למשאב? העניין הוא שלא ניתן להזדהות עם ה-MI מחוץ למשאב שהוא קשור אליו. וזה עובד מפני שהתעודה שבאמצעותה מזדהה ה-MI נמצאת על המשאב.

אגב, כשאני כותבת משאב אני מדברת על Resource, למשל: שרת DB, מכונה וירטואלית Function App, Key Vault ועוד ועוד ועוד...

נתון נוסף שטוב להכיר בנושא הוא שלא לכל משאב אפשר לקשור MI, הנה רשימה של כל המשאבים שניתן להקצות להם MI. בעצם כאשר אנו קושרים בין משאב ל-MI, אנחנו יוצרים את האפשרות להזדהות עם ה-MI (מהמשאב הזה בלבד).

יש משאבים שבאמצעותם הרבה יותר אינטואיטיבי להבין את הסיפור הזה, כמו למשל מכונות וירטואליות, אני בחרת ב-Function App בשביל הדוגמא, רק כדי שתוכלו לראות סוגים נוספים ומעניינים ☺.

אז Function App זו דרך נחמדה ליצור פונקציות שעושות דברים ויוטרגו אוטומטית בעת אירוע מסוים, למשל: כל פעם שמישהו עושה:

```
Get https://sapir-function-app
```

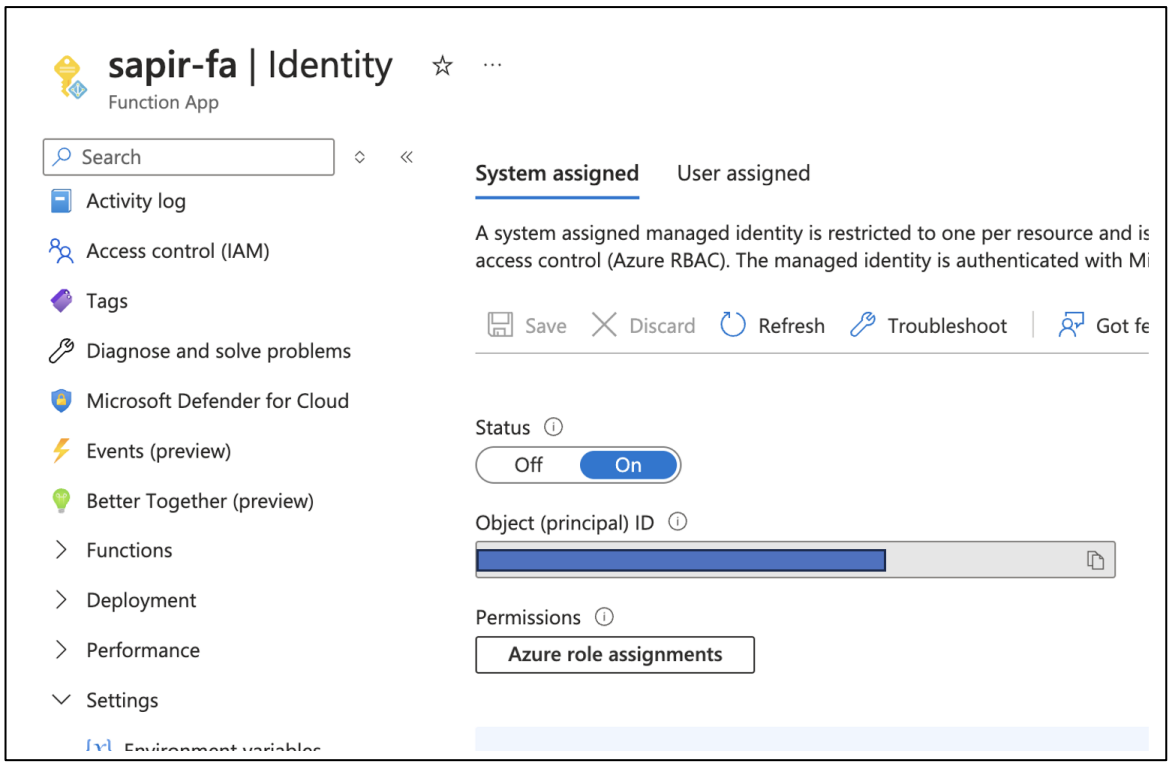
תרוץ פונקציה שהולכת ל-DB ומתעדת את כתובת ה-IP שפנתה אלי. הפונקציות עצמן רצות על סוג של מכונה, במקרה שלי בחרתי לכתוב את הפונקציה בשפת PowerShell, אז מדובר במכונת ווינדוס. ועליה (אתם בטח כבר מנחשים) יש MI.

בואו נתחיל לקנפג ונזרום משם!

ראשית, יש שני סוגים של MI:

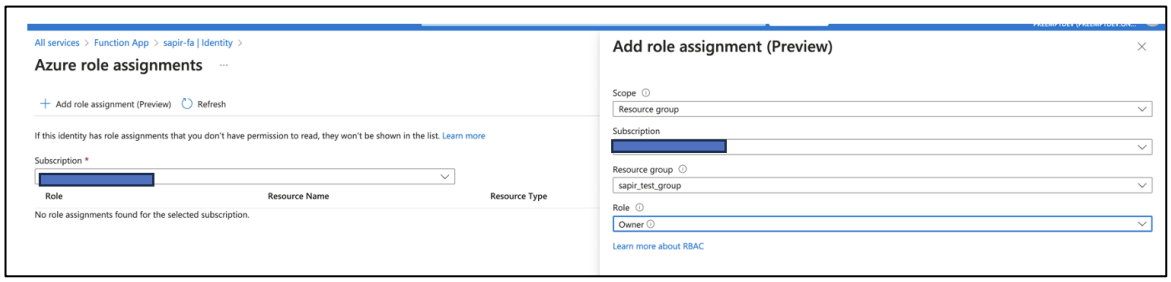
- System assigned
- User assigned

יש ביניהם כמה הבדלים, כאשר אנחנו יוצרים System assigned, שמעתה יקרא SAMI (קיצור של System Assigned Managed Identity), אנחנו בעצם קודם יוצרים משאב, ואז יכולים לסמן "צור SAMI עבור המשאב הזה". הנה:



ה-MI הזה, לנצח יהיה שייך ל-Function App הזה, ולא ניתן לשייך אותו למשאבים אחרים. נוכל גם לראות שכל השליטה שיש לנו בנוגע ל-SAMI זה אם הוא קיים או לא, אין לנו דרך לבחור את הסיסמא שלו, לראות אותה או לנהל אותה.

אבל כן נוכל לתת לו הרשאות במידה ונרצה (הרשאות RBAC). כך זה נראה:



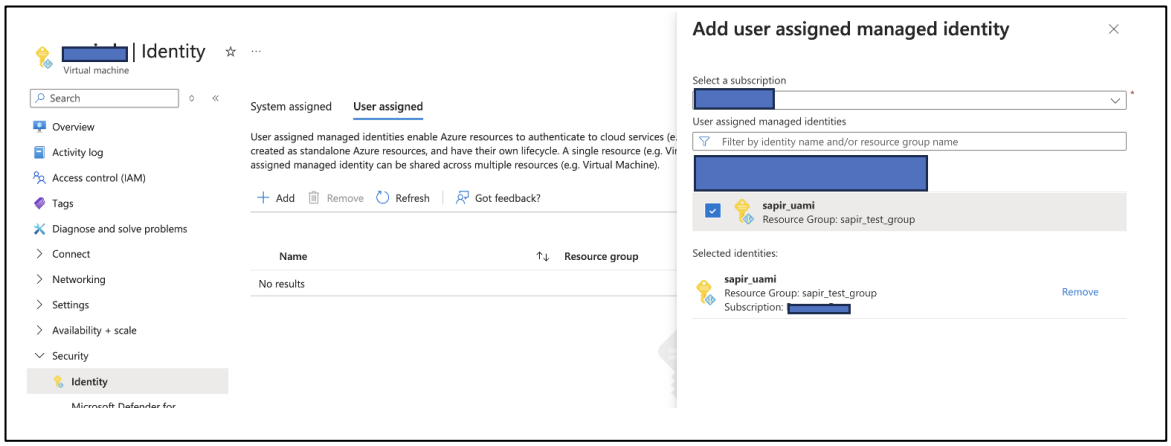
לא אכנס פה לדיבורים על Subscriptions ו-Resource Groups וכו', כי זה לא מאוד חשוב, אבל בגדול אלו תחומים שונים (Scopes) ועליהם אני נותנת ל-SAMI שלי גישה. זאת אומרת, במקרה הזה נתתי הרשאות

Owner רק על ה-Resource Group הספציפי sapir_test_group. (לצורך ההדגמה Resource Group יכול להכיל שרתי DB מכונות וירטואליות וכו').

אז ראינו איך יוצרים SAMI ואיך נותנים לזה הרשאות, יופי. מה זה **User Assigned Managed Identity**? או בשמו השני - **UAMI**? ההבדל בין UAMI ל-SAMI זה שלעומת SAMI שקשור רק למשאב אחד לנצח, UAMI יכול להיות קשור למספר משאבים.

מה ההיגיון כאן? נניח שיש לנו קבוצה של מכונות וירטואליות שלכולן אנחנו רוצים לתת את אותן הרשאות, זה מעט מכביד ליצור SAMI לכל אחת מהמכונות ולתת לכל SAMI כזה את אותו סט הרשאות. לכן, על מנת לחסוך בפעולות (ובכללי, זה פשוט מרגיש נכון יותר), נוכל ליצור בעצמינו MI, בהתחלה הוא לא יהיה קשור לשום משאב, פשוט Standalone MI, כמובן שלא נוכל להזדהות איתו או משהו בסגנון, רק לתת לו הרשאות. לאחר מכן, נוכל לקשר אותו לכמה משאבים שנרצה במקביל.

אז יצרתי UAMI בשם sapir_uami, וכעת אני יכולה להחיל אותו על משאבים שונים, וכמובן לתת לו הרשאות כרצוני. החלטתי להשים אותו על מכונה וריטואלית:



מגניב. נסכם שוב מה הבנו עד שלב זה:

1. יש 2 סוגים של MI
2. אפשר לקשור User Assigned Managed Identity למספר משאבים
3. אפשר לתת הרשאות ל-MI
4. אי אפשר לשלוט על הסיסמא של MI

אגב, כדי לזהות אם ה-MI הוא System Assigned או User Assigned, תוכלו להסתכל על הערך הבא:


```
},
{
    "id": "[redacted]",
    "deletedDateTime": null,
    "accountEnabled": true,
    "alternativeNames": [
        "isExplicit=True",
        "/subscriptions/66022d93-18cf-406d-aa68-323cf7819b3f/resourcegroups/sapir_test_group/providers/Microsoft.ManagedIdentity/userAssignedIdentities/sapir_uami"
    ],
    "createdDateTime": "2024-09-03T07:17:50Z",
    "deviceManagementAppType": null,
    "appDescription": null,
    "appDisplayName": null,
    "appId": "[redacted]",
    "applicationTemplateId": null,
    "appOwnerOrganizationId": null,
    "appRoleAssignmentRequired": false,
    "description": null,
    "disabledByMicrosoftStatus": null,
    "displayName": "sapir_uami",
    "errorUrl": null,
    "homepage": null,
    "isAuthorizationServiceEnabled": false,
```


תחת alternativeNames יהיה ערך בשם isExplicit:

- True == UAMI
- False == SAMI

הנה query נחמד באמצעות המודול az:

```
az ad sp list --query="[?servicePrincipalType=='ManagedIdentity' &&
alternativeNames[?contains(@,'isExplicit=False')]]" --all
```

אגב, אם כבר מסתכלים על ה-attributes שקיבלנו, אנחנו יכולים לראות appId, אבל ממתי יש אפליקציה קשורה לכל הסיטואציה הזו? מה זו האפליקציה הזו?

אז האמת שהתשובה לזה די משעשעת. אם אי פעם שמעתם על המונח "פתרון פלסטר" , זה בערך מה שזה.

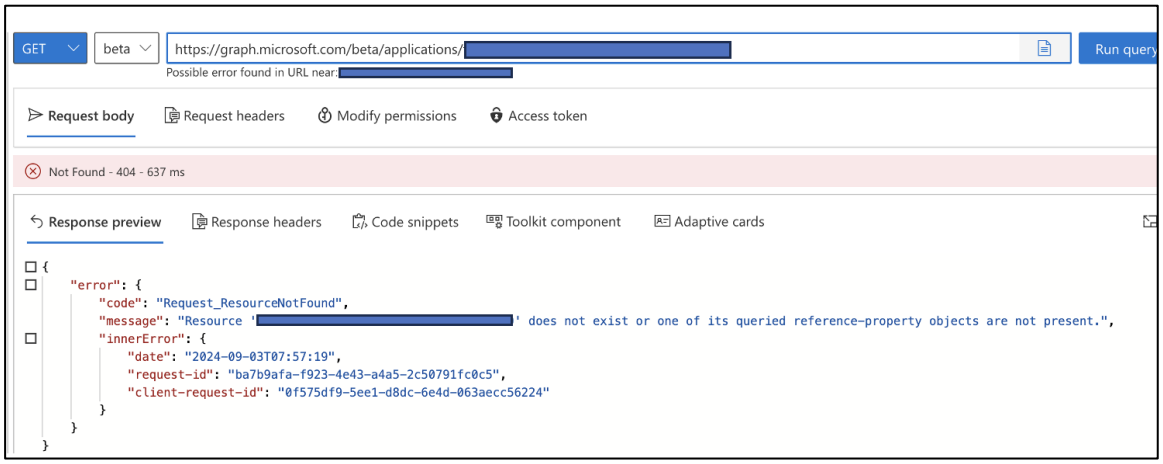
כפי שציינתי קודם, MI הוא סוג של Service principal, ו-Service Principal זו הרי ישות שמייצגת אפליקציה.

אבל במקרה שלנו, MI הוא servicePrincipal שלא מייצג אפליקציה, אלא קשור למשאב.

אבל, משום שהוא בתבנית של servicePrincipal, מיקרוסופט הייתה חייבת לשים ערך כלשהו בשדה appId, וכנראה שהוא לא יכול להיות null (לעומת appDisplayName למשל), הם פשוט שמו שם GUID אקראי, רק כדי למלא את המשבצת הזו.

די משעשע, אבל זה הסיפור.

ואם תקחו את ה-GUID שיש בשדה appId ותנסו לראות לאיזו אפליקציה הוא שייך, אתם תגלו שהיא לא קיימת!



עכשיו אחרי שקינפגנו לנו **MI** לשחק איתו, בואו נראה מה באמת הולך שם, סתם כי זה כיף. אז נניח שיש לנו מכונה וירטואלית וקנפגנו עליה MI, כעת אנחנו יכולים לבצע אותנטיקציה עם ה-MI מתוך המכונה. איך? פשוט נפתח Powershell על המכונה ונריץ את הפקודה הבאה:

```
Connect-AzAccount -Identity
```

כעת התעניינתי איך נראה ה-Token שחוזר מאותנטיקציה שכזו, כידוע, יש לי חיבה עזה ל-Token-ים. אז תהיתי לעצמי, האם יהיה לנו refresh token? לכמה זמן יהיה תקף? מה ההרשאות שיהיו כתובות בו, מה ה-scope? וכו'.

אז הנה ה-Token שלנו!


```
{ "typ": "JWT",
 "alg": "RS256",
 "x5t": "MGLqj98VNLoXaFfpJCBpgB4JaKs",
 "kid": "MGLqj98VNLoXaFfpJCBpgB4JaKs"
}.
{"aud": "https://management.azure.com/",
 "iss": "https://sts.windows.net/eba5ea9a-b43a-4fab-af1d-f245e133078d/",
 "iat": 1720866122,
 "nbf": 1720866122,
 "exp": 1720952822,
 "aio": "E2dgYFgatTRLqeZzTkV17vLTBR8eAAA=",
 "appid": "f5c8f961-4667-4a12-afdc-e58090afffd9",
 "appidacr": "2",
 "idp": "https://sts.windows.net/eba5ea9a-b43a-4fab-af1d-f245e133078d/",
 "idtyp": "app",
 "oid": "8c7d7ffb-cd90-4d9e-beca-b73e60ca3f01",
 "rh": "0.ARMBmuql6zq0q0-vHfJF4TMHjUZIf3kAutdPukPawfj2MBMTAQA.",
 "sub": "8c7d7ffb-cd90-4d9e-beca-b73e60ca3f01",
 "tid": "eba5ea9a-b43a-4fab-af1d-f245e133078d",
 "uti": "xqdVPppWUEa7Iq_f5eqnAA",
 "ver": "1.0",
 "xms_idrel": "22 7",
 "xms_mirid":
"/subscriptions/GUID/resourcegroups/sapir_resource_group/providers/Microsoft.Compute/vi
rtualMachines/sapir-vm",
 "xms_tcdt": "1720437592"
}.[Signature]
```

מחשבות? נראה שאנחנו מסתכלים על Access Token. לאחר שהמרתי את ה-expiration time לזמן קריא, התברר שה-Token הזה תקף ליום, זה מעט שונה מ-AT של משתמשים, שכן הוא תקף לבערך שעה. בנוסף, אין לנו refresh token, ה-MI שלנו יודע לחדש את ה-Token שלו פעם ביום לבד. השדה xms_mirid, מציג לנו את המשאב שקשור ל-MI הזה, ממנו ביקשתי את ה-Token. נחמד מאוד ☺, למדנו עוד משהו.

אגב, יש הבדל קטן בפקודה כאשר רוצים להתאמת עם UAMI, צריך לפרט את ה-Id של ה-MI. אצרף קישור בסוף המאמר לגיטהאב עם כל מה שצריך!

והינה בונוס: הפקודות האלו ידפיסו לכם הכל יפה עם הרבה אינפורמציה:

```
$currentAzureContext = Get-AzContext
$azureRmProfile =
[Microsoft.Azure.Commands.Common.Authentication.Abstractions.AzureRmProfileProvider]::I
nstance.Profile;
$profileClient = New-Object
Microsoft.Azure.Commands.ResourceManager.Common.RMProfileClient($azureRmProfile);
$profileClient.AcquireAccessToken($currentAzureContext.Subscription.TenantId).AccessTok
en;
Write-Host "$($profileClient)"
```

דרך נוספת להשיג Token עם MI, היא להריץ את הפקודה הבאה: (שמאחורי הקלעים היא מה ש-connect az עושה):

```
$response = Invoke-WebRequest -Uri
'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-
01&resource=https://management.azure.com/' -Method GET -Headers @{Metadata="true"}
```

חשוב לציין, בכל משאב הדרך לביצוע אותנטיקציה היא מעט שונה. כאן אנו רואים הזדהות דרך מכונה וירטואלית, כפי שניתן לראות, מתבצעת בקשה קלאסית לקבל Oauth token, רק שבמקום לפנות ל-Tenant, אנחנו פונים לכתובת הסטטית הבאה: 169.254.169.254.

זו היא הכתובת של שרת ה-IMDS. זה בגדול השרת שאחראי על ניהול המכונות הוירטואליות ב-Azure, ואנחנו רואים אותו, כי הרצתי את הפקודה הזו על מכונה וירטואלית.

אני חושבת שעברנו כברת דרך עד כה, אז בואו נסכם שוב:

1. אפשר להתאמת בקלות עם MI על המשאב שקשור אליו
2. אין שום התעסקות מול הסיסמא של MI
3. ה-Token של MI תקף ליום שלם, ואין לו Refresh token

המחשבה האופנסיבית המתבקשת

אם השגתי גישה למשאב בצורה כזו או אחרת, אני לא צריך לדעת כלום מעבר לפקודת PS אחת, על מנת לבצע אותנטיקציה ב-tenant שהמשאב הזה נמצא בו, ולהשיג מידע!

כמובן שיש פה כמה נקודות שחשוב לציין:

- אני צריך גישה ראשונית למשאב
- אני צריך שעל המשאב יהיה מקונפג MI
- אני צריך של MI יהיו את ההרשאות המתאימות להשיג את המידע שאני רוצה להשיג

ובכל זאת, האם זה לא תרחיש סביר? אני בטוחה שקיימים הרבה ארגונים שיש להם MI עם הרבה יותר הרשאות משהו צריך, וכמובן, שהישויות האלו לא מנוטרות כמעט, בטח שלא כמו משתמשים חזקים, אם כי אין שום הבדל ביניהם!

בעצם, יש הבדל גדול, כדי להתאמת עם MI אתה לא צריך סיסמא, או MFA, אתה רק צריך גישה למשאב!

אז בונוס: סקריפט בטיחות, בדקו את ה-MI בסביבה שלכם, לאילו משאבים הם מקושרים, ומה ההרשאות שלהם, אולי יש כאלו שאפשר למחוק? לרדד הרשאות? או פשוט לנטר טוב יותר (ונדבר על פתרונות ניטור בהמשך):

```
az login
$sps = az ad sp list --filter "servicePrincipalType eq 'ManagedIdentity'" |
ConvertFrom-Json foreach($sp in $sps)
{
Write-Host "Managed Identity:`r`n Id: $($sp.id)`r`n Name: $($sp.displayName)"
$rbac = az role assignment list --assignee $sp.id | ConvertFrom-Json foreach($r in
$rbac)
{
 Write-Host "RBAC:`r`n $($r.roleDefinitionId)`r`n $($r.roleDefinitionName)`r`n
$($r.scope)"
}
Write-Host "`r`n==========================================================`r`n"
}
```

דוגמא לפלט:

```
Managed Identity:
 Id: 8c7d7ffb-cd90-4d9e-beca-b73e60ca3f01
 Name: sapir-vm
RBAC:
 /subscriptions/84945766-aaff-413d-bf94-2e9743f7693a/providers/Microsoft.Authorization/roleDefinitions/
 Reader
 /subscriptions/84945766-aaff-413d-bf94-2e9743f7693a

 ==========================================================

Managed Identity:
 Id: 77b59ebd-c98b-4a3a-a39c-30a5c0ccfbf4
 Name: sapir_ua_mi

 ==========================================================
```

מחקרים!

אז חשוב לציין, אחוז ממש גדול מהמחקרים האלו, חדש ברמות! (ממש הוצגו ב-BlackHat באוגוסט האחרון). זה מאוד מרגש, ואני חושבת שזה רק מתחיל, ונראה עוד ועוד תקיפות ומחקרים על הנושא.

נתחיל ממחקר שפורסם על ידי **Dirk-jan** (ביולי האחרון!):

https://dirkjanm.io/persisting-with-federated-credentials-entra-apps-managed-identities/

במחקר הזה היו 2 מטרות:

1. להצליח להתנער מהעובדה שניתן להזדהות עם MI רק דרך משאב שהוא מושם עליו
2. להבין מה הסיסמא/תעודה של ה-MI

בגדול אפשר להניח שסעיף 1 מכיל את סעיף 2 בתוכו, שכן אם הגעת למצב שאתה מזדהה עם MI לא מתוך המשאב שהוא מקושר אליו, כנראה השגת את הפרטי הזדהות שלו.

כדי להבין את המחקר הזה צריך טיפה ידע במה שנקרא Federated Credentials, מי שמגיע מהעולם של AWS SSO כנראה מכיר כבר את הקונספט.

Federated Credentials

בגדול הרעיון הוא לסמוך על Identity Provider אחר שיאמת את המשתמשים שלך. לצורך העניין, אם יש לי משתמש ב-Okta ואני רוצה להזדהות איתו ל-Entra, אני יכול לקנפג Federated Credentials כך שבסופו של דבר המשתמש שלי יזדהה רגיל מול Okta, ו-Okta בגדול ישלח ל-Entra משהו כמו "אימתתי אותו הוא סבבה". מפה ניהול ההרשאות של המשתמש נעשה כבר ב-Entra והכל טוב ויפה, אבל ההזדהות הראשונית מתבצעת דרך ה-Identity Provider שהחלטתי שאני סומך עליו.

בקטע יותר טכני: ה-Idp שאנחנו סומכים עליו ייצר מפתח פרטי וציבורי, כך שאת הפרטי ישמור אצלו את הציבורי ישלח ל-Entra. בכל פעם שהוא יאמת משתמש, הוא יחתום על ה"אימתתי אותו הוא סבבה" עם המפתח הפרטי שלו.

כך Entra יכולה לפענח את ההודעה (משום שיש לה את המפתח הציבורי), והיא יכולה להיות בטוחה שההודעה הגיעה מה-Idp הזה, שכן רק לו יש את המפתח הפרטי שלו. זה הוא קונספט שאנחנו כבר די מכירים, מזכיר את ה-CBA Authentication מהמאמר הקודם שלי על Passwordless Authentication.

מה הרעיון שהוצג במחקר? ליצור מיני Idp משלנו, ולהגדיר Federated Credentials על User Assigned Managed Identities. למה דבר כזה בכלל אפשרי? לא ברור.

החלק שבעצם מעניין במתקפה הזו היא שהגיוני שנוכל לקנפג Federated Authentication על משתמשים וישויות אחרות, אבל דווקא ב-MI זה מאוד מוזר, שכן חלק נכבד מהקונספט של ישות מסוג זה הוא העובדה שאין לנו שום שליטה על הסיסמא שלה!

לצורך העניין, כאשר אני מנסה להוסיף תעודה או סיסמא ל-Service Principal שמייצג את ה-MI אני מקבלת שגיאה שאומרת שהדבר לא אפשרי:

```
PS C:\> Add-MgServicePrincipalPassword -ServicePrincipalId "77b59ebd-c98b-4a3a-a39c-30a5c0ccfbf4" -BodyParameter $params
Add-MgServicePrincipalPassword_Add: Managed and social identities cannot be modified

Status: 400 (BadRequest)
ErrorCode: Request_BadRequest
Date: 2024-07-13T15:43:59

Headers:
Cache-Control              : no-cache
Vary                       : Accept-Encoding
Strict-Transport-Security  : max-age=31536000
request-id                 : 5dd2ae3c-c4b6-44b6-b8f5-60e23c8565a6
client-request-id          : 66c5e0b1-c2da-436b-8fd8-bda95f670e9d
x-ms-ags-diagnostic        : {"ServerInfo":{"DataCenter":"UAE North","Slice":"E","Ring":"2","ScaleUnit":"000","RoleInstance":
                             "DX1PEPF00000427"}}
x-ms-resource-unit         : 1
Date                       : Sat, 13 Jul 2024 15:43:58 GMT
```

אז ה"פרצה" המעניינת פה, היא היכולת לגעת בסיסמא של הישות MI באמצעות הוספה של Federated Credentials.

מה המצרכים למתכון הזה?

קודם כל אנחנו צריכים שיתקיימו מספר דברים על מנת שהמיני idp שלנו יעבוד. האמת היא, שתוכלו לקרוא על זה קצת יותר במאמר הראשון שלי על Token-ים. שכן פריטים אלו הכרחיים לצורך תהליך האותנטיקציה ב-OIDC סטנדרט:

- אנחנו צריכים קובץ מסוג well-known/openid-configuration.
- וקובץ מפתחות שינותב דרך הערך jwks_uri

מסמך המפתחות יכיל את המפתח הציבורי.

כעת, אנחנו צריכים לקנפג Federated Credentials עבור ה-MI שלנו, לשם כך אנו צריכים role עם ההרשאה הבאה:

```
Microsoft.ManagedIdentity/userAssignedIdentities/federatedIdentityCredentials/write
```

שימו לב שלא מדובר כאן ב-Entra Permissions, אלא ב-RBAC. שזה תמיד מוזר לי כי בסוף MI הוא ישות שמנוהלת חצי ב-Entra וחצי ב-Azure management. בכל מקרה ה-RBAC הבאים יספקו את ההרשאה הנחוצה:

- Contributor / Owner
- Managed Identity Contributor
- Azure Red Hat OpenShift Federated Credential Role

לאחר הקנפוג נוכל להתאמת עם ה-MI מכל מקום שנרצה! חשוב לציין שהמתקפה לא תעבוד על System-Assigned Managed Identity אלא רק על User-assigned, שכן System-assigned תמיד קשור (bound) רק למשאב אחד ויכול להתאמת רק ממנו.

עבודת המגן

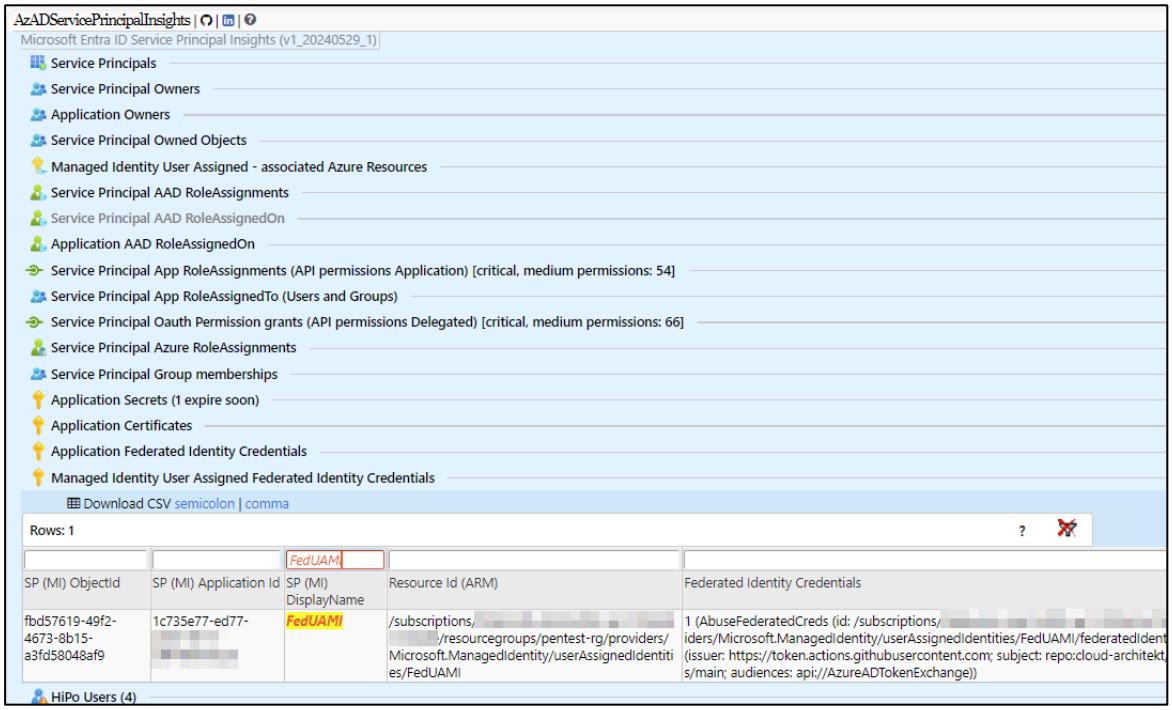
אז אם להיות כנה, הקונספט של תקיפות על MI הוא די חדש ועד כמה שאני יודעת לא באמת נראה בשטח יותר מידי עד כה. אני מניחה שזה נובע ממספר סיבות:

1. גם תוקפים פחות הכירו את הקונספט (ודווקא עכשיו זה התפרסם יותר עם 3 מאמרים והרצאה ב-DefCon האחרון מאת Karl Fosaaen- קישור בסוף המאמר).
2. הרבה מהתקיפות שם בחוץ הן להוסיף secret לאפליקציה (אחרי פישינג למשתמש עם הרשאות מתאימות) ובינינו זה הרבה יותר קל
3. זה שלא זיהו מתקפה כזו עד עכשיו, לא אומר שהיא לא קיימת

אז למרות הסיבות הללו, אני עדיין אציע כאן מספר רעיונות כדי לשים עין על Managed identities.

כלים:

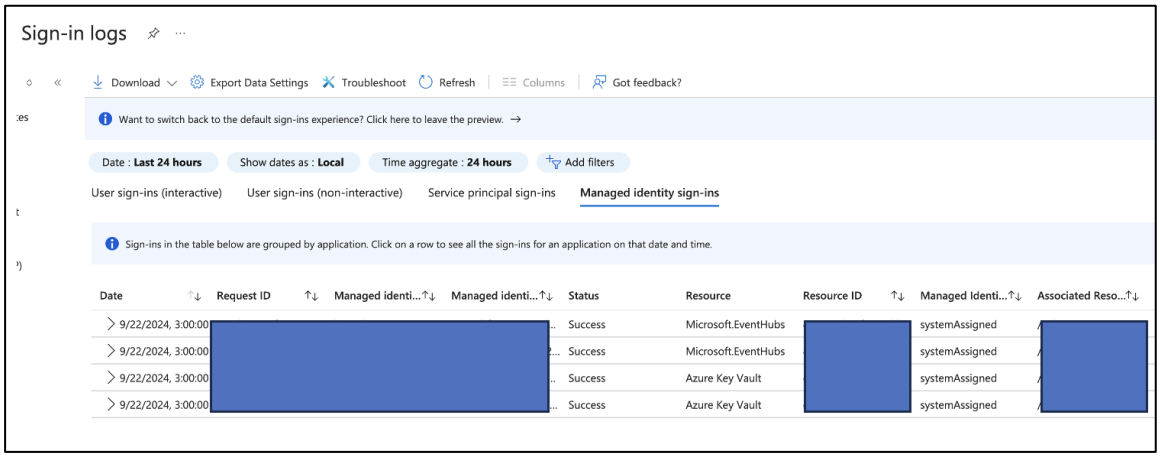
יש כלי ממש נחמד שמביא הרבה מאוד אינפורמציה בצורת דוח. הכלי מביא גם כל מיני abuseים אפשריים ומידע (גם על ישויות וגם על Federated Credentials לדוגמא):



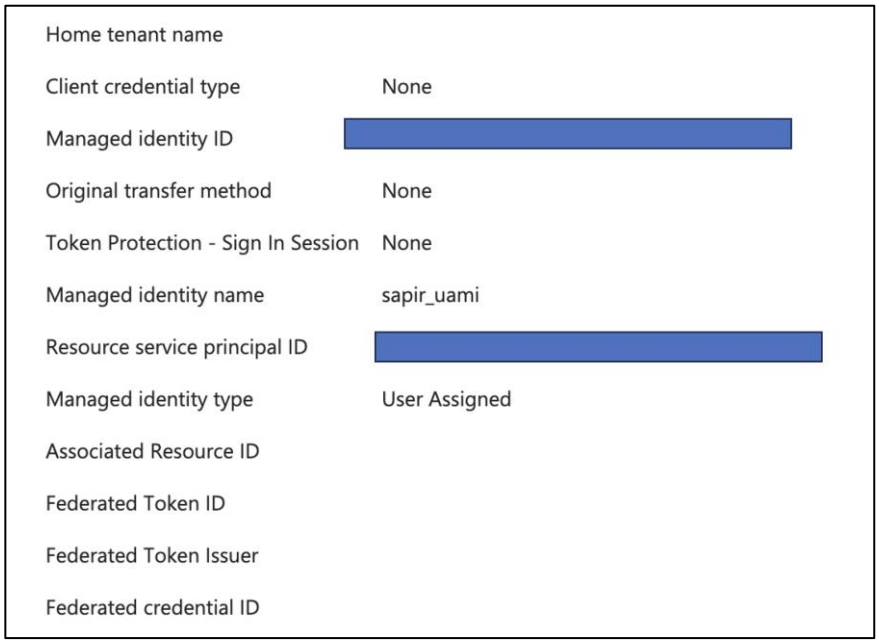
מעבר למתקפה על Federated Credentials, דיברנו על כך שגישה למשאב שיש עליו MI עם הרשאות גבוהות יכולה להוביל להסלמת הרשאות משמעותית, זיהוי של דבר כזה הוא משמעותית יותר קשה, בואו נראה מה נוכל לעשות לגבי זה.

ב-**Azure** יש 3 סוגים לוגים, נעבור על כולם:

1. **Sign-In logs** -לא יודעת למה, אבל Azure ממש הוסיפו קטגוריה נפרדת עבור MI Sign-In , אפשר ממש לראות את זה ב-GUI:

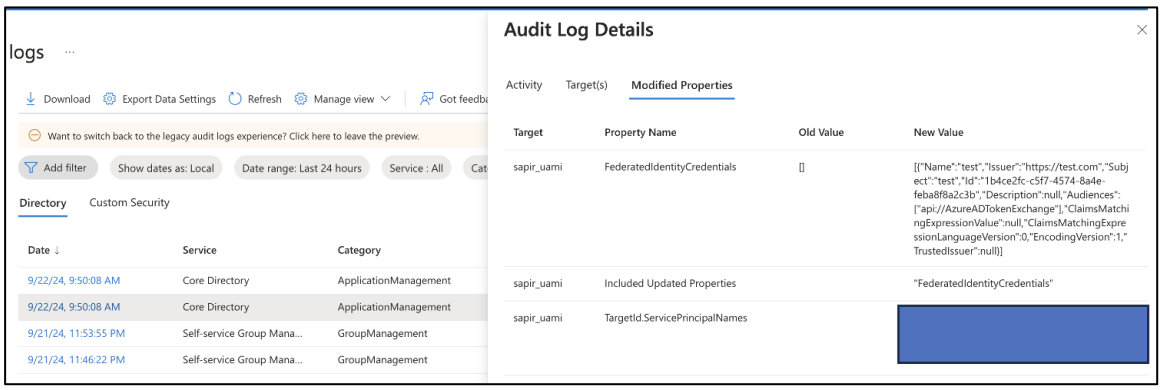


נוכל להשתמש בשדה Client credential type בשביל לזהות אותנטיקציה עם Federated Credentials. כמו כן, נוכל לראות את השדות Federated*, במידה ותוקף קנפג אותם, נוכל להמשיך בחקירה באמצעות זיהוי ה-Token issuer:



לצערי, לא מצאתי לוג MI עם מידע על Location לדוגמא, שהיה יכול להיות נחמד במקרה של תקיפה. כנראה שזה משום שה-MI עצמו רץ בתוך ה-Resource שקנפגתי אותו עליו.

2. **Audit log** - קינפגתי Federated Credentials ל-UAMI שלי, וקיבלתי עליו לוג:



הלוג עצמו הוא מסוג "Update service principal". סך הכל אני חושבת שאין סיבה שלא לנטר תרחיש כזה, שכן הוא לא אמור לקרות. רק חשוב לציין שיש לבדוק שה-ServicePrincipal הוא באמת MI אחרת זה כן עלול להביא FP.

3. **GraphActivityLog** - אני לא אכנס יותר מידי לסוג הלוגים הזה, מכמה סיבות:
   - הוא דורש קונפיגורציה שעולה כסף (לעומת האחרים שלא דורשים)
   - שיטת התשאול בלוג הזה הוא באמצעות שאילתות KQL (לא שיש להן syntax מסובך במיוחד)

עניין שחשוב לציין לגבי לוג זה, הוא שהלוג מתעד כל שאילתת GraphAPI שנעשית ב-Tenant. זה קצת מזכיר לנטר LDAP בסביבת AD. אני קצת לא אוהבת את הלוג הזה כי הוא גורם לך לחשוב שיש לך משהו חזק ביד, כשבפועל אתה רואה רק פעולות שקורות ב-Entra, שכן אלו ה-API ש-GraphAPI מיחצן, ולא רואה דברים שקורים בעולם הגדול של Azure. בנוסף, יש המון שיטות להתחמק מהלוג הזה, שכן יש דומיינים מקבילים שמיחצנים API שניתן לבצע באמצעותם את אותן הפעולות, ולא להרשם ללוג.

הדוגמא הכי קלאסית - GraphAPI לעומת AzureADAPI:

GraphAPI - https://graph.microsoft.com

AzureADAPI - https://graph.windows.net

השני כביכול נחשב deprecated ובשימוש פנימי של מיקרוסופט, כאשר בפועל, רוב כלי התקיפה (כדוגמאת RoadTools ו-AADInternals) משתמשים בעיקר בו, ולכן לא ניתן לזהות אותם באמצעות ה-GraphActivityLog.

בקיצור, אני לא חובבת גדולה של הלוג הזה, אבל אם במקרה יש לכם אותו, ובמקרה יש לכם MI שמתשאלת דברים ב-Entra, כן תוכלו לזהות את זה.

האם זה תרחיש נפוץ או בכלל מעניין? אני באמת לא בטוחה, אבל אם יש detection-ים שניתן לכתוב כך שיצרו שום False Positives? - אז למה לא?

באמצעות השדה ServicePrincipalId נוכל לזהות את ה-MI שלנו:

```
1 MicrosoftGraphActivityLogs
```

Results Chart

Showing the first 30,000 results. Learn more on how to narrow down the result set.

TimeGenerated [UTC]	Location	RequestId	OperationId

RequestMethod	GET
ResponseStatusCode	200
AadTenantId	[redacted]
IPAddress	[redacted]
UserAgent	ReactorNetty/1.0.32
RequestUri	https://graph.microsoft.com/v1.0/auditLogs/signIns?$filter=(createdDateTime+gt-
DurationMs	10427337
ResponseSizeBytes	92
SignInActivityId	7ZkOeWAp9kaJX5BtX6QCAA
Roles	Directory.Read.All AuditLog.Read.All
TokenIssuedAt [UTC]	2024-09-22T06:29:08Z
AppId	[redacted]
ServicePrincipalId	[redacted]
IdentityProvider	https://sts.windows.net/[redacted]
ClientAuthMethod	1

אז אם כבר יש לכם את הלוג הזה, הייתי מוסיפה תשאול אם במקרה השדה הזה מכיל ID של MI. כי אם כן, זה בהחלט קצת מוזר.

באופן כללי, אני חושבת שהדרך הטובה ביותר להגן מפני הסלמת הרשאות עם MI היא פשוט ניהול נכון של הרשאות ותמיכה בעקרון ה-least privileges, וכן, זה ממש קלישאתי, אבל תפסיקו לתת הרשאות מוגזמות רצח סך הכל אתם רוצים פונקציה שתדליק אוטומטית מכונה וירטואלית לא צריך לתת לה הרשאות ניהול מלאות על כל המכונה עכשיו. באמת שזה ימנע הרבה צרות בעתיד ☺

סיכום

MI הוא רק תת-תחום של קונספט חדש לגמרי שתופס תאוצה בזמן האחרון בעולם הענן. והוא התחום של הישויות המנוהלות בענן. אמנם זה עוד (יחסית) חדש , אבל אני בטוחה שעוד נראה דברים מעניינים שם.

ואין כמו להזכר תמיד שיש עולם שלם שם בענן, וכל פיצ'ר קטן שלו יכול בקלות למלא 20 עמודים ב-DigitalWhisper ☺

על המחברת

@sapirxfed. ביום יום עושה שטויות ב-CrowdStrike, ובהפסקות עושה retweet לכל דבר שקשור ל-AD או ל-AAD (טוב, ולאחרונה גם AWS וסתם דברים רנדומלים של ענן... ☺)

אוהבת לכתוב קוד, מנסה למצוא חולשות, ומעריצה שרופה של DigitalWhisper!

ביבליוגרפיה

- סרטון מעולהההה על MI:

https://www.youtube.com/watch?v=rC1TV0_sIrM

בכללי כל הסרטונים של הבחור הזה הם זהב טהור.

- עמוד גיטהאב עם כל מיני פקודות של MI:

https://github.com/johnthebrit/RandomStuff/blob/master/ManagedIdentity/mngIdDD.ps1

- המאמר של Dirk-jan:

https://dirkjanm.io/persisting-with-federated-credentials-entra-apps-managed-identities/

- מאמר על הגנות על המתקפה ש-Dirk-jan מצא וגם כל מיני רעיונות לזיהוי:

https://www.cloud-architekt.net/identify-prevent-abuse-uami-fedcreds/

- דוקומנטציה של מיקרוסופט תמיד טוב שיש:

https://learn.microsoft.com/en-us/entra/identity/managed-identities-azure-resources/overview

- כל מיני דברים Karl Fosaaen פרסם על MI, ממש מעניין:

https://www.netspi.com/blog/technical-blog/cloud-pentesting/extracting-managed-identity-certificates-from-azure-arc-service/

https://www.youtube.com/watch?v=efF5Up7zBrg

https://www.netspi.com/blog/technical-blog/cloud-pentesting/azure-user-assigned-managed-identities-via-deployment-scripts/