TEAM AXON

HUNTERS

MASTERING AZURE MANAGED IDENTITIES:

Attack & Defense, Part 2

Research Paper

TABLE OF CONTENTS

3	Introduction
4	Identifying Azure Managed Identities
11	Threat Hunting Managed Identity Abuse
41	Incident Investigation & Response
54	Summary
55	About Hunters

Introduction

In **part one** of this series, we explored the foundational aspects of Azure Managed Identities (MIs), focusing on their benefits, abuse scenarios, and the expanding attack surface they present. Specifically, we examined how MIs configured on Azure Virtual Machines (VMs) can be exploited to access Azure resources, escalate privileges, and target endpoints within Azure Entra ID and Microsoft 365. While MIs are designed to streamline credential management and enhance security, these scenarios highlight their potential misuse when not adequately monitored or secured.

This second part shifts focus to proactive defense. Building on the abuse scenarios and the potential blast radius of a compromised MI, we'll delve into threat-hunting methodologies and detection strategies that security teams can use to uncover signs of MI abuse.

Recent research by NetSPI — particularly the work of Karl Fossaen and his DEF CON 32 talk "**Identity Theft is Not a Joke, Azure!**", was instrumental in raising awareness around MI abuse. Their contributions, alongside other researchers in the community, helped frame the importance of not only understanding these attack paths but also building reliable detection and investigation techniques around them. This blog builds on that momentum - focusing less on the offensive techniques and more on the defensive strategies available to security teams today.

By analyzing attack patterns, auditing MI configurations, and leveraging native Azure monitoring tools, organizations can identify vulnerabilities and mitigate risks before they escalate. This guide is not just informative — it's actionable. It provides practical insights to help defenders stay ahead of evolving threats and secure their environments against the misuse of MIs.

Identifying Azure Managed Identities

To effectively hunt for threats involving abuse of Managed Identities (MIs), accurately identifying these identities within your Azure environment is crucial. Given a specific user, how can we confirm it is a MI and not another Azure user type?

Generally, there are three primary methods for identifying MIs:

1. Reviewing the Azure Portal
2. Querying Azure Resources
3. Examining Azure Logs

Review Azure Portal

In the Azure Management Portal (Azure Portal), we can find System-Assigned Managed Identities (SAMIs) by inspecting the resources that support them, such as virtual machines, app services, and other Azure services. However, there isn't a dedicated section that directly lists all SAMIs across the subscription.

It's simpler for user-assigned managed identities (UAMIs), as all UAMIs are listed on the “Managed Identities” page.

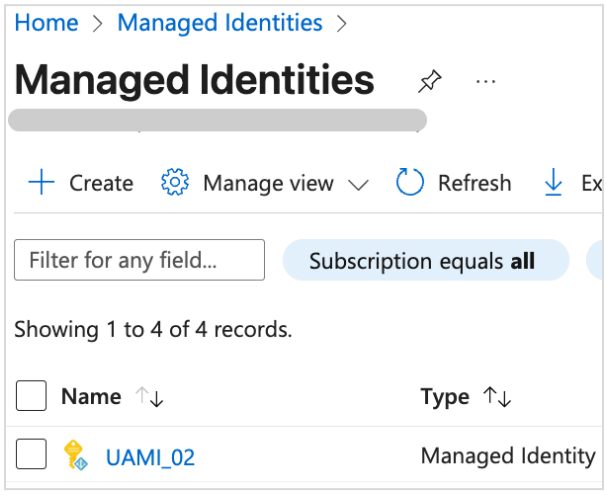

Figure 1 - Managed Identities page on Azure portal

Query Azure Resources

While the Azure Portal doesn't provide a sufficient way to review SAMIs, querying Azure using tools like Azure CLI offers more flexibility.

After installing Azure CLI and authenticating with a user with list permissions, we can use the following PowerShell Script to list all MIs, categorized into MI types (SAMI/UAMI).

```shell
$sps = az ad sp list --filter "servicePrincipalType eq 'ManagedIdentity'" --output json |
ConvertFrom-Json
$sps | Select-Object -Property displayName, Id, servicePrincipalType, appId, `
  @{Name="ManagedIdentityType"; Expression={
      $identityType = ($_.alternativeNames | Where-Object { $_ -match "isExplicit=(\w+)" }
| ForEach-Object {
        if ($matches[1] -eq "True") {
          "UAMI"
        } elseif ($matches[1] -eq "False") {
          "SAMI"
        } else {
          "Unknown"
        }
      })
      if ($identityType) {
        $identityType
      } else {
        "Unknown"
      }
    }} | Format-Table -AutoSize
```

We can also use the following one-liner to present only a specific MI type, e.g., SAMI:

```shell
az resource list --query "[?identity.type=='SystemAssigned'].{Name:name,
principalId:identity.principalId}" --output table
```

Examine Azure Logs

In addition to Azure Portal and CLI-based querying, the different Azure logs might also shed light on the existence of MIs. Using logs to identify or map MIs can be valuable in the following use cases:

- **Lack of permissions to access Azure Portal or query Azure CLI.**
  For example, the organization's SOC analysts and third-party providers such as MSSPs or IR retainers typically lack enough permissions, so they mainly rely on the logs.
- **Deleted MIs**
  In cases where the MI in scope was deleted, we won't find it on Azure Portal or using CLI, leaving us with the logs as a last resort.

MI authentication events under Azure Sign-Ins logs

The Azure Sign-Ins logs have a dedicated category for MI-related sign-ins. On the Azure Portal, it's presented as "Managed identity sign-ins":

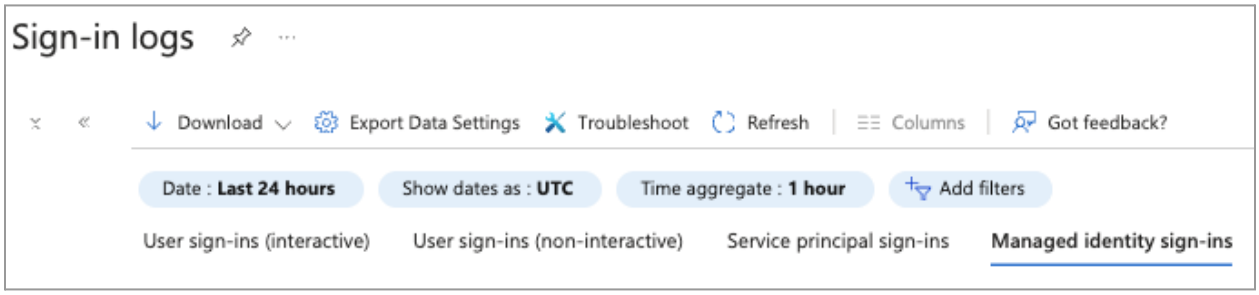

Figure 2 - Managed identities sign-in tab under Azure Sign-in logs

Azure Sign-In logs help identify MIs but do not differentiate between System-Assigned and User-Assigned MIs. Azure Audit and Azure Activity logs are more suitable for creating a comprehensive mapping of all MIs.Yet, it's important to remember that sometimes not all log types are available and have sufficient retention, so Azure Sign-In logs could partially complete the picture.

MI creation events under Azure Audit logs

Creating a new SAMI and UAMI would trigger *"Add service principal"* under Azure Audit logs.

How can we differentiate between the creation logs of SAMI and UAMI?

For UAMI, the modified property *ManagedIdentityResourceId* indicates the identity is **user-assigned** (*UAMI_02* is the name we gave for the newly created UAMI):

> /subscriptions/<subscription-id>/resourcegroups/<resource-group>/providers/Microsoft.ManagedIdentity/**userAssignedIdentities**/UAMI_02

While for SAMI, the modified property *ManagedIdentityResourceId* indicates the identity is attached to a VM resource:

> /subscriptions/<subscription-id>/resourcegroups/<resource-group>/providers/Microsoft.Compute/**virtualMachines**/sample_vm_name

We use the following Snowflake (SF) query to identify a SAMI/UAMI creation using Azure Audit, and write the results into a new SF table: *managed_identities_inventory*.

That table would be later used for hunting queries.

Note: Throughout the research, we used SF as our database. Of course, the queries can be implemented over any database, with the required syntax adjustments.

```sql
SELECT
  MIN(event_time)                                    AS first_seen,
  MAX(event_time)                                    AS last_seen,
  inner_f.value:newValue::string                     AS managed_identity_resource,  -- Parse newValue to ARRAY
  outer_f.value:displayName::string                  AS managed_identity_name,
  outer_f.value:id::string                           AS managed_identity_id,
  -- Determine Managed Identity Type
  CASE WHEN managed_identity_resource ILIKE '%userAssignedIdentities%' THEN 'UAMI'
    ELSE 'SAMI'
  END                                                AS managed_identity_type,
  'Azure Audit: Creations of new managed identities'  AS source_description
FROM
```

```
  RAW.AZURE_AUDIT,
  LATERAL FLATTEN(input => PARSE_JSON(properties_target_resources)) AS outer_f,  -- Flatten the outer JSON array
  LATERAL FLATTEN(input => outer_f.value:modifiedProperties) AS inner_f  -- Flatten the inner modifiedProperties array
WHERE operation_name = 'Add service principal'
    AND inner_f.value:displayName::string = 'ManagedIdentityResourceId'
    AND properties:identity::string = 'Managed Service Identity'
    AND event_time > CURRENT_TIMESTAMP - INTERVAL '180 days'
GROUP BY managed_identity_resource,
         managed_identity_name,
         managed_identity_id,
         managed_identity_type
```

Tips:

- We use lateral flatten to overcome the nested pattern of fields in the log.
- Consider changing the time-based restriction for environments with high log volume to maintain query performance and manage table size.

MI creation events under Azure Activity logs

On Azure Activity, if the field `xms_mirid` (Managed Identity Resource Identifier) is not null, it indicates the use of a MI. We use the following SF query for that:

SQL

```
SELECT MIN(event_time)                                                                  AS first_seen,
       MAX(event_time)                                                                  AS last_seen,
       identity:claims:xms_mirid::string                                                AS managed_identity_resource,
       SPLIT_PART(managed_identity_resource, '/', -1)                                   AS managed_identity_name,
       identity:claims:"http://schemas.microsoft.com/identity/claims/objectidentifier"::string AS managed_identity_id,
       -- Determine Managed Identity Type
       CASE WHEN managed_identity_resource ILIKE '%userAssignedIdentities%' THEN 'UAMI'
        ELSE 'SAMI'
       END                                                                              AS managed_identity_type,
       'Azure Activity: Operations that were initiated by managed identities'           AS source_description
FROM RAW.AZURE_ACTIVITY
WHERE managed_identity_resource IS NOT NULL
  AND event_time > CURRENT_TIMESTAMP - INTERVAL '180 days'
GROUP BY managed_identity_resource,
         managed_identity_name,
         managed_identity_id,
         managed_identity_type
```

We understand that both log types, Azure Audit and Azure Activity, can indicate the existence and usage of MIs and assist us in categorizing each into SAMI or UAMI. Yet, every log type has its disadvantage that prevents it from comprehensively mapping MIs.

Azure Audit can assist in mapping of MIs by their creation event. However, it's susceptible to visibility gaps due to insufficient log retention.

Azure Activity, on the other hand, can assist in mapping MIs by their ongoing activities. However, it won't cover cases of inactive MIs.

As a result, creating a unified table based on the two queries we've created above is recommended to achieve a comprehensive mapping of MIs using logs.

```SQL
CREATE TABLE investigation.managed_identities_inventory AS
SELECT
  MIN(event_time)                                          AS first_seen,
  MAX(event_time)                                          AS last_seen,
  -- Parse newValue to ARRAY
  inner_f.value:newValue::string                           AS managed_identity_resource,
  outer_f.value:displayName::string                        AS managed_identity_name,
  outer_f.value:id::string                                 AS managed_identity_id,
  -- Determine Managed Identity Type
  CASE WHEN managed_identity_resource ILIKE '%userAssignedIdentities%' THEN 'UAMI'
    ELSE 'SAMI'
  END                                                      AS managed_identity_type,
  'Azure Audit: Creations of new managed identities'  AS source_description
FROM
  RAW.AZURE_AUDIT,
  -- Flatten the outer JSON array
  LATERAL FLATTEN(input => PARSE_JSON(properties_target_resources)) AS outer_f,
  -- Flatten the inner modifiedProperties array
  LATERAL FLATTEN(input => outer_f.value:modifiedProperties) AS inner_f
WHERE OPERATION_NAME = 'Add service principal'
    AND inner_f.value:displayName::string = 'ManagedIdentityResourceId'
    AND properties:identity::string = 'Managed Service Identity'
    AND event_time > CURRENT_TIMESTAMP - INTERVAL '180 days'
GROUP BY managed_identity_resource,
```

```
         managed_identity_name,
         managed_identity_id,
         managed_identity_type
UNION ALL
SELECT MIN(event_time)                                          AS first_seen,
       MAX(event_time)                                          AS last_seen,
       identity:claims:xms_mirid::string                        AS managed_identity_resource,
       SPLIT_PART(managed_identity_resource, '/', -1)           AS managed_identity_name,

identity:claims:"http://schemas.microsoft.com/identity/claims/objectidentifier"::string
                                                                AS managed_identity_id,
       -- Determine Managed Identity Type
       CASE WHEN managed_identity_resource ILIKE '%userAssignedIdentities%' THEN 'UAMI'
        ELSE 'SAMI'
       END                                                      AS managed_identity_type,
       'Azure Activity: Operations that were initiated by managed identities'
                                                                AS source_description
FROM RAW.AZURE_ACTIVITY
WHERE managed_identity_resource IS NOT NULL
  AND event_time > CURRENT_TIMESTAMP - INTERVAL '180 days'
GROUP BY managed_identity_resource,
         managed_identity_name,
         managed_identity_id,
         managed_identity_type
```

Notes:

- The new table is written under a different schema (*investigation*), but of course, it can be defined differently depending on the organizational policies.

Threat Hunting Managed Identity Abuse

This section focuses on threat hunting techniques designed to surface potential misuse, lateral movement, and privilege escalation involving MIs. Through hunting queries and analytical patterns, we demonstrate how to leverage various Azure log sources, including Azure Activity Logs, Microsoft Graph API logs, and Entra ID sign-ins, to uncover abnormal behaviors, unauthorized access attempts, and token misuse. These queries aim to detect threats in real time, support historical investigations, and identify deviations from normal behavior across your Azure environment.

While researching different types of MI abuse across services such as Azure Function Apps, VMs, Automation Accounts, and more, we concluded that a service-specific approach is often too narrow. It often introduces irrelevant noise or lacks the necessary telemetry for reliable detection. Instead, the most effective hunting strategies focus on service-agnostic behaviors such as stolen JWT token usage and anomalous activity patterns.

That said, service-specific logs remain highly valuable for investigation and response. While they may not always be actionable for real-time detection, they often provide critical context for validating alerts and understanding the scope of impact. We explore this further in the next section, **Incident Investigation & Response.**

We also found that no single detection logic is sufficient on its own. As a result, we developed a collection of hunting queries -written in Snowflake SQL, that address different abuse scenarios. These logics are designed to be modular, complementary, and adaptable to various threat models. They can also be translated into other query languages, such as KQL, while preserving their core detection logic.

Hunting Queries: At a glance

Click to jump to the relevant query

Hunting Query	Query	Fidelity Level
1	Explicit request for IMDS from a VM with SAMI	High
2	Microsoft Graph Enumeration using Managed Identity	High
3	Sensitive Graph Roles-usage by a Managed Identity	High
4	MI's token request for unusual endpoints	High
5	Unique Token Identifier Usage From Multiple IP Addresses	High
6	MI's mass token types requested	Medium
7	Managed Identity Activities from Non-Azure IP Addresses	Medium
8	Unusual Action Types by a Managed Identity	Medium
9	Attached UAMI used from a New Azure Resource	Medium
10	MI performs activity on Entra ID	Medium
11	SAMI activity from abnormal IP	Low
12	MI accesses unusual resources	Low

Hunting Query 1 - Explicit request for IMDS from a VM with SAMI:

Thesis

MIs request tokens from IMDS frequently as part of the regular operations. Threat actors who gain control over a VM with an attached SAMI may try to request a token explicitly using PowerShell/CMD.

To detect such an activity, we look at MI sign-ins and correlate those events with host-based process connection events, based on the name of the SAMI and the hostname, while the destination IP is the IMDS (169.254.169.254), and process name is related to CMD or Script-based tools (e.g., PowerShell.exe, Python.exe).

In this hunting thesis, we correlated with Windows security event 5156. However, other host-based process connection events, such as Sysmon event ID 3 and EDR network events, should also work.

Data source(s)

- "Managed identity sign-ins" category under Azure Sign-Ins logs.
- Windows security event 5156 ("The Windows Filtering Platform has permitted a connection").

Fidelity level

- **High**

Tuning & Observations

- **False Positive use cases:**
  - Although not common, explicit requests of SAMI's tokens can be legitimate. Each organization should examine those use cases, try to exclude them, and reduce FP rates.

- **False Negative use cases:**
  - Focusing on specific processes: The query focuses on particular processes that initiated the connection to the IMDS. Those processes (PowerShell.exe, cmd.exe, etc.) might indicate explicit token requests. Yet, threat actors may

request tokens using different processes that don't match the processes outlined within the query. As a result, an organization should consider removing the processes list and go the other way around - filter out common and legitimate processes that connect to the IMDS, e.g., WindowsAzureGuestAgent.exe, to name a few. That may decrease the FN rate (i.e., increase the coverage) at the cost of increasing the FP rate.

- Visibility gaps: Another FN scenario might be caused by visibility gaps. Event 5156 is known as a verbose log, so many organizations decide to avoid collecting it or collect it only on certain highly sensitive assets. Hence, it's essential to check whether this event is collected widely or consider other alternatives, such as Sysmon event ID 3, EDR network events, etc.
- Focusing on Windows: Choosing event 5156 as the host-based process connection event limits us to Windows VMs. Using EDR network events may help extend the coverage of this query.

Query

```SQL
SELECT azure_signin.event_time,
    azure_signin.category,
    azure_signin.properties:servicePrincipalId::string                    AS managed_identity_id,
    azure_signin.properties:servicePrincipalName::string                  AS managed_identity_name,
    azure_signin.caller_ip_address,
    azure_signin.properties_user_principal_name,
    azure_signin.properties_is_interactive,
    azure_signin.properties_resource_display_name::string                 AS target_endpoint_name,
    azure_signin.properties_resource_id::string                           AS target_endpoint_id,
    wel.event_time,
    wel.event_id,
    wel.machine_name,
    SPLIT_PART(wel.machine_name, '.', 1)                                  AS computer_name,
    wel.message_details:application_information:application_name::string   AS process_name,
    wel.message_details:application_information:process_id::string         AS process_id,
    wel.message_details:network_information:source_address::string        AS source_address,
    wel.message_details:network_information:source_port::string           AS source_port,
    wel.message_details:network_information:destination_address::string    AS destination_address,
    wel.message_details:network_information:destination_port::string       AS destination_port,
    wel.message_details:network_information:direction::string             AS direction,
    wel.message_details:network_information:protocol::string              AS protocol
FROM RAW.AZURE_SIGNIN AS azure_signin
LEFT JOIN RAW.UNIVERSAL_WEL AS wel
```

```
ON azure_signin.properties:servicePrincipalName::string = SPLIT_PART(wel.machine_name, '.', 1)
WHERE azure_signin.category = 'ManagedIdentitySignInLogs'
    AND azure_signin.event_time > CURRENT_DATE
    AND wel.event_time > CURRENT_DATE - INTERVAL '15 minutes'
    AND wel.event_time BETWEEN azure_signin.event_time - INTERVAL '15 minutes' AND
azure_signin.event_time + INTERVAL '15 minutes'
    AND wel.event_id = 5156
    AND wel.message_details:network_information:direction::string = 'Outbound'
    AND wel.message_details:network_information:destination_address::string = '169.254.169.254'
    AND wel.message_details:application_information:application_name::string ILIKE ANY (
                                        '%\\\\powershell.exe',
                                        '%\\\\cmd.exe',
                                        '%\\\\conhost.exe',
                                        '%\\\\python.exe')
```

Notes:

- Results can be enriched and investigated using a variety of host-based events: Process Creation events (id 4688), PowerShell Operational events (id 4104), EDR logs, Sysmon logs, and more.
- This query monitors the current day. Each organization should consider the timeframe for monitoring (current day, past week/month, etc.).

Hunting Query 2 - Microsoft Graph Enumeration using Managed Identity:

Thesis

This threat-hunting thesis focused on Microsoft Graph, which differentiates it from most of the other theses we mentioned here, focusing on the main Azure log sources (Sign-in, Activity, Audit).

This hunting query looks for potential enumeration conducted using a compromised MI. It is logical to assume that one of the first things a threat actor will conduct after getting unauthorized access to a MI JWT access token is some kind of enumeration.

Different parts of this phase can be conducted using Microsoft Graph, directly using HTTP requests (as demonstrated in part 1 of this series), or by using different tools that use it behind the scenes. This thesis aims to detect both.

Data source(s)

- Microsoft Graph Activity Logs
- Managed Identities Inventory

Fidelity level

- **High**

Tuning & Observations

- Identifying potential enumeration activity in Microsoft Graph can be challenging due to different types of "noise," which most often requires a comparison to a baseline detection method (UEBA). However, from our experience, the results are significantly less noisy when limiting the search to MIs only, and no UEBA usage is required.
- The thresholds mentioned in the query below are adjustable, and we recommend modifying them based on the numbers associated with your Azure/M365 environment.

Query

- In this hunting query, we used a CTE that includes a group of activities that align with potential enumeration characteristics, based on pre-defined thresholds.
- This CTE includes some lookups, including fetching the full request URI, the URI Endpoint Base of each requested URI, and the number of distinct requests by each MI.
- The thresholds (which can be adjusted in case of significantly different "normal" numbers in your organization) are set to identify only cases with significant requests, distinct URIs, and distinct URI endpoint bases accessed by a specific MI over X minutes (we used 1 hour, however it can be modified, for example to 15 minutes, depending on the tested environment).
- We also added the option (which can be uncommented) to filter out URIs that are more likely to be spammy.

```sql
SQL
WITH graph_enum_activity AS (
    SELECT MIN(time)                                                    AS min_event_time,
```

```
            MAX(time)                                                                   AS max_event_time,
            DATE_TRUNC('HOUR', time)                                                    AS hour_of_events,
            user_principal_object_id,
            signin_activity_id                                                          AS token_identifier,
            token_issued_at                                                             AS token_issue_time,
            properties:ipAddress                                                        AS source_ip_address,
            ARRAY_AGG(DISTINCT user_agent)                                              AS
distinct_user_agents,
            ARRAY_AGG(DISTINCT request_uri)                                             AS
distinct_request_uris,
            ARRAY_AGG(
                DISTINCT '/' || REGEXP_SUBSTR(
                    request_uri,
                    '^https://graph.microsoft.com/((v1.0|beta)/[^/()?]+)',
                    1, 1, 'e'
                )
            )                                                                           AS
distinct_endpoint_base,
            ARRAY_AGG(DISTINCT roles)                                                    AS distinct_roles,
            ARRAY_AGG(DISTINCT app_id)                                                  AS distinct_app_ids,
            ARRAY_AGG(DISTINCT response_status_code)                                    AS
distinct_response_codes,
            ARRAY_SIZE(distinct_endpoint_base)                                          AS
amount_of_endpoint_base,
            ARRAY_SIZE(distinct_request_uris)                                           AS
amount_of_request_uris,
            COUNT(*)                                                                    AS amount_of_requests
    FROM RAW.MICROSOFT_GRAPH_ACTIVITY_LOGS
    WHERE time BETWEEN '2025-03-01 19:00:00' AND '2025-04-01 22:30:00'
          AND request_method = 'GET'
          -- In case of any particular spammy requested URIs in your environment, feel free to add them to the list
          AND NOT request_uri ILIKE ANY (
              '%users/delta?$deltatoken%',
              '%/info/logoUrl%'
          )
          AND NOT MICROSOFT_GRAPH_ACTIVITY_LOGS.user_agent ILIKE '%Microsoft Office/%'
    GROUP BY user_principal_object_id,
             token_identifier,
             token_issue_time,
             source_ip_address,
             hour_of_events
    HAVING amount_of_requests > 60
           AND amount_of_endpoint_base > 5
           AND amount_of_request_uris > 30
)
SELECT g.hour_of_events,
       g.user_principal_object_id,
       g.token_identifier,
       g.token_issue_time,
       g.source_ip_address,
```

```
        g.distinct_user_agents,
        g.distinct_request_uris,
        g.distinct_endpoint_base,
        g.distinct_roles,
        g.distinct_app_ids,
        g.distinct_response_codes,
        g.amount_of_endpoint_base,
        g.amount_of_request_uris,
        g.amount_of_requests
FROM graph_enum_activity AS g
LEFT JOIN INVESTIGATION.MANAGED_IDENTITIES_INVENTORY AS managed_identities_inventory
    ON g.user_principal_object_id = managed_identities_inventory.managed_identity_id
WHERE managed_identities_inventory.managed_identity_id IS NOT NULL
```

Hunting Query 3 - Sensitive Graph Roles-usage by a Managed Identity

Thesis

In this thesis, we looked for sensitive graph roles used by MIs. Of course, this doesn't mean that the relevant hits indicate malicious activity, but it does, at the very least, require the attention of an analyst or threat hunter.

Sensitive Graph Roles like `RoleManagement.ReadWrite.Directory`, `AppRoleAssignment.ReadWrite.All`, `Mail.Read`, `Directory.ReadWrite.All`, etc., are not usually used by MIs and are of high interest to potential threat actors. When used, they may indicate malicious activity.

Data source(s)

- Microsoft Graph Activity Logs
- Azure Managed Identities Inventory

Fidelity level

- **High**

Tuning & Observations

- **False Positive use cases:**
  - There might be cases in which other sensitive graph roles were targeted by a threat actor. The list we used in the hunting query above includes roles wer believe are important to track, however there are additional ones that can be interesting and significant. Consider adding additional graph roles of your choice. The following Microsoft documentation can be used for reference: https://learn.microsoft.com/en-us/graph/permissions-reference

Query

This hunting query joins the Microsoft Graph Activity Logs table and the Azure Managed Identity inventory table while looking for Token Roles that are considered sensitive or more likely to be targeted by threat actors. Interestingly, from our experience, those roles are not commonly used by MIs, making any hit that associates a MI with any sensitive roles important enough to be investigated.

SQL

```sql
SELECT graph_activity_table.time,
       graph_activity_table.app_id                                  AS app_id,
       graph_activity_table.user_principal_object_id                AS upn_id,
       graph_activity_table.managed_identity_name,
       graph_activity_table.roles                                   AS token_roles,
       graph_activity_table.request_uri,
       graph_activity_table.properties:wids                         AS token_wids,
       graph_activity_table.properties:ipAddress                    AS ip_address,
       graph_activity_table.token_issued_at                         AS token_issue_time,
       graph_activity_table.request_method                          AS request_method,
       graph_activity_table.signin_activity_id                      AS token_id,
       graph_activity_table.request_uri                             AS request_uri,
       graph_activity_table.user_agent                              AS user_agent,
       graph_activity_table.response_status_code                    AS status_code,
       graph_activity_table.response_size_bytes                     AS size_bytes,
       graph_activity_table.app_id                                  AS app_id,
       graph_activity_table.user_principal_object_id                AS upn_id
FROM RAW.MICROSOFT_GRAPH_ACTIVITY_LOGS                 AS graph_activity_table
JOIN INVESTIGATION.MANAGED_IDENTITIES_INVENTORY        AS managed_identities_inventory
  ON graph_activity_table.upn_id = managed_identities_inventory.managed_identity_id
     AND graph_activity_table.time BETWEEN '2025-01-07 06:30:00' AND '2025-01-07 11:00:00'
     AND graph_activity_table.roles ILIKE ANY (
         '%RoleManagement.ReadWrite.Directory%',
         '%AppRoleAssignment.ReadWrite.All%',
         '%Mail.Read%',
         '%Directory.ReadWrite.All%'
     )
```

Hunting Query 4 - MI's token request for unusual endpoints:

Thesis

MI actions are typically pre-defined and repetitive. Hence, we'd expect the requested endpoints to be relatively static. MIs that suddenly request a token for an endpoint it has never asked for before, or at least not recently, could raise a suspicion that it's misused.

Data source(s)

- "Managed identity sign-ins" category under Azure Sign-Ins logs.

Fidelity level

- **High**

Tuning & Observations

- **False Positive use cases:**
  - The resources the MI is attached to may change their actions, resulting in the endpoints they ask tokens for. Such cases may generate FP leads.
- **False Negative use cases:**
  - This thesis compares a given token request to a historical baseline. If the baseline already includes a token request for the relevant endpoint by the relevant MI, then an anomaly would not be triggered.

Query

Logic: In the hunting query below, we used:

- We create a baseline for `managed_identity_id - target_endpoint_id` pairs and put it under `TOKEN_REQUEST_HISTORY`. Then, we look for a new pair.
- The time periods in this query are arbitrary (e.g., a 120-day learning period). Every organization can modify the times according to its needs and limitations.
- This query monitors the current day. Each organization should consider the monitoring timeframe (current day, past week/month/etc).
- There are several ways to implement this thesis on Snowflake, including using JOIN and PARTITION functionalities. Some of those ways could be more efficient performance-wise. Yet, clarity and simplicity of the query were the primary considerations while deciding on the query version to share.

SQL

```sql
WITH token_request_history AS (
    SELECT MIN(event_time)                                          AS first_seen,
           properties:servicePrincipalId::string                    AS managed_identity_id,
           properties_resource_id::string                           AS target_endpoint_id
    FROM RAW.AZURE_SIGNIN
    WHERE category = 'ManagedIdentitySignInLogs'
          -- define a learning period of 4 months
          AND event_time > CURRENT_DATE - INTERVAL '120 days'
    GROUP BY managed_identity_id, target_endpoint_id
),
new_mi_and_endpoint_pairs AS (
    SELECT managed_identity_id,
           target_endpoint_id
    FROM token_request_history
    WHERE -- the pair should be new
          first_seen > CURRENT_DATE
          -- we want to avoid a new managed identity
          AND managed_identity_id IN (
              SELECT managed_identity_id
              FROM token_request_history
              WHERE first_seen < CURRENT_DATE - INTERVAL '30 days'
          )
)
SELECT MIN(event_time)                                              AS first_seen,
       MAX(event_time)                                              AS last_seen,
       category,
       properties:servicePrincipalId::string                        AS managed_identity_id,
       properties:servicePrincipalName::string                      AS managed_identity_name,
       caller_ip_address,
       properties_user_principal_name,
       properties_is_interactive,
       properties_resource_display_name::string                     AS target_endpoint_name,
       properties_resource_id::string                               AS target_endpoint_id,
       properties_risk_state,
       properties_risk_level_during_signin,
       result_type,
       COUNT(*)                                                     AS number_of_requests
FROM RAW.AZURE_SIGNIN AS azure_signin
WHERE azure_signin.category = 'ManagedIdentitySignInLogs'
      -- the hunting is running on the current day
      AND azure_signin.event_time > CURRENT_DATE
      -- filter on the new mi-endpoint pairs
      AND EXISTS (
          SELECT 1
          FROM new_mi_and_endpoint_pairs AS new_mi_and_endpoint_pairs
          WHERE new_mi_and_endpoint_pairs.managed_identity_id = azure_signin.properties:servicePrincipalId::string
                AND new_mi_and_endpoint_pairs.target_endpoint_id = azure_signin.properties_resource_id::string
      )
GROUP BY category,
         managed_identity_name,
```

```
managed_identity_id,
target_endpoint_name,
target_endpoint_id,
caller_ip_address,
properties_user_principal_name,
properties_is_interactive,
properties_risk_state,
properties_risk_level_during_signin,
result_type
```

Hunting Query 5 - Unique Token Identifier Usage From Multiple IP Addresses:

Thesis

In this threat-hunting thesis, we look for classic token theft based on the Unique Token Identifier values available in some Azure log sources. In this case, we specifically focused on the Azure activity log source that includes the Unique Token Identifier value in the (`IDENTITY_CLAIMS:uti`) to look for instances in which activities were conducted from two different IP addresses using the same access token - this, of course, can indicate a JWT access token theft. A JWT access token can be used from practically everywhere, including non-Azure resources, after its theft.

Data source(s)

- Azure Activity

Fidelity level

- **High**

Tuning & Observations

Hunting Query Notes:

- In case you find this query noisy in your environment (which is unlikely, but possible), you can try to clean out some more likely to be automation-related operations, by adding the following condition to the hunting query below: `AND NOT OPERATION_NAME ILIKE ANY ('%DEPLOYMENTSTACKS%', '%WORKFLOWS%', '%RESTOREPOINTS%', '%MICROSOFT.NETWORK%')`
    - Or any other potentially noisy actions repeatedly used by MIs in your environment.

Query

- This is a relatively simple 'Group By' hunting query that looks for cases in which the initiated activity was conducted by a service principal of type MI.
- The MI type was identified by evaluating the content of the "`xms_mirid`" field, which wouldn't exist in the case of a regular service principal and will be populated with an identifier in the case of activity conducted by a MI.
- The Group By is being conducted using the unique token identifier, identifiers of the specific MI, and its type. The results include only cases in which multiple IP addresses were found as source IPs of the activity (`NUMBER_OF_SOURCE_IPS > 1`).

SQL

```
SELECT MIN(event_time)                                                    AS min_event_time,
       MAX(event_time)                                                    AS max_event_time,
       identity_claims:uti                                                AS unique_token_id,
       ARRAY_AGG(DISTINCT caller_ip_address)                              AS source_ip_addresses,
       ARRAY_AGG(DISTINCT operation_name)                                 AS operation_names,
       ARRAY_AGG(DISTINCT identity_authorization:evidence.principalType)  AS principal_types,
       identity_claims:appid                                              AS principal_app_id,
       ARRAY_AGG(DISTINCT identity_claims:aud)                            AS token_audience,
       ARRAY_AGG(DISTINCT identity_authorization:evidence.role)           AS role_type,
       ARRAY_AGG(DISTINCT identity_authorization:evidence.roleAssignmentScope)  AS role_assignment_scope,
       ARRAY_AGG(DISTINCT resource_id)                                    AS target_resource_id,
       ARRAY_AGG(DISTINCT identity_claims:xms_az_rid)                     AS managed_identity_resource_id,
       --- available only for UAMI
       identity_claims:xms_mirid                                          AS managed_identity_name,
       --- if UAMI, it includes userAssignedIdentities. Otherwise it's SAMI
       CASE
```

```
            WHEN managed_identity_name ILIKE '%userAssignedIdentities%' THEN 'UAMI'
            ELSE 'SAMI'
        END                                                                 AS managed_identity_type,
        ARRAY_AGG(DISTINCT result_type)                                     AS result_types,
        ARRAY_SIZE(source_ip_addresses)                                     AS number_of_source_ips,
        COUNT(*)                                                            AS numbmer_of_activities
FROM RAW.AZURE_ACTIVITY
-- UAMI example
WHERE event_time BETWEEN '2025-01-01' AND '2025-03-01'
      AND identity_authorization:evidence.principalType = 'ServicePrincipal'
      AND identity_claims:xms_mirid IS NOT NULL
GROUP BY unique_token_id,
         principal_app_id,
         managed_identity_name,
         managed_identity_type
HAVING number_of_source_ips > 1
```

Hunting Query 6 - MI's mass token types requested:

Thesis

From an attacker's perspective, gaining access to an MI access token unlocks a wide range of opportunities. MIs can be exploited across various Azure services that support them, and a stolen token may grant access to different types of resources (as we covered in part 1 of this research), including:

- **Azure Resource Manager (ARM) APIs** – Potential full control over Azure resources.
- **Storage accounts** – Access to sensitive data stored in Azure Blob or File storage.
- **Key Vaults** – Ability to retrieve secrets, certificates, or encryption keys.
- **Microsoft Graph** – A vector for attacks within Entra ID and the broader Microsoft 365 ecosystem.

When a threat actor gains access to a resource with an attached MI, they will likely attempt to enumerate its assigned permissions. In many cases, this involves requesting various token types, decoding them to analyze their scope, or directly using them to conduct unauthorized activities.

In this threat hunting thesis, we look for this kind of case, identifying a single MI that requests multiple token types within a short time window. Such behavior may indicate a compromised MI, with an attacker probing its potential permissions by requesting different tokens to determine what actions they can execute.

Note: Unlike other queries that target specific services such as Virtual Machines or Key Vaults, this query is service-agnostic and applicable across the Azure platform.

Data source(s)

- "Managed identity sign-ins" category under **Azure Sign-Ins logs**.

Fidelity Level

- **Medium**

Tuning & Observations

Notes:

- If this threat-hunting query is found to be a little noisy in your environment, please conduct the required adjustments:
  - Modification of the `UNIQUE_RESOURCE_TYPES` threshold.
  - Exclude specific token types used by your organization's MIs that are less likely to be probed by a threat actor as part of the enumeration.
  - Exclusion of specific groups of MI and requested token types. For example, "Managed_Id_XYZ" always asks for the following four token types: ARM, Key Vault, Azure Arc, and Azure Monitor.

False-Positive Example:

- A MI named "azure_resource_tracker" was identified requesting four different token types in a short timeframe.
- The requested token types included Azure Resource Manager, Azure SQL, Azure Key Vault, and Azure Storage.
- Even though the token types mentioned are known to be important and can cause severe damage if compromised, we found that this MI typically requests them on a daily basis.
- After investigating, it was found that this MI is used to map the organizational Azure resources as part of an automatic activity, sending this information to a known cost optimization application.

False-Negative Example:

There are potential blind spots for this threat hunting thesis:

- We will miss cases where the threat actor requested different types of tokens slowly, not all/multiple at once.
- Cases in which the actor focused on a specific area, such as Azure Key Vaults, asked for a minimal number of tokens (less than 4).
- The requested tokens were related to services we excluded from the hunting query, like Azure Arc.

Query

In the hunting query below, we used:

- `TIME_SLICE` to look for events in a time window of 15 minutes.
- We looked specifically for sign-in events of the MI category, equivalent to MI token requests.
- `GROUP_BY` to group the results by multiple columns, such as Service principal name, Service principal ID, 15-minute time window, etc.
- While looking for cases of at least four distinct token types requested, excluding token types less likely to be related to this kind of probing/enumeration of attached permissions.

SQL

```sql
SELECT DATE_TRUNC('DAY', event_time)                                    AS day_of_events,
       TIME_SLICE(CAST(event_time AS TIMESTAMP_NTZ), 15, 'MINUTE')      AS fifteen_min_interval,
       MIN(event_time)                                                  AS min_event_time,
       MAX(event_time)                                                  AS max_event_time,
       properties:servicePrincipalName                                  AS service_principal_name,
       properties:servicePrincipalId                                    AS service_principal_id,
       ARRAY_AGG(DISTINCT properties_resource_display_name)             AS resource_display_name,
       ARRAY_SIZE(resource_display_name)                                AS unique_resource_types,
       ARRAY_AGG(DISTINCT properties:uniqueTokenIdentifier)             AS uti,
       COUNT(*)                                                         AS number_of_tokens
FROM RAW.AZURE_SIGNIN
-- Adjust threshold based on needs
WHERE event_time BETWEEN '2024-10-01' AND '2025-01-25'
      -- Filtering out EventHubs sign-in logs
      AND properties_resource_display_name NOT IN ('Microsoft.EventHubs')
      -- Looking specifically for Managed Identity Sign-in logs
      AND category = 'ManagedIdentitySignInLogs'
      -- Excluding specific Resource types
      AND NOT properties_resource_display_name ILIKE ANY (
              '% Arc %',
              '%Azure Monitor%',
              '%ServiceBus%',
              '%GuestNotificationService%',
```

```
            '%hybridcompute%'
        )
GROUP BY service_principal_name,
         service_principal_id,
         day_of_events,
         fifteen_min_interval
-- Adjust threshold in case needed, based on your organizational routine MIs usage
HAVING unique_resource_types > 3
```

Hunting Query 7 - Managed Identity Activities from Non-Azure IP Addresses:

Thesis

This thesis examines potential Azure activities conducted using a stolen JWT access token. This time, we focus on cases where the threat actor potentially stole and used the token from a non-organizational Azure resource.

While MI activity typically originates from Azure-owned infrastructure, anomalous activity from non-Azure IP addresses may indicate credential compromise or misuse. Defenders can detect suspicious MI activity by filtering out known Azure and organizational IPs, revealing potential attacker-controlled infrastructure and unauthorized access attempts.

Data source(s)

- Azure Activity Logs

Fidelity level

- **Medium**

Tuning & Observations

- The attacker operates from one of Azure IP Addresses (for example, from a virtual machine that is part of the threat actor's environment)

- It is important to note that the IP ranges used in the query below are relatively broad, and are not specifically limited to the exact Azure ranges found in the **Microsoft publication** . You can use the exact IP addresses/ranges if required.
- The hunting query below does not filter organizational IP ranges; however, it can be conducted in case of unexpected noise.

Query

The following hunting query follows very simple logic. We only filter for MI activities based on the `xms_mirid` field in the activity logs, looking for any activity originating from non-Azure IP addresses. The filtering excludes any IP address not part of the Azure IP range.

SQL

```
SELECT event_time                                          AS event_time,
       identity:claims:xms_mirid                           AS managed_identity,
       identity_authorization:evidence:principalId         AS principal_id,
       caller_ip_address                                   AS source_ip_address,
       operation_name                                      AS event_name,
       identity_claims:appid                               AS app_id,
       category                                            AS category,
       level                                               AS level,
       resource_id                                         AS resource_id,
       result_type                                         AS result_type
FROM RAW.AZURE_ACTIVITY
WHERE managed_identity IS NOT NULL
      AND event_time > CURRENT_TIMESTAMP - INTERVAL '60 days'   -- Adjust to your preference
      -- Remove Azure IP Ranges
      AND NOT (caller_ip_address ILIKE ANY (
                  '102.%', '103.%', '104.%', '108.%', '111.%', '128.%', '13.%', '130.%', '131.%',
                  '132.%', '134.%', '135.%', '137.%', '138.%', '147.%', '150.%', '151.%', '157.%',
                  '158.%', '167.%', '168.%', '172.%', '191.%', '193.%', '198.%', '199.%', '20.%',
                  '202.%', '204.%', '207.%', '209.%', '213.%', '216.%', '23.%', '4.%', '40.%',
                  '48.%', '50.%', '51.%', '52.%', '57.%', '64.%', '65.%', '68.%', '69.%', '70.%',
                  '72.%', '74.%', '85.%', '9.%', '94.%', '0.%', '98.%'
               ))
```

Hunting Query 8 - Unusual Action Types by a Managed Identity:

Thesis

This threat-hunting thesis looks for cases in which the threat actor gained unauthorized access to a MI JWT access token, but used it from within the targeted Azure infrastructure. In this case, some of the IP-related hunting queries mentioned in this research won't be sufficient.

To detect, we create a baseline of "normal" action types conducted by the organizational MIs, comparing the recent activities to this baseline.

This query is helpful because it is highly probable that a threat actor will deviate from the typical actions performed by the MI.

Data source(s)

- Azure Activity Logs

Fidelity level

- **Medium**

Tuning & Observations

False-positive use case:

- This hunting query can potentially produce false positives related to MIs associated with internal scanning, asset management, and identity management systems that routinely conduct different actions against many organizational resources. If this is the case for you, consider focusing on other MIs using this query.

Query

- In this hunting query, we used a CTE of Standard Operations, which collects the normally conducted operation names from the last few weeks in an array named "`OPERATION_NAMES`".
- Later, we use this list of operations to compare recent activities conducted by the same MIs and identify operation names that don't align with the baseline.
- We also track cases of newly seen MIs that weren't part of the baseline to ensure that they are not only not missed but properly categorized differently.

SQL

```
WITH standard_operations AS (
    SELECT identity_authorization:evidence.principalId         AS service_principal_id,
           identity_claims:appid                               AS principal_app_id,
           ARRAY_AGG(DISTINCT operation_name)                  AS operation_names
    FROM RAW.AZURE_ACTIVITY
    WHERE event_time BETWEEN '2025-01-01 17:00:00' AND '2025-02-01 16:59:00'
          AND identity_claims:xms_mirid IS NOT NULL
    GROUP BY 1, 2
)
SELECT a.event_time                                                    AS event_time,
       a.identity_claims:uti                                           AS unique_token_id,
       a.caller_ip_address                                             AS source_ip_addresses,
       a.operation_name                                                AS anomalous_operation_names,
       a.identity_authorization:evidence.principalType                 AS principal_types,
       a.identity_claims:appid                                         AS principal_app_id,
       a.identity_authorization:evidence.principalId                   AS service_principal_id,
       a.identity_claims:aud                                           AS token_audience,
       a.identity_authorization:evidence.role                          AS role_type,
       a.identity_authorization:evidence.roleAssignmentScope           AS role_assignment_scope,
       a.resource_id                                                   AS target_resource_id,
       a.identity_claims:xms_az_rid                                    AS token_requested_from_resource,
       a.identity_claims:xms_mirid                                     AS managed_identity_name,
       a.operation_name                                                AS anomalous_operation_name,
       CASE
           WHEN a.identity_claims:xms_mirid ILIKE '%userAssignedIdentities%' THEN 'UAMI'
           ELSE 'SAMI'
       END                                                             AS managed_identity_type,
       a.result_type                                                   AS result_type,
       a.caller_ip_address                                             AS number_of_source_ips,
       -- Flag for deviations: If operation is NOT in the baseline, mark as Anomaly
       CASE
           WHEN s.principal_app_id IS NULL THEN 'Newly Seen MI (Not In baseline)'
           WHEN s.principal_app_id IS NOT NULL
                AND ARRAYS_OVERLAP(ARRAY_CONSTRUCT(a.operation_name), s.operation_names) = FALSE
           THEN 'Anomalous'
```

```
            ELSE 'Normal'
        END                                                                    AS anomaly_flag
FROM RAW.AZURE_ACTIVITY AS a
LEFT JOIN standard_operations AS s
    ON a.identity_authorization:evidence.principalId = s.service_principal_id
WHERE a.event_time BETWEEN '2025-02-02 17:00:00' AND '2025-04-01 19:30:00'
      AND a.identity_authorization:evidence.principalType = 'ServicePrincipal'
      AND a.identity_claims:xms_mirid IS NOT NULL
      AND anomaly_flag = 'Anomalous'
```

Hunting Query 9 - Attached UAMI used from a New Azure Resource:

Thesis

Threat actors who have gained unauthorized access to a resource group with a user-assigned MI can exploit it by attaching this UAMI to an Azure resource for privilege escalation.

For example, a compromised Entra ID user account that has the Contributor RBAC role targeting the “AXON-MGMT-RG” attached to it. The “AXON–MGMT-RG” has a different type of resource in it, among them a UAMI named “MGMT-UAMI”.

This UAMI has different permissions attached to it, including the “Contributor” role targeting “AXON-PROD-RG”, “AXON-DEV-RG”, etc.

The threat actor, who gained access to a user with high privileges on the “AXON-MGMT-RG”, doesn’t have much to do now, besides attaching this UAMI to any resource, e.g. a VM that exists in the “AXON-MGMT-RG”, accessing this VM and escalating his privileges, requesting an ARM token “on behalf” of this UAMI.

With this hunting query, we aim to identify the above scenario. In addition to looking at the UAMI's actual attachment to a new resource, we look specifically for UAMI access tokens created on unusual resources— resources from which they don’t originally operate. We use the `xms_az_rid` and `xms_mirid` fields in the Azure Activity logs to do that.

- `xms_mirid` represents the MI resource ID, which indicates that a MI was involved in this activity and provides the exact MI resource identifier.
- `xms_az_rid` - This field indicates the actual Azure resource from which the token request originated. This is super useful for UAMI detection/investigation because

in the case of UAMI (in contrast to SAMI), the `xms_mirid` doesn't provide this information, only the actual name of the MI.

Data source(s)

- Azure Activity Logs

Fidelity level

- **Medium**

Tuning & Observations

- This query is considered Medium-fidelity; however, it should be easily adjustable to remove most noisy results.
    - An example of a false positive is a case in which dynamic MI resources are used for scanning/mapping the network, which are commonly used by known cloud products. Excluding those service principals/MIs that initiated the known activity types can be a relatively easy and useful step for a threat-hunting/detection implementation.

Query

This hunting query is composed of multiple parts, detailed as follows:

- CTE that includes the standard operations conducted by each MI over the months before the hunting timeframe. This is done using a Group-by logic, grouping by MI identifiers while aggregating the operation names and "`xms_az_rid`" fields to build lists of the origin resources and the activities each MI conducts.
- The main part of the query joins the Azure activity table with the CTE, looking for activities originated by UAMI, where the ANOMALY_FLAG column is set to "Anomalous".
    - This column is set to "Anomalous" only for cases in which the results of ARRAY_OVERLAP results are "False", which means that any of the array items of "`xms_az_rid`" field of the main logs lookup, weren't included in the array items of "`xms_az_rid`" field in the normal resources/activities CTE.

SQL

```sql
WITH standard_operations AS (
    SELECT MIN(event_time)                                          AS min_event_time,
           MAX(event_time)                                          AS max_event_time,
           identity_authorization:evidence.principalId              AS service_principal_id,
           identity_claims:appid                                    AS principal_app_id,
           ARRAY_AGG(DISTINCT identity_claims:xms_az_rid)           AS token_requested_from_resource,
           ARRAY_AGG(DISTINCT operation_name)                       AS operation_names
    FROM RAW.AZURE_ACTIVITY
    WHERE event_time BETWEEN '2024-11-01' AND '2025-01-01' -- learning period of 2 months
          AND identity_claims:xms_mirid IS NOT NULL
          AND identity_claims:xms_mirid ILIKE '%userAssignedIdentities%'
          AND identity_claims:xms_az_rid IS NOT NULL
    GROUP BY 3, 4
)
SELECT MIN(event_time)                                              AS min_event_time,
       MAX(event_time)                                              AS max_event_time,
       a.identity_claims:uti                                        AS unique_token_id,
       a.caller_ip_address                                          AS source_ip_address,
       a.operation_name                                             AS anomalous_operation_names,
       a.identity_authorization:evidence.principalType              AS principal_types,
       a.identity_claims:appid                                      AS principal_app_id,
       a.identity_authorization:evidence.principalId                AS service_principal_id,
       a.identity_claims:aud                                        AS token_audience,
       a.identity_authorization:evidence.role                       AS role_type,
       a.identity_authorization:evidence.roleAssignmentScope        AS role_assignment_scope,
       a.resource_id                                                AS target_resource_id,
       COALESCE(a.identity_claims:xms_az_rid, 'New Unidentified Azure Resource')
                                                                    AS token_requested_from_resource,
       a.identity_claims:xms_mirid                                  AS managed_identity_name,
       a.operation_name                                             AS anomalous_operation_name,
       CASE
           WHEN a.identity_claims:xms_mirid ILIKE '%userAssignedIdentities%' THEN 'UAMI'
           ELSE 'SAMI'
       END                                                          AS managed_identity_type,
       ARRAY_AGG(DISTINCT a.result_type)                            AS result_type,
       -- Flag for deviations: If operation is NOT in the baseline, mark as Anomaly
       CASE
           WHEN s.principal_app_id IS NULL THEN 'Newly Seen MI (Not In baseline)'
           WHEN s.principal_app_id IS NOT NULL
                AND ARRAYS_OVERLAP(
                    ARRAY_CONSTRUCT(a.identity_claims:xms_az_rid),
                    s.token_requested_from_resource
                ) = FALSE
           THEN 'Anomalous'
           ELSE 'Normal'
       END                                                          AS anomaly_flag
FROM RAW.AZURE_ACTIVITY AS a
LEFT JOIN standard_operations AS s
    ON a.identity_authorization:evidence.principalId = s.service_principal_id
WHERE a.event_time BETWEEN '2025-01-01' AND '2025-01-08'
```

```
      AND a.identity_authorization:evidence.principalType = 'ServicePrincipal'
      AND a.identity_claims:xms_mirid IS NOT NULL
      AND a.identity_claims:xms_mirid ILIKE '%userAssignedIdentities%'
      AND anomaly_flag = 'Anomalous'
GROUP BY unique_token_id,
         source_ip_address,
         anomalous_operation_names,
         a.identity_claims:appid,
         a.identity_authorization:evidence.principalId,
         a.identity_claims:xms_az_rid,
         token_audience,
         role_type,
         role_assignment_scope,
         target_resource_id,
         token_requested_from_resource,
         managed_identity_name,
         anomalous_operation_name,
         managed_identity_type,
         anomaly_flag,
         principal_types
```

Hunting Query 10 - MI performs activity on Entra ID:

Thesis

MI should typically access Azure resources and shouldn't normally make changes in Entra ID (for example, assign roles, add credentials to app, etc.). Broad read activity on Entra ID, e.g., listing all users in the directory initiated by SAMI, is also unusual, and we might like to detect it as well. This is a specific use case of the scenario "MI accesses unusual resources".

Data source(s)

- Azure Audit
- Managed Identities Inventory

Fidelity level

- **Medium**

Tuning & Observations

- **Hunting Query Notes:**
  - This hunting query can be useful for organizations where MIs are not commonly used for Entra ID-related activities, like application management and Entra ID privileges management.
    - For organizations in which these kinds of actions are actually normal, this query can be used as a template to build additional logic, such as filtering out specific operation names that are normally conducted by organizational MIs.
- **False Positive use cases:**
  - Depending on the organization, it's possible to see MIs that operate on Entra ID. For example, a UAMI associated with an application that automatically configures permissions on Entra ID.
- **False Negative use cases:**
  - If the compromised MI was originally intended to operate on Entra ID, there is a risk that the SOC analysts would consider that activity benign even when the MI is compromised.

Query

This is a straightforward query that looks for any operation logged in the Azure Audit logs, and was conducted by a MI.

- We use the `managed_identities_inventory` table we've created earlier in this document to identify MIs.

```sql
SELECT azure_audit.event_time,
       azure_audit.operation_name,
       azure_audit.properties_category,
       azure_audit.caller_ip_address,
       azure_audit.properties_activity_display_name,
       azure_audit.properties_additional_details[0]:value::string          AS user_agent,
       azure_audit.properties_initiated_by:app:displayName::string         AS initiating_app_display_name,
       azure_audit.properties_initiated_by:app:servicePrincipalId          AS initiating_service_principal_id,
       managed_identities_inventory.managed_identity_type                  AS managed_identity_type,
       azure_audit.properties_result,
       azure_audit.properties_target_resources
FROM RAW.AZURE_AUDIT AS azure_audit
JOIN INVESTIGATION.MANAGED_IDENTITIES_INVENTORY AS managed_identities_inventory
    ON initiating_service_principal_id = managed_identities_inventory.managed_identity_id
WHERE LOWER(properties_result) = 'success'
      AND azure_audit.event_time > CURRENT_TIMESTAMP - INTERVAL '60 days'
```

Hunting Query 11 - SAMI activity from abnormal IP:

Thesis

Since MIs rely on access tokens rather than static credentials, a compromised token could allow an adversary to authenticate and access cloud resources under the guise of a legitimate identity. An attacker could operate from their Azure instance, making the source IP appear legitimate within Microsoft's infrastructure. This significantly complicates traditional network-based detection mechanisms, as security teams may rely on Azure IP ranges as inherently trusted.

This thesis investigates suspicious MI activity by detecting potential token compromise and unauthorized usage within Azure environments. The research focuses on identifying cases where an MI token is used from an unexpected IP address, specifically cases where the token was used from multiple IP addresses in a short timeframe, using the LAG Snowflake function.

It should be useful for the detection of token replay, lateral movement within short timeframes, etc.

Data source(s)

- Azure Activity logs

Fidelity level

- **Low**

Tuning & Observations

- It is possible to increase the `time_diff_minutes < 10` to a larger time difference, or even avoid this restriction, to reduce the chance of potentially missing relevant findings. Tuning it based on your environment is recommended.
- In addition, in case of some automation activities that might lead to false-positives, you can consider adding Resource_ID exclusions, like for example: `AND RESOURCE_ID NOT ILIKE '%ORCA%'`

False Negative use cases:

- The attacker operates from a UAMI MI
- The stolen token was used long after, which doesn't align with the time difference restriction we use in the main part of the query

Query

- In this query, we used a CTE that tracks organizational SAMIs' activities using the LAG function to collect additional information about previous activities conducted by the same SAMI with the same characteristics (for example, the same IP, the same unique token identifier, etc.).
- Using the extra information from the CTE, we looked for specific cases of potential token theft. We looked for results in which the second IP (identified by the LAG() function) differed from the first IP. We also limited the time difference between the first IP identification and the second IP identification to 10 minutes.

SQL

```
WITH sami_activity_with_lag AS (
    SELECT event_time,
           identity:claims."http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"
                                                                                    AS name_identifier,
           caller_ip_address                                                        AS current_ip,
           resource_id                                                              AS current_resource_id,
           identity_claims:uti                                                      AS current_token_id,
           -- Previous event details for the same SAMI
           LAG(caller_ip_address) OVER (
               PARTITION BY identity:claims."http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"
               ORDER BY event_time
           )                                                                        AS prev_ip,
           LAG(event_time) OVER (
               PARTITION BY identity:claims."http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"
               ORDER BY event_time
           )                                                                          AS prev_event_time,
           LAG(resource_id) OVER (
               PARTITION BY identity:claims."http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"
               ORDER BY event_time
           )                                                                          AS prev_resource_id,
           LAG(identity_claims:uti) OVER (
               PARTITION BY identity:claims."http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"
               ORDER BY event_time
           )                                                                          AS prev_token_id
    FROM RAW.AZURE_ACTIVITY
    WHERE identity:claims."http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" IN (
              SELECT DISTINCT managed_identity_id
              FROM INVESTIGATION.MANAGED_IDENTITIES_INVENTORY AS managed_identities_inventory
```

```
            WHERE managed_identities_inventory.managed_identity_type = 'SAMI' -- Ensure only SAMI
        )
        AND resource_id NOT ILIKE ANY (
            '%POLICYDEPLOYMENT%',
            '%SETBYPOLICY%',
            '%DIAGNOSTICSETTINGS%'
        )
        AND event_time > CURRENT_TIMESTAMP - INTERVAL '60 days'
)
SELECT name_identifier,
       prev_event_time                                                    AS first_seen_time,
       event_time                                                         AS second_seen_time,
       TIMESTAMPDIFF(MINUTE, prev_event_time, event_time)                 AS time_diff_minutes,
       -- IP Address Change Detection
       prev_ip                                                            AS first_ip,
       current_ip                                                         AS second_ip,
       CASE WHEN prev_ip <> current_ip THEN TRUE
            ELSE FALSE
       END                                                                AS ip_changed,
       -- Resource Tracking
       prev_resource_id                                                   AS first_resource_id,
       current_resource_id                                                AS second_resource_id,
       prev_token_id                                                      AS first_token_id,
       current_token_id                                                   AS second_token_id,
       -- Token Reuse Check
       CASE WHEN prev_token_id = current_token_id THEN TRUE
            ELSE FALSE
       END                                                                AS token_reused
FROM sami_activity_with_lag
WHERE ip_changed
      AND time_diff_minutes < 10 -- Focus on rapid IP changes
ORDER BY name_identifier, prev_event_time
```

Hunting Query 12 - MI accesses unusual resources:

Thesis

This hunting logic is based on behavioral baselining of Azure MIs to detect potential misuse or anomalous activity. It assumes that during a stable observation window (e.g., 30–60 days in the past), MIs typically interact with a limited set of operations and resource types within their intended scope. The query establishes a behavioral profile by learning this normal pattern, based on combinations of identity, resource group, and operation. Then, in a more recent timeframe (e.g., the last 30 days), it flags any activity that deviates from this learned behavior. These deviations might include a MI accessing a

different resource group or performing unfamiliar operations, which could indicate abuse, lateral movement, or misconfigured permissions.

Data source

- Azure Activity Logs

Fidelity level

- **Low**

Tuning & Observations

- In case of some automation activities that might lead to false-positives, you can consider adding Resource_ID exclusions, like for example:
  `AND mird not ilike '%OrcaScannerIdentity%'`
- If there is insufficient data (e.g., less than 2 months), the query results may be less reliable due to the lack of a strong baseline. In such cases, it is recommended to adjust the time intervals to ensure enough data is available to establish a meaningful baseline.

Query

- In this query, we created a CTE that includes the baseline of each MI's activities over the last X days, including the operations they conducted against the different resource groups.
- Then, for the main part of the query, we look for recent activities conducted by the same MI while filtering out any combination that was already seen as part of the CTE learning.
- It is worth mentioning that as part of this query, we use this regex `regexp_substr(act.resource_id, '^(.*)/[^/]+$', 1, 1, 'e')` to conduct the comparison against the Resource group, instead of specifically comparing the resource to its baseline.

SQL

```sql
WITH cte_learning AS (
    SELECT identity_claims:appid                                                    AS app_id,
           identity_claims:xms_mirid                                                AS mird,
           operation_name,
           REGEXP_SUBSTR(resource_id, '^(.*)/[^/]+$', 1, 1, 'e')                     AS base_resource_id  -- in
order to get resource groups
    FROM RAW.AZURE_ACTIVITY
    WHERE event_time BETWEEN CURRENT_TIMESTAMP - INTERVAL '60 days'
                         AND CURRENT_TIMESTAMP - INTERVAL '30 days'  -- set time to your liking
          AND identity_claims:xms_mirid IS NOT NULL
    GROUP BY app_id,
             mird,
             base_resource_id,
             operation_name
)
SELECT identity_claims:appid                                                        AS app_id,
       identity_claims:aud                                                          AS aud,
       operation_name,
       ARRAY_AGG(DISTINCT event_time)                                               AS time_chart,
       identity_claims:xms_mirid                                                    AS mird,
       ARRAY_AGG(DISTINCT REGEXP_SUBSTR(act.resource_id, '^(.*)/[^/]+$', 1, 1, 'e'))   AS rsc_id
FROM RAW.AZURE_ACTIVITY AS act
WHERE event_time > CURRENT_TIMESTAMP - INTERVAL '30 days'  -- set time to your liking
      AND identity_claims:xms_mirid IS NOT NULL
      AND NOT EXISTS (
          SELECT 1
          FROM cte_learning AS learn
          WHERE learn.app_id = act.identity_claims:appid
                AND learn.mird = act.identity_claims:xms_mirid
                AND learn.base_resource_id = REGEXP_SUBSTR(act.resource_id, '^(.*)/[^/]+$', 1, 1, 'e')
                AND learn.operation_name = act.operation_name
      )
GROUP BY app_id,
         aud,
         operation_name,
         mird
```

Incident Investigation & Response

Now that we've laid out a detailed threat-hunting approach built around multiple detection theses, guiding what comes next is equally important. This section outlines key investigation techniques for potentially compromised MIs, offering correlation strategies, and recommendations for using complementary log sources. The goal is to equip defenders not only with how to detect suspicious activity but also to trace, validate, and respond to it effectively.

Compromised Managed Identities - Investigation Guidelines

There are various "entry points" for Azure MIs investigations. You may be aware of a specific compromised token, observe suspicious usage of a MI by an unauthorized entity, or detect unusual behavior that suggests malicious activity occurring under the context of an MI.

The following guidelines outline recommended investigative actions to evaluate potentially compromised MIs. Please note that the order of the steps can differ depending on the investigation trigger.

1. Evaluate the Characteristics and Permissions of the MI

- Is the identity a SAMI or a UAMI?
- Assess the types of permissions granted to the MI to understand its potential blast radius.
- Take into account:
  - **Azure RBAC roles**
  - **Entra ID (Azure AD) roles**
  - **API permissions** (e.g., Graph API scopes like `Mail.Read`)
- Consider the scope of these permissions, for example, does the MI have Contributor access at the **subscription level**, or is it limited to a specific **resource group**?

2. Analyze Token Requests in Azure Sign-In Logs

- Use Azure Sign-In Logs to examine the **resources** for which access tokens were requested.
- Determine if these token requests align with the typical behavior of the MI.
- If tokens were requested for **unusual services**, this may indicate suspicious activity.
- In confirmed compromise cases, these token request logs are valuable for **incident scoping**.

3. Identify Activities Conducted with the Tokens

- Investigate what actions were performed using the tokens.
- Correlate token requests to activities across log sources depending on token type:
  - **ARM token** → Azure Activity Logs.
  - **Graph tokens** → Azure Audit Logs, Microsoft 365 Audit Logs, Microsoft Graph Activity Logs.
  - **Service-specific tokens** → Key Vault, Storage Account logs, etc.
- If available, use the **unique token identifier** (`properties:uniqueTokenIdentifier`) to correlate token requests with activity logs, allowing for more precise investigation.

4. Identify the Potential Attack Path

Focus on **how** the MI may have been compromised:

- Which **users** (Entra ID or local resource users) authenticated to the resource hosting to which the MI was attached?
- Are there access attempts from **unusual IP addresses**?
- Did any **unusual operations** occur on the resource to which the MI is attached before the suspicious MI activity?

5. Investigate Log Sources of Interest

As you examine logs, consider the following investigative questions:

- What IP addresses were involved?
  - Are they internal?
  - Are they from known Azure service IPs?
- Do the activities align with expected MI behavior?

    - **Example**: If an MI usually deploys VMs but was used to read blob storage or reset VM passwords, this may indicate misuse.
- What is the **impact** of the observed activity?
    - **Example**: If the MI was used to reset VM credentials using `enablevmaccess`, this suggests **lateral movement**.
- Was the MI attached to other resources?
- Is there evidence of **further propagation** within Azure or hybrid environments?
- Keep in mind that there are additional complementary log sources (not the "main" ones) that can be super useful for different scenarios. (more details in the "Complementary Log Sources" section below).

6. Leverage Previous Hunting Queries for Scoping

Queries presented in the threat hunting section are also highly useful for investigation.

For example, if the investigation begins with the understanding that an attacker compromised multiple Entra ID accounts and one had access to a resource group with a high-privilege UAMI, use anomaly-based hunting queries to check whether that MI was misused for **privilege escalation** or **lateral movement**. Filter for the specific UAMI and compare its recent behavior to its historical baseline.

7. Expand the Investigation Scope

Once initial findings are confirmed:

- Identify **indicators of compromise (IOCs)**.
- Investigate any suspicious activity on:
    - **Entra ID objects** (user creation, role changes, etc.)
    - **Service Principals & Enterprise Apps**
    - **Microsoft 365 infrastructure**
- Use established methodologies to trace the attacker's path across systems and services.

If you're unfamiliar with Azure log sources or need additional guidance, refer to our **Human-Friendly Guide: Incident Response & Threat Hunting in Microsoft Azure.**

Up Next: To support your investigation further, the following section introduces **correlation techniques** that tie together different data sources for a more holistic view.

Cross Microsoft Data Sources Correlations

As mentioned above, one of the most important aspects of investigating a compromised MI (and not only for this case) is properly correlating the different Microsoft log sources to keep the investigation efforts focused and efficient.

The four main log sources—Azure Activity logs, Azure Audit logs, Azure Sign-in logs, Graph activity logs, and M365 Audit logs—can all be relevant to MI investigations, depending on the permissions and privileges granted to the MI of interest.

Some of the log sources mentioned above can be easily correlated. The correlation can be conducted based on the **Unique Token Identifier** value available in some of the log sources.

The Azure Sign-in logs provide details about the relevant resource type for which the sign-in was conducted. For example, a sign-in conducted to an Azure Key Vault resource type can be correlated to a specific token created as part of this sign-in event. This token will provide access to this specific resource type, which is included in the "aud" field of the JWT access token.

This token has a unique identifier value that represents it, and can be used to correlate the specific token to the actions conducted using it over some of the different data sources.

In other cases, where the unique token identifier doesn't exist in the log source, we can use "classic" correlation methods, looking for actions that were conducted in approximate time to the token creation, source IP-based correlations, etc.

The table below summarizes the Unique Token Identifier and the availability of accurate source IPs in the different log sources, to facilitate this kind of correlation:

Log Source	Unique Token Identifier	IP Address	Comments
Azure Sign-in	PROPERTIES:uniqueTokenIdentifier	No	No source IP for MI Sign-ins
Azure Audit	N/A	Partial	IP address may reflect Azure service IPs rather than the original client IP
Azure Activity	IDENTITY_CLAIMS:uti	Yes	
Graph Activity	PROPERTIES:signInActivityId	Yes	
M365 Audit	RECORD_SPECIFIC_DETAILS:app_access_context.unique_token_id	Partial	IP addresses tend to reflect Azure service IPs rather than the original client IP

Here is an example of an investigation query that can be used to correlate Sign-in (Token request) to the relevant Azure Activity entries related to it, for a specific MI:

```sql
SELECT
    -- Sign-in Information
    azure_signin.event_time                                          AS signin_event_time,
    azure_signin.properties:servicePrincipalName                     AS service_principal_name,
    azure_signin.properties:servicePrincipalId                       AS service_principal_id,
    azure_signin.properties_resource_display_name                    AS resource_display_name,
    azure_signin.properties:uniqueTokenIdentifier                    AS uti,
    azure_signin.tenant_id                                           AS tenant_id,
    -- Azure Activities Details
    azure_activity.identity_claims:uti                               AS activity_uti,
    azure_activity.identity_claims:appid                             AS app_id,
    azure_activity.identity_claims:aud                               AS aud,
    azure_activity.identity_claims:groups                            AS groups,
    azure_activity.identity_authorization:evidence.principalId       AS principal_id,
    azure_activity.identity_authorization:evidence.role              AS role,
    azure_activity.caller_ip_address                                 AS activity_caller_ip_address,
    azure_activity.operation_name                                    AS operation_name,
    azure_activity.resource_id                                       AS resource_id,
    azure_activity.result_signature                                  AS result_signature,
```

```
       azure_activity.result_type                                                AS result_type
FROM RAW.AZURE_SIGNIN AS azure_signin
JOIN RAW.AZURE_ACTIVITY AS azure_activity
    ON azure_signin.properties:uniqueTokenIdentifier = azure_activity.identity_claims:uti
       -- Adjust timeframe based on incident's time
       AND azure_signin.event_time BETWEEN '2024-12-08 16:00:00' AND '2024-12-08 22:10:00'
       -- Make sure to add some extra time in the activity event time, to detect related activities.
       AND azure_activity.event_time BETWEEN '2024-12-08 16:00:00' AND '2024-12-10 22:10:00'
       AND azure_signin.category = 'ManagedIdentitySignInLogs'
       AND service_principal_id = '<INSERT_SERVICE_PRINCIPAL_ID_OF_MI>'
       -- In case of specific token of interest, insert the Unique Token Identifier value below
       -- AND uti = '<INSERT_UNIQUE_TOKEN_IDENTIFIER_OF_INTEREST>'
```

Complementary Log Sources

Additional log sources, in addition to the main log sources mentioned in the table above, should be considered when investigating different cases of compromised MIs.

Some log sources will be irrelevant for some investigations, while others can be crucial for other incidents. Those log sources can be referred to as complementary service-specific log sources.

Here are a few practical examples:

Azure Key Vault logs

For cases where the affected MI had privileges to sensitive resources like Key Vaults, checking which secrets the MI accessed can be crucial for scoping the investigation, analyzing the potential damage, conducting the necessary eradication and containment steps, etc.

The Azure Key Vault logs have two important fields that can be useful for this kind of case:

1. `Identity.claim.xms_mirid` - represents the MI resource ID (for SAMIs)
2. `identity.claim.xms_az_rid` - Azure resource from which the token request was requested (for UAMIs)

Here is a simple investigation query that can be used to look for the Key Vault actions that were conducted by a specific MI:

```sql
SELECT event_time,
       caller_ip_address,
       raw:identity.claim.appid                  AS application_id,
       raw:identity.claim.oid                    AS application_object_id,
       raw:identity.claim.xms_mirid              AS managed_identity_id,
       raw:identity.claim.xms_az_rid             AS managed_identity_token_source,
       raw:identity.claim.xms_az_nwperimid       AS xms_az_nwperimid,
       operation_name,
       operation_version,
       properties:clientInfo                     AS user_agent,
       properties:httpStatusCode                 AS status_code,
       resource_id                               AS key_vault_name,
       properties:isAddressAuthorized            AS is_address_authorized,
       properties:isRbacAuthorized               AS is_rbac_authorized,
       raw:resultType                            AS result_type
FROM RAW.AZURE_KEY_VAULT_LOGS
WHERE event_time BETWEEN '2025-04-05' AND '2025-04-06'
      -- here we filter on specific common operations related to Key Vault,
      -- but there are other operations that could be relevant as well
      AND operation_name IN ('SecretGet', 'SecretList')
      AND application_object_id = '<INSERT_MI_APPLICATION_OBJECT_ID>'
ORDER BY event_time ASC
```

Note: In the query above, we searched for activities related to a specific MI using the application object ID. However, you can also look for it using other attributes, including the application ID and both `xms_mirid` and `xms_az_rid` mentioned above.

Azure Storage logs

A storage account is, of course, another type of resource that a compromised MI can access. For this resource type, looking for activities conducted by a specific MI is also possible. To demonstrate this, in contrast to the Snowflake SQL queries we used throughout this entire research doc, we will demonstrate the usage of KQL and log analytics (this is for you - KQL fans):

```
Shell
StorageBlobLogs
| where OperationName == "GetBlob"
| where TimeGenerated between (datetime(2025-04-06T15:50:00Z) ..
datetime(2025-04-06T17:40:00Z))
| where RequesterObjectId == "<INSERT_OBJECT_ID_OF_MI_SERVICE_PRINCIPAL>"
| project TimeGenerated, Type, Category, AccountName, OperationName, Uri, CallerIpAddress,
AuthenticationHash, RequesterObjectId, RequesterAppId, RequesterTenantId,
RequesterAudience, UserAgentHeader, ObjectKey
```

In the query above, we look specifically at the “GetBlob” operation name for the example; however, any operation conducted by the MI of interest can be relevant.

Note: interestingly, as part of our simulations, the access attempts originated from other Azure resources using the MI (for example, a VM to which the MI was attached), were logged with a local IP address in the storage account logs. While similar GetBlob activities by the MI identity from non-Azure resources were logged with an external IP. It might be an interesting logic for detection/hunting as well. Still, it requires further validation, and potentially other services or different types of networking configurations in different Azure subscriptions can lead to false positives.

Some services have dedicated logs, or what we treat as potential forensics artifacts, that can be useful for MI-related investigations. Below, you can find three examples of this kind of service. Keep in mind that in the case of these services (and others), the investigation can be focused on what happened as a result of the compromised MI usage and the services that were compromised to gain access to the MI of interest.
For example, a compromised MI could have led to unauthorized access to an Azure Function App as part of a lateral movement. However, it can also be the other way around, in the case that a threat actor gained unauthorized access to a function app with an attached MI, stole a JWT access token from it, and used the MI to continue with the attack.

Here are a few examples of the service-specific log sources related to this kind of service:

Azure Function Apps

Several Azure activity log entry types can be found in cases of updated function apps, including:

- Update Website
- Update Web Apps Functions
- Write <script_name> (for example Write Run.ps1)

All of the above may provide indications for interaction with the relevant function app, which, in the context of MI investigations, can be related to a prior edit of the app to, for example, extract JWT access tokens of a MI attached to the function app. (You can read more about it in this **blog** by SpecterOps.)

Besides those Azure activity logs, that are part of the "main" log sources we already discussed, there are additional data sources that can be used to gather additional forensics artifacts:

1. **Function Apps Invocation Logs**—This log source provides up to 20 of the most recent function invocation traces. It can be useful to get information about execution times and the reasons for execution (for example, programmatically called via the host APIs).
2. **Application Insights** - provides extended monitoring capabilities to the created function app. It can be very useful to get additional information about the function app usage, also in visualized graphs that can assist in identifying anomalies, but also to get verbose request details, including the targeted URL, request type, etc. using Transaction search.

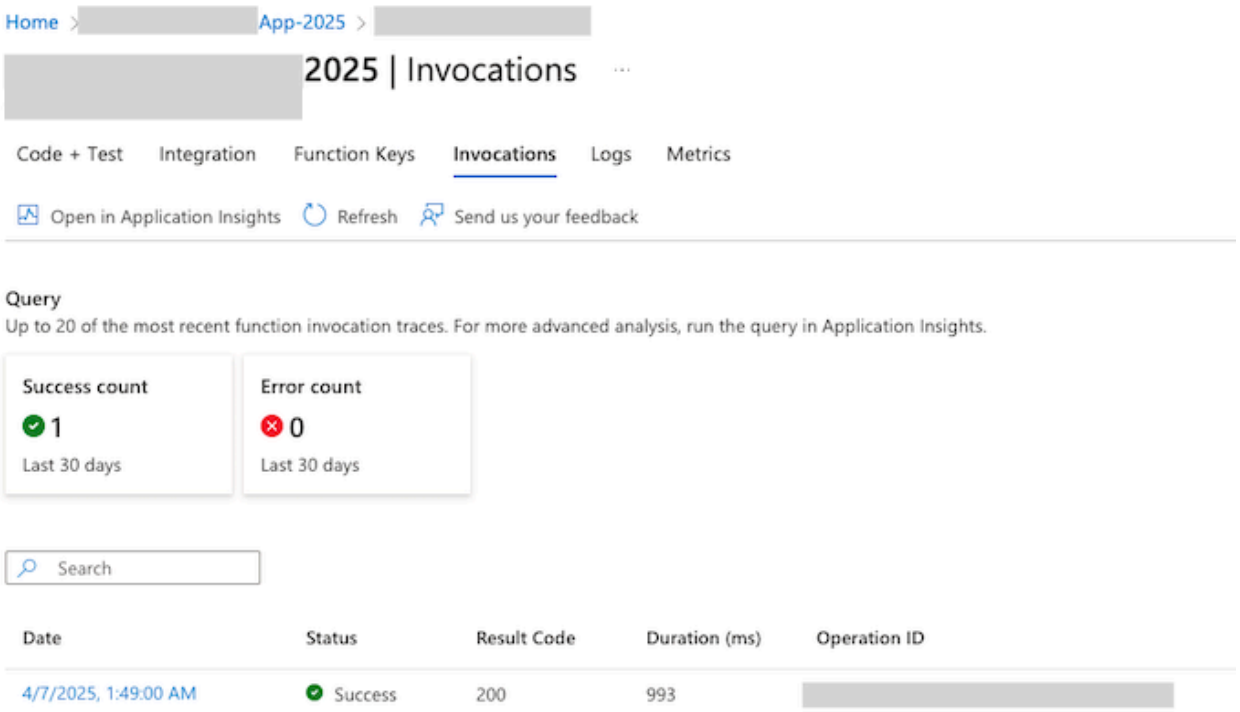

Figure 3 - Function apps - Invocations logs

This kind of extra visibility can be very important for investigations that require further details or cases in which the main log sources were unavailable or insufficient.

Azure Automation Accounts

The dedicated logs of Azure Automation Accounts have even more potential to play the role of crucial forensics artifacts in incident investigations. Besides the classic Azure activity logs related to those, like:

- Write an Azure Automation runbook draft
- Create or update an Azure Automation runbook
- Publish an Azure Automation runbook draft
- Etc.

However, while the logs above are important, they provide partial information and lack crucial details like the actual content that was written and/or published in the relevant automation account.

1. **View last test** - provides the option to look at the output of the last tested runbook execution. Can be very useful, in case the last execution was conducted by a

threat actor. Think of a case in which a JWT token was requested, it can be nice to find the actual token used by the attacker.

2. **Job History**—This section under the automation account's runbook of interest includes a list of past jobs and their status. By clicking on each, you should be able to see its output as well. As mentioned in the "view last test" option above, it can be very useful to get the actual output a threat actor saw while abusing the automation account.

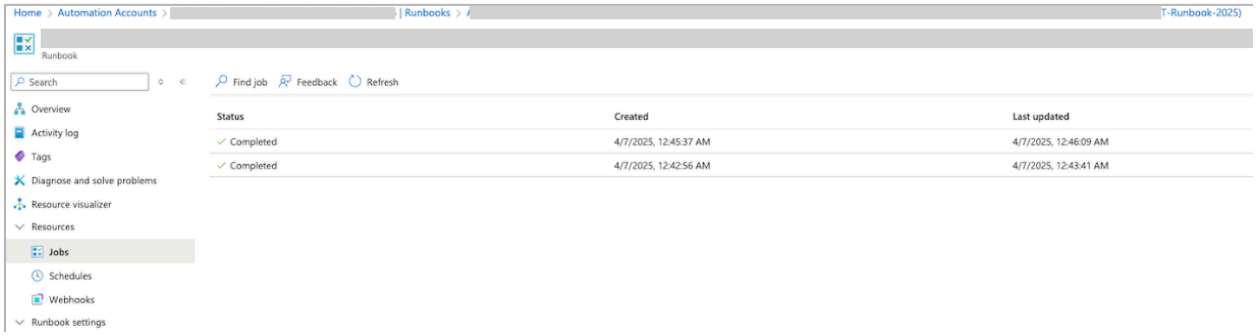

Figure 4 - Automation accounts job history

Azure Deployment Scripts

Azure Deployment Scripts are a first-class ARM resource that enables to run PowerShell or Bash scripts as part of an infrastructure deployment, without needing a separate VM or pipeline. They automatically provision a temporary container or sandbox, execute your script (e.g., to bootstrap resources or configure settings) and capture output/logs.

Azure Deployment Scripts can be abused for extracting JWT access tokens. It can be done in case the threat actor creates a new Deployment Script or modifies an existing one, and asks for an access token explicitly.

Azure Activity Logs provide some visibility into Deployment Scripts actions by operations like:

- *MICROSOFT.RESOURCES/DEPLOYMENTS/WRITE*
- *MICROSOFT.RESOURCES/DEPLOYMENTS/VALIDATE/ACTION*

Yet, that visibility is limited and might be insufficient for a root cause analysis as part of an IR investigation.

Besides those logs, there is additional information that might be considered as forensic artifacts to some extent. That information is available under the Azure Portal → Deployment Scripts management page.

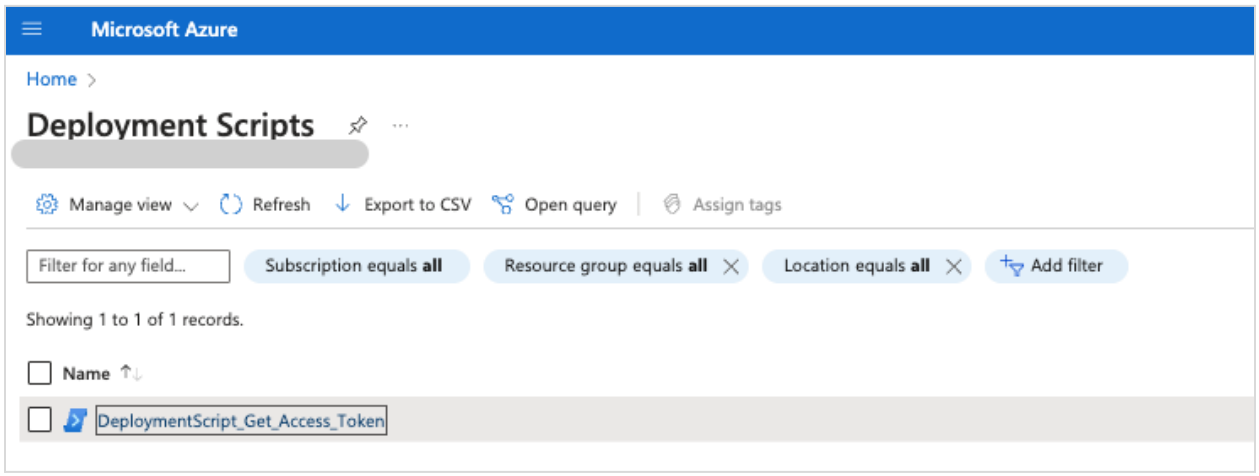

Figure 5 - Deployment Scripts management page under Azure Portal

For each script, we can find general details on it:

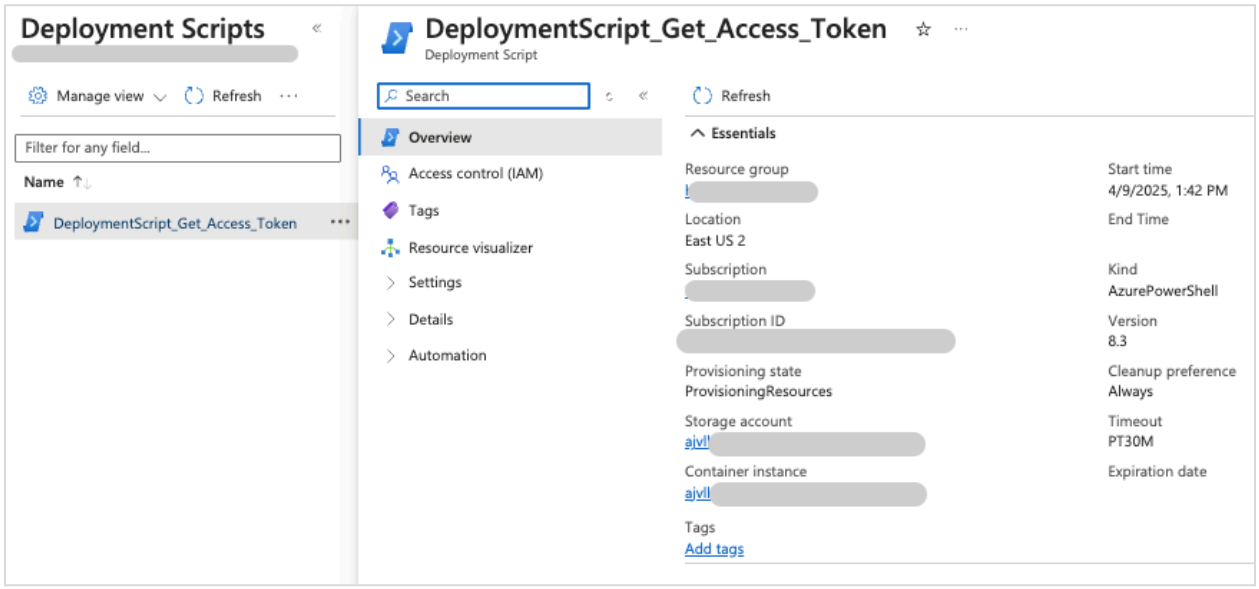

Figure 6 - Deployment Script details

In addition, the content of the script might be available as well. In the following example, we see that the command *Get-AzAccessToken* was executed from within the Deployment Script.

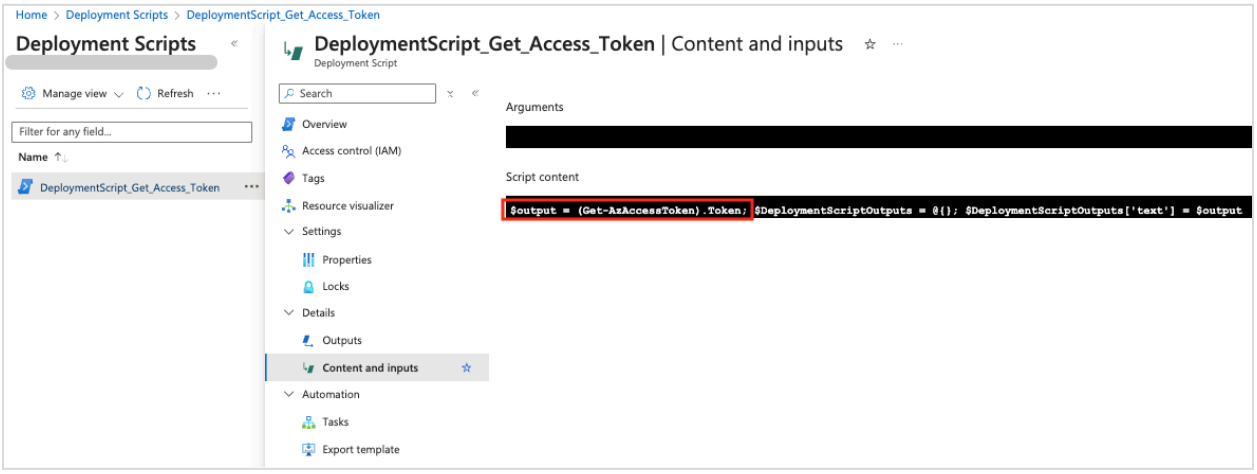

Figure 7 - Deployment Script content

Summary

This part of the research series explored key aspects of threat hunting and incident investigation in compromised Azure MIs cases. We examined methods for identifying and inventorying MIs across the environment. We introduced a collection of hunting queries designed to detect various suspicious behaviors and misuse patterns, and provided investigation guidelines tailored for MI abuse.

These guidelines leveraged primary Microsoft/Azure log sources and complementary telemetry to help analysts uncover activity context, map potential blast radius, and trace lateral movement paths. Together, these tools and techniques support a more effective and holistic defense strategy against identity-based threats in Azure, especially those involving Non-Human Identities (NHIs), which remain a critical and often overlooked part of the attack surface.

We hope this research sparked new ideas and offered practical tools defenders can use right away. It's our small contribution to helping the community stay one step ahead of identity-based threats in the cloud.

ABOUT HUNTERS

Hunters is transforming security operations with AI-powered automation, making it especially impactful for small SOC teams that need to maximize efficiency without large security budgets. As a leading next-gen SIEM, the Hunters SOC Platform is designed to go beyond traditional SIEM limitations by integrating Agentic AI, Copilot AI, machine learning, and graph-based correlation to automate detection, investigation, and response. Trusted by leading organizations such as Cimpress, OpenLane, and The RealReal.

Team Axon is an elite cybersecurity research team at Hunters, composed of seasoned professionals with deep expertise across various cybersecurity domains, including Incident Response, Digital Forensics, Red Teaming, Cloud Research, Detection Engineering, and Threat Research.

Notable research and contributions from Team Axon include the discovery of significant cybersecurity threats such as:

- **DeleFriend**: Discovery of a design flaw in Google Cloud Platform's domain-wide delegation potentially exposing Google Workspace to compromise.
- **VEILDrive**: Identification and analysis of threat campaigns leveraging Microsoft services and novel malware.
- **Malicious Chrome Extensions Campaign**: Early exposure of an active attack, providing timely indicators of compromise (IOCs) and technical details to the broader community.

Together, Hunters and Team Axon equip organizations with advanced capabilities to detect, investigate, and respond swiftly to emerging cyber threats.

To find out how Hunters can help your small SOC team, **reach out to us here.**