Azure security foundations

Christophe Parisel

Draft 0.2, 26 April 2024

linkedin.com/in/parisel

labyrinthinesecurity@github

Part 1:
IAM

Chapter 1:
Equivalencing Azure RBAC permissions

Introduction

In Azure, RBAC provides coarse and fine-grained access management for Azure resources, enabling users to have different permissions for different resources. However, with the increasing complexity and scale of cloud deployments, managing and verifying the correctness of such access control systems has become challenging.

In this chapter, we propose an approach to the formalization and reasoning about Azure RBAC fine-grained control plane permissions, leveraging congruence theory. By framing Azure RBAC permissions as a structured set, we define congruence relations that identify equivalent permissions. The congruence relation forms the basis for a logical model of Azure RBAC and will serve as a tool for reasoning about control plane permissions and their interactions in the next chapters of this book.

We introduce a formal approach to reasoning about Azure RBAC permissions, utilizing the mathematical foundation of congruences theory. The congruence theory offers robust tools for reasoning about equivalence relations and operations that are compatible with these relations.

Our approach aims to bring several key benefits for Azure customers:

1. Simplification of the management of RBAC permissions by identifying and grouping equivalent permissions.
2. Automation of permissions de-escalation.
3. Automation of role mining.
4. Enhanced understanding of the interaction between different permissions, which aids in maintaining least privilege access.
5. Improved predictability and verification of the behavior of RBAC permissions.
6. Increased IT security by ensuring the correct assignment and management of permissions.

Pre-requisites

The concepts of equivalence relations and congruences form the foundation of many branches of mathematics. They are tools that allow us to classify and reason about mathematical objects. In this section, we will introduce these concepts briefly and illustrate them with an example from number theory.

Equivalence Relations

An equivalence relation is a particular type of relation between objects that partitions a set into equivalence classes. In formal terms, a relation "~" on a set S is an equivalence relation if it satisfies three properties:

- Reflexivity: For every element a in S, $a \sim a$.
- Symmetry: For every pair of elements a and b in S, if $a \sim b$, then $b \sim a$.
- Transitivity: For any three elements a, b, and c in S, if $a \sim b$ and $b \sim c$, then $a \sim c$.

The set of all elements related to a particular element forms an equivalence class. These classes partition the set: every element belongs to exactly one equivalence class, and each class has at least one element.

Congruences

A congruence is a type of equivalence relation compatible with all operations on the set: hence it is a very useful property.

More formally, a relation "~" on a set S equipped with an operation ∘ is a congruence if it is an equivalence relation and, for any a, b, c, d in S, if a ~ b and c ~ d, then a ∘ c ~ b ∘ d.

An example of a congruence relation in number theory is congruence modulo n. For two integers a and b, we say a is congruent to b modulo n, written a ≡ b (mod n), if n divides the difference a - b. This relation is an equivalence relation and is compatible with both addition and multiplication, making it a congruence. This compatibility is the property that for all integers a, b, c, d, if a ≡ b (mod n) and c ≡ d (mod n), then a + c ≡ b + d (mod n) and a * c ≡ b * d (mod n).

These mathematical concepts provide a powerful framework to structure and understand complex systems, such as permissions. In the following sections, we apply these concepts to formalize and reason specifically about RBAC permissions.

The WAR partition of Azure RBAC

The set of Azure Permissions

In Azure Role-Based Access Control (RBAC), permissions are integral for defining access to the control plane of Azure resources. Each permission is a string that denotes a specific action or a set of actions on a particular resource type. The set of all Azure control plane permissions, denoted by P, is a large and complex set that captures all possible actions on all available resource types in Azure.

A permission in P is a string of the form 'Microsoft.provider/resourceType/op', where 'Microsoft.provider' is the resource provider, 'resourceType' is the type of the resource, and 'op' is the operation on the resource (read, write, action, delete).

Classification of Permissions

Operation types

To make use of the equivalence relation, we first classify permissions into operation types. This is achieved by examining the operation token in the permission string, i.e., the specific operation on the resource.

- Read Class: This class includes permissions where the operation is 'read' or ends with '/read'. For example, the permission 'Microsoft.Web/serverfarms/read' belongs to the read class. Furthermore, if the last segment of the permission string is a wildcard "*", but the segment before it indicates a 'read' op (for instance, 'Microsoft.Web/serverfarms//read'), it still belongs to the Read class as the wildcard does not change the nature of the permission.
- Write/Delete Class: This class is for permissions where the operation is 'write', 'delete', ends with '/write' or '/delete', or is ''. A wildcard "*" alone without a preceding 'read' segment would indicate a broader permission scope and is classified under write/delete.

- Action Class: This class includes permissions where the operation is 'action' or ends with '/action'. For example, the permission 'Microsoft.Web/serverfarms/join/action' belongs to the action class.

These operation types provide us with a way to reduce the complexity of the set of permissions by grouping them into broader categories. This categorization is at the core of defining the equivalence relation and the subsequent congruence.

The union-find data structure

To utilize operation classes efficiently, we employ the Union-Find data structure. The Union-Find data structure is a fundamental tool used in computer science to keep track of a partition of a set into disjoint subsets. It provides two primary methods:

- Find: Determine which subset a particular element is in. This method can be used for determining if two elements are in the same subset.
- Union: Join two subsets into a single subset.

In our context, the 'Find' method helps us identify the operation class of a permission. The 'Union' method allows us to merge permissions into the same class if they are equivalent (belong to the same class).

The efficiency of the Union-Find algorithm is enhanced by using a technique called Union by Rank. The rank of a subset represents an upper bound for the height of the trees representing the subsets. When two subsets are unioned, the one with the lower rank is attached to the root of the one with a higher rank. This technique helps keep the tree balanced, thus improving the efficiency of the 'Find' method.

By combining the classification of permissions, the equivalence relation, the Union-Find algorithm, and Union by Rank, we are able to efficiently partition the set of permissions into equivalent classes of actions.

With this classification at hand, we can now define the equivalence relation based on the operation type.

Equivalence Relation

We define an equivalence relation "~" on a set of Azure permissions based on a particular property – the type of the operation. That is, two permissions are equivalent if they belong to the same operation class (read, write/delete, or action).

We can show that "~" is an equivalence relation by proving that it satisfies reflexivity, symmetry, and transitivity:

- Reflexivity: Every permission is equivalent to itself because it belongs to the action class as itself.
- Symmetry: If permission p1 is equivalent to p2 because they belong to the same operation class, then p2 is also equivalent to p1 because the class of p2 is the same as that of p1.
- Transitivity: If permission p1 is equivalent to p2 and p2 is equivalent to p3 (because they all belong to the same class), then p1 is also equivalent to p3 because it belongs to the same class as p3.

WAR partition

Consequently, the space of Azure permissions is partitioned into three equivalence classes: W(rite), A(ction) and (R)ead.

Comparing roles with a congruent relation

Note that the classification of operations can be thought as hierarchical, with Write taking precedence over Action, which in turn takes precedence over Read. For instance, if two permission strings s1 and s2 were to be compared, with the first containing 'read' and the second 'delete', the first string would take precedence: s1 > s2.

As we have partitioned the set of Azure permissions into W,A and R classes using "~", we now enrich this relation with a specific function to consider the hierarchical structure of Azure RBAC. This will facilitate management and analysis of permissions. For that purpose, we introduce the combine() function. and show that "~" is a congruence with respect to combine().

combine() is a binary function that takes two permissions and merges them. The result is a permission which has the 'highest' privilege of the two input permissions.

In terms of our defined equivalence classes, W (write) is considered the highest privilege, followed by A (action), and lastly R (read).

Here is a sample implementation of *combine()* in Python:

```python
def combine(permission1, permission2):
    # Order of precedence: write/delete > action > read > unknown
    precedence = {'unknown': 0, 'read': 1, 'action': 2, 'write/delete': 3}

    class1 = classify_permission(permission1)
    class2 = classify_permission(permission2)

    # Return the permission with the highest precedence
    if precedence[class1] > precedence[class2]:
        return permission1
    else:
        return permission2
```

The *classify_permission* function called within *combine* is defined as such:

```python
def classify_permission(permission):
    if permission == "*" or permission.endswith("/*"):
        return "write/delete"

    segments = permission.lower().split("/")

    if any("write" in segment or "delete" in segment for segment in segments):
        return "write/delete"

    if any("action" in segment for segment in segments):
        return "action"

    if any("read" in segment for segment in segments):
        return "read"

    return "unknown"
```

We claim that combine()is congruent with respect to "~". In mathematical terms, this means:

For any permissions p1, p2, q1, q2,

if p1 ~ p2 and q1 ~ q2, then the results of combining p1 with q1 and p2 with q2 are equivalent: combine(p1,q1) ~ combine(p2,q2)

Proving congruence of "~" with the combine function

Claim

Let p1, p2, q1, q2 be permissions such that p1 is equivalent to p2 and q1 is equivalent to q2. If we combine p1 with q1 and p2 with q2, the results are equivalent.

Proof

By definition of our equivalence relation, permissions p1 and p2 belong to the same class. Similarly, q1 and q2 belong to the same class.

Let's denote the classes of p1, p2 as [p] and the classes of q1, q2 as [q]. Then, due to the rules of the combine operation:

combine(p1, q1) = [p] if [p] > [q]

combine(p1, q1) = [q] if [q] >= [p]

And similarly,

combine(p2, q2) = [p] if [p] > [q]

combine(p2, q2) = [q] if [q] >= [p]

Since [p] and [q] are the same classes for both operations, it follows that combine(p1, q1) and combine(p2, q2) belong to the same class and hence are equivalent.

Therefore, combine holds the congruence property with respect to our defined equivalence relation on permissions.

Associativity and commutativity of the Combine Function

Two other key properties of combine() are that it is associative and commutative:

Associativity: for any p1, p2 and p3, combine(p1,combine(p2,p3))=combine(combine(p1,p2),p3)

Commutativity: for any p1 and p2, combine(p1,p2)=combine(p2,p1)

This is due to the simple total order that we imparted to the equivalence classes: W > A > R.

Roles comparison

Suppose we have two roles r1 and r2, and we want to determine whether they belong to the same realm which can be the "Administrator" realm (where Write operations are allowed) , the "User" realm (where Action operations are allowed, but no Write operations), or the "Auditor" realm (for read only access).

Simple roles can be represented as sets of permissions. For instance, r1={p1,p3,p5} and r2={p2,p4}

Because combine() is associative and commutative, all applications of combine() on r1 and r2's permissions are the same.

For r1, we have combine(p1,combine(p3,p5))=combine(combine(p5,p1),p3).

Because "~" is congruent with respect to combine(), the application of combine() on the permissions of any role will return either Write, Action or Read consistently for this role.

Therefore, using the congruence relation, we can classify any role into Administrator, User or Auditor realm, and we can compare roles pairwise.

Other functions congruent with "~" useful for IAM

In the context of RBAC, several operations reflecting common tasks in managing permissions compatible with a congruence relation can be:

1. Union: This operation could represent the combining of permissions from multiple roles or groups. For example, if a user is part of two roles, each with a set of permissions, the total set of permissions for the user would be the union of these two sets. Since combine() is a congruence, the union operation preserves the equivalence classes.
2. Intersection: This operation could represent finding the common permissions between two roles or groups. For example, to find permissions that are common in two different roles, you could use the intersection of the two sets of permissions. Again, since combine() is a congruence, the intersection operation preserves the equivalence classes
3. Expansion: This operation could represent the interpretation of wildcard or other shorthand notation in permissions. For example, if you have a permission that grants read access to all

resources ('*/read'), the expansion operation would translate this into a set of read permissions for each individual resource.

4. Contraction: The inverse of expansion, this operation would replace a set of permissions with a shorthand notation if possible. For example, if a user has read permissions on every individual resource, the contraction operation could replace this with a single permission granting read access to all resources ('*/read').
5. Composition: This operation could represent the chaining or sequencing of permissions. This would be especially relevant in systems where the order of operations matters, such as in a workflow where certain actions must be performed before others.

As an example, regardless of p1 and p2 belonging to the same equivalence class or not, expand(combine(p1, p2)) is in the same equivalence class as combine(p1,expand(p2)).

How congruence helps reasoning about roles

Through our analysis, we identified an equivalence relation "~" which was found to be congruent over a set of permissions equipped with the combine() operation. Although "~" partitions this set into just three equivalence classes (Write, Read and Action), the congruence offers enough expressiveness to formally verify statements pertaining to roles represented as set of permissions.

This is of significance because, if "~" were merely an equivalence relation without being a congruence, we wouldn't be able to guarantee that the comparison of permissions in distinct roles preserve the equivalence classes. Consequently, comparing roles could result in unpredictable and inconsistent outcomes.

Limitation

Azure roles are more complex that simple sets of permissions: Azure role assignments operate on scopes, our simple congruence is not powerful enough to capture scopes. We will see in chapter 3 how to overcome this limitation.

Takeaways

"~" is the critical relation that we will leverage in the second chapter of this book: theory of de-escalation.

Congruence is the critical property that we will leverage in the third chapter of this book: unsupervised role mining.

Chapter 2:
Theory of de-escalation

Introduction

Measuring the extent of an Entra principal's right in Azure is an absolute necessity. Without such a metrics, cloud customers cannot determine whether assigned roles and permissions meet least privilege principles. However, doing so accurately with the current native tools remains challenging.

In this chapter, leveraging the combine property of chapter 1, we propose,

1. a way to measure RBAC rights and to compare them with a desired value;
2. a scalable way to customize SPN role assignments so that rights get de-escalated to the desired value in an automated way, using machine learning techniques.

Pre-requisites

N-tuples

An n-tuple is an ordered set of n items $x = (x1, \dots xn)$.

For integer n-tuples, one can define the L1 norm as $||x||_1=|x1|+\dots+|xn|$.

Let's consider the absolute value of the sum of items in an integer n-tuple: $|x1 + \dots xn|$

Recall that the absolute value follows the triangle inequality:

For all integers x1 and x2, $|x1+x2| \leq |x1|+|x2|$

Therefore, the sum is bounded from above by the L1 norm:

For all x, $|x1 + \dots + xn| \leq ||x||_1$

Due to this property, let's call this summing operation L1-minus.

Is L1-minus a norm? Yes, because the absolute value is a norm. Let's use the symbol $||x||_-$ to depict it.

Induced distances

The induced distance d(,) of a norm is defined as: $d(x,y)=||x-y||$

It is well known that the induced distance of L1 is a distance, but is it true for L1-minus? In general, no. To see why, consider the following tuples: x=(10,12) and y=(5,17),

- $||x||_-=|x_1+x_2|=|10+12|=22$
- $||y||_-=|y_1+y_2|=|5+17|=22$
- $d(x,y)=||x-y||_-=|(x_1-y_1)+(x_2-y_2)|=|(10-5)+(12-17)|=|5-5|=0$

We see that L1-minus is not point separating: if the distance between two tuples x and y is zero, it doesn't mean that x is equal to y.

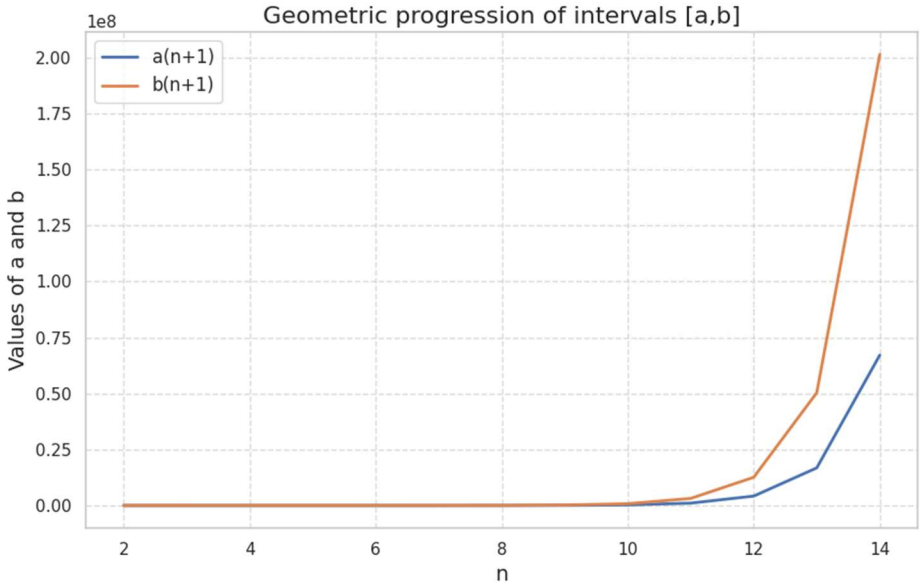
Now, if the items contained in the n-tuple are defined over non-overlapping (except at origin O) well-chosen intervals, then we can make L1-minus point-separating. There are many ways to go, here is the one we will resort to throughout this chapter:

Consider intervals $I_i = [a_i, b_i]$ with the extra condition that $\forall p,q \in [a_i, b_i] \cup O$, $p \neq q$: $|p-q| > \Sigma\, b_{i-1}$

Let's call this condition "separation of concerns".

Setting $p=a_i$ and $q=0$, we see that all intervals are disjoints since $a_i > \Sigma\, b_{i-1}$

Setting $p=a_i$ and $q=b_i$, we see that the length of intervals follows a geometric progression. The growth is exponential, as can be shown from the following progression which adheres to the separation of concerns:



Exponential growth of intervals $I_i=[a_i,b_i]$, with $a_0=1$, $b_0=3$, $card(I_i)=3$ for all i,

$$a_{n+1}=\Sigma b_n$$

$$b_{n+1}=card(I_n)*\Sigma b_n=3\Sigma b_n$$

$$b_{n+1}-a_{n+1}=2\Sigma b_n$$

Our set of n-tuples is going to be built from the cross product of the n intervals $I_1, \ldots I_n$, plus the origin point O: $x=(x1,\ldots xn) \in I_1 \times \ldots \times I_n \cup \{0, \ldots, 0\}$

Let's prove that this version of L1-minus is point separating. Take two 1-tuples x and y, and suppose $d(x,y)=0$ (i)

We start at n=1, and consider the single interval $I1 = [a_1,b_1]$. By definition, $d(x,y)= |x1-y1|$. For n=1 L1-minus equals L1, the absolute value. So x=y=0.

The proof is finished for n=1.

We switch to the general case "n".

$d(x,y)= |(x_1-y_1)+\ldots+(x_n-y_n)|$

Let's define function $p(x,y)= (x_1-y_1)+\ldots+(x_{n-1}-y_{n-1})$

We have $d(x,y)=|p(x,y)+(x_n-y_n)|$ thus $d(x,y)=0$ iff $p(x,y)=y_n-x_n$ (ii)

Let's show that $p(x,y)< y_n-x_n$, to disprove (ii).

Without loss of generality, consider $y_n>x_n$ (otherwise, we carry on the reasoning with $d(y,x)$).

Setting $p=y_n$ and $q=x_n$ in the extra condition on I_n, $\{ \forall p,q \in I_n, |p-q| > \Sigma b_{n-1} \}$, we get $(y_n-x_n)> b_1 + \ldots + b_{n-1}$ (iii).

Observe that, by definition of b, $(x_1-y_1) < b_1, \ldots, (x_{n-1}-y_{n-1}) < b_{n-1}$

So $p(x,y)< b1+\ldots+bn-1$

Which we rewrite as $p(x,y)<(y_n-x_n)$ (using condition iii)

Consequently, $d(x,y) \neq 0$ (iv)

(iv) is in contradiction with (i), our initial hypothesis (that $y_n>x_n$) is wrong, hence $x_n=y_n$.

We repeat the reasoning at level n-1 and conclude that $x_{n-1}=y_{n-1}$.

We carry on until we get $x_1=y_1$

Finally, $d(x,y)=0$ implies $x=y$.

The proof is finished.

The WAR distance

In chapter 1, we explained a core foundation of Azure RBAC: the WAR partition. Using the combine congruence, each atomic permission is uniquely categorized as belonging to the (W)rite,(A)ction or (R)ead equivalence class.

An Azure role assignment is a partition of (combined) permissions granted along the W, A and R equivalence classes on a scope ‘s’. For each role, let’s define the 3-tuple $(w,a,r)_s$ where w, a and r are natural numbers belonging to the following equivalence classes W, A and R:

W = { 950, 900, 850, 800, 750, 700, 600, 500, 400, 300, 200, 100 }

A = { 45, 40, 35, 30, 20, 10 }

R = { 4, 3, 2, 1 }

We verify that the minimum difference of any two distinct integers in W, 50, is greater than the sum of the higher bounds for A (45) and R (4), which is 45+4=49.

Likewise, the minimum difference of any two distinct integers in A, 5, is greater than the higher bound of R (4).

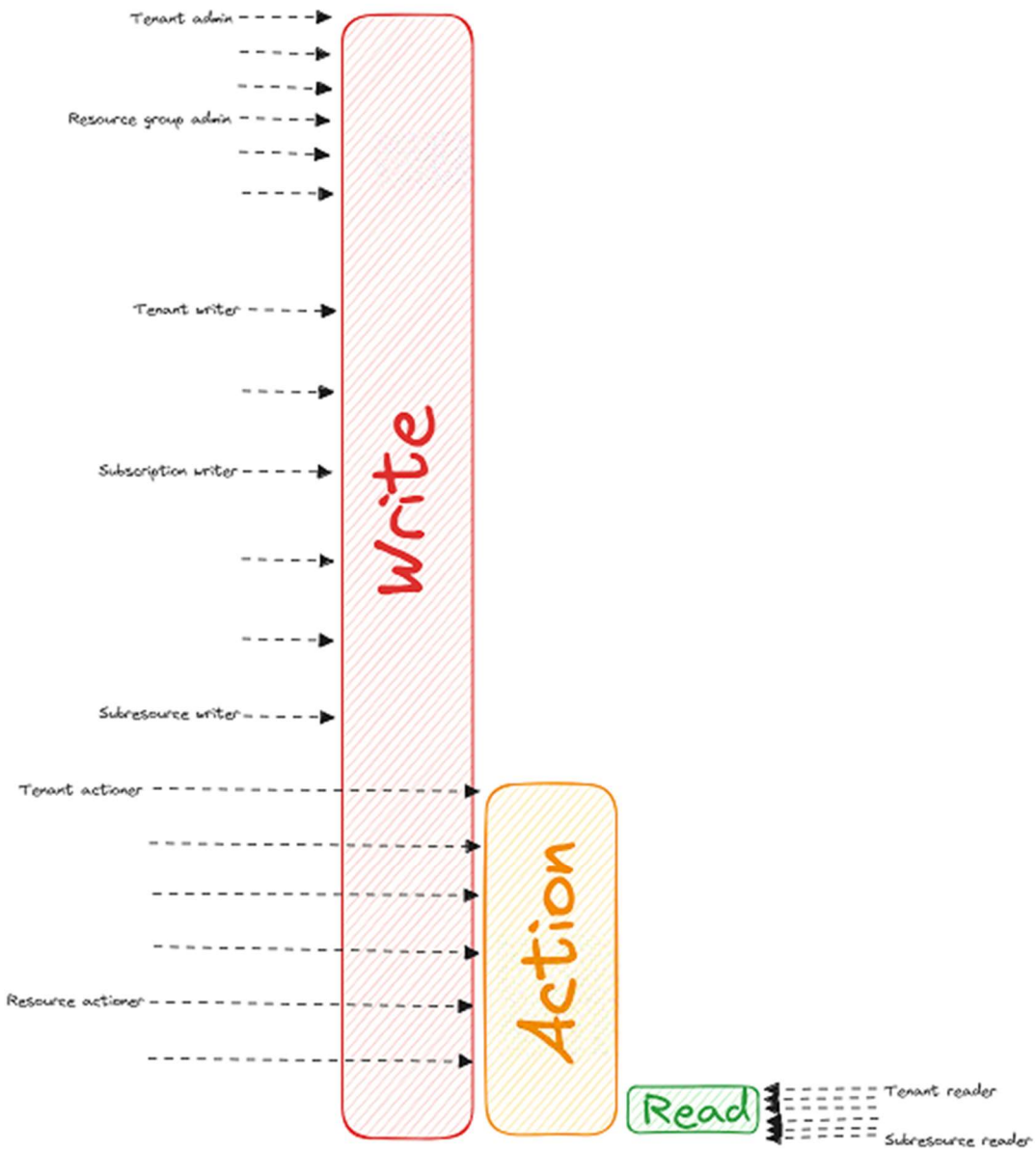
So, the induced L1-minus distance over $(w,a,r)_s$ permissions assigned to any Azure scope s is point separating and is indeed a distance. We call it the WAR distance.

The WAR distance ranges from 999 (950+45+4) to 0.

999 is the distance d(S,O) between S, the superadmin tuple with maximum (tenant) permissions, to O, the origin tuple (0,0,0) which is granted no permissions at all.

In the context of the WAR distance, we can think the “separation of concerns”property as a way to distinguish 3 independent spheres of influence operating imperviously from one another, without risk of a head-on collision:

- the (W)rite ops area
- the (A)ction ops area
- the (R)ead ops area



Separation of concerns in the L1-minus norm of the WAR distance

Using the WAR distance to calculate the distance between any two Azure principals

The norm of tuple $(w,a,r)_s$ at any scope 's' is a natural number attributed according to the following algorithm:

For each principal's role assigned to scope s, we identify any wildcard permission, excluding role assignments: if the scope is the whole tenant, we choose 950. If it is a management group, we choose 900. If it is a subscription or a resource group, we choose 850 or 800 respectively. If it is a resource or a sub-resource, we choose 750 or 700 respectively.

If no wildcard permissions are attached to the assigned role, we look at other putative W permissions, still excluding role assignments: if the scope is the whole tenant, we choose 600. If it is a management group, we choose 500. If it is a subscription or a resource group, we choose 400 or 300 respectively. If it is a resource or a sub-resource, we choose 200 or 100 respectively. If there are no W permissions, we choose 0.

We then switch to A permissions: if the role assignment is attached such permission at tenant level, we choose 45. If it is a management group, we choose 40. If it is a subscription or a resource group, we choose 35 or 30 respectively. If it is a resource or a sub-resource, we choose 20 or 10 respectively. If there are no A permissions, we choose 0. Note that we don't need to exclude role assignments here, because these are only granted through W (or wildcard) permissions.

Finally, we consider R permissions. If the principal's role assignment is attached such permission at tenant scope or at the management group scope, we choose 4. If it is a subscription or a resource group, we choose 3 or 2 respectively. If it is a resource or a sub-resource, we choose 1. If there are no R permissions, we choose 0. Once again, we don't need to exclude role assignments here, because these are only granted through W (or wildcard) permissions.

Silhouette of a principal

In practice, we coalesce the WAR norms of all role assignments of a given principal into a single aggregated "virtual" norm, its "silhouette".

To coalesce assignments in a meaningful way, we consistently take the maximum scopes 'sw', 'sa' and 'sr' at which a W, A or R permission operates in each role assignment. The resulting silhouette tuple features a WAR norm in 3 independent scopes: $(w,a,r)_{sw,sa,sr}$

Here is an illustration of measuring the WAR norm of a principal's silhouette tuple:

Scope	Has superadmin permission	Has write permission	Has action permission	Has read permission
Tenant	+950	+600	+45	+4
Management group	+900	+500	+40	+4
Subscription	+850	+400	+35	+3
Resource Group	+800	+300	+30	+2
Resource	+750	+200	+20	+1
Subresource	+700	+100	+10	+1

Thus, a silhouette granting superadmin rights on a subresrource and write + action + read permissions on a tenant would have a WAR norm of 700+45+4=749.

We don't add 600 (tenantwide write permission) to the silhouette because it is superseded by superadmin rights on a subscription (850). Both superadmin and write permissions belong to the W equivalence class, in a staged way ranging from 950 (tenant admin) to 100 (subresource writer).

Note that

1. Being superadmin on a mere subresource takes precedence over being tenantwide writer,
2. Our measurement (749) is strictly lower than resource superadmin's silhouette (750),
3. We don't care what tenants, subscriptions, etc mean from a customer business perspective. These are just resource placeholders. They may not even be nested, but rather, completely independent. What matters is scope maximization.

Takeaway: it is possible to compute the distance of any two principals' RBAC roles, by comparing their silhouettes.

This distance is focused on control plane operations, not IAM operations.

The Delegate & Assign distance

The previous distance is useful for measuring the extent of purely operational rights. We purposefully excluded role assignments, which are IAM management roles.

Let's build another independent distance for accurately measuring the extent of IAM roles.

Role assignment delegation, currently in preview, opens new avenues to define a wider range of intermediate IAM role than currently supported in Azure and Entra: when these intermediate roles are implemented by Azure customers, the D & A distance will make it possible to measure the extent of IAM roles quite precisely for the first time in Azure's history!

Let's start from an Entra principal 'p' who is assigned an Azure role with the ability to assign other Azure roles. What kind of permissions is this p entitled to grant?

1. Role assignments, in a kind of recursive fashion ("an assignment granting role assignments"),
2. Operational write (including superadmin),
3. Operational action,
4. Operation read,
5. Any combination of the above.

To which population?

1. BRONZE: A group of principals excluding p herself, which p doesn't manage,
2. SILVER: A group of principals including p, which p doesn't manage,
3. GOLD: An arbitrary population of principals (in this case, p is not only role assigner, but doubles up as a group membership manager),
4. Any combination of the above.

Note that the construction of this distance is largely independent from the underlying implementation of the Microsoft Preview (which is based on *constrained conditions*). Should the design be replaced with another logic, the core D&A distance would remain usable.

Here are the main differences between the WAR distance and the D&A distance:

- The WAR distance computes a numeric value for an Azure role assignment which is a pair (role definition, scope), whereas the D&A distance computes a numeric value for a pair (role definition, metallicity of Entra principals) which has no existence in Entra or Azure;
- The WAR distance depends on the WAR partition and scope (or scopes, in case of aggregated assignments), the D&A distance depends on a repartitioning of the permissions space along 4 equivalence classes (W role assignments + W excluding role assignments + A +R) and the metal classification of target population(s).

Let's define the 4-tuple $(da,w,a,r)_m$ of a metal 'm' where da, w, a and r are 4 natural numbers belonging to the following DA, W, A and R sets:

DA = { 192, 128, 64 }

W = { 48, 32, 16 }

A = { 12, 8, 4 }

R = { 3, 2, 1 }

As before, we can easily check that the minimum distance of any two distinct elements in DA, 64, is greater than the sum of the higher bounds of W (48), A (12) and R (3), which is 48+12+3=63.

The same reasoning holds for W, A and R.

So, D&A, the induced L1-minus distance over $(da,w,a,r)_m$ tuples is indeed a distance.

Observe that the D&A distance is nothing else than the geometric progression proposed in the pre-requisites:

$$a_{n+1}=\Sigma b_n$$
$$b_{n+1}=card(I_n)*\Sigma b_n=3\Sigma b_n$$

Tee D&A distance ranges from 255 (192+48+12+3) to 0. 255 is the distance d(S,O) between S, the superadmin 4-tuple with maximum permissions, to O, the origin 4-tuple (0,0,0,0) which grants no permissions at all.

Using the D&A distance to calculate the IAM distance between any two Azure principals

The norm of tuple (da,w,a,r) is a natural number attributed according to the following algorithm:

For each principal's role assigned to any scope, we identify the ability for assignees to permit role assignment permissions in the W partition of Microsoft.Authorization' resource provider. If such a permission exists, we identify to which assignees the principal may assign such "roles-assigning" roles: this step depends on the implementation of Azure roles delegation. Currently, this requires examining IAM conditions, which are written in a special language inherited from ABAC conditions (https://learn.microsoft.com/en-us/azure/role-based-access-control/conditions-format)

If this population is GOLD, we choose 192. If it is SILVER, we choose 128. Lastly, if it is BRONZE, we choose 64.

Next, we review the ability for assignees to grant control plane operations in the following order: first we identify write operations (W), then action operations (A) and finally read operations (R).

If, for example, GOLD assignees may grant write operations, we choose 48. If they are SILVER, we choose 32. If they are BRONZE, we choose 16. Likewise for action operations and read operations.

The calculation is additive:

- (additivity along the permissions axis) : if a SILVER population is entitled to grant role assignments and read operations, we choose 128 + 2 = 130.
- (additivity along the population axis): if a BRONZE population is entitled to grant write operations and a GOLD population is entitled to grant action operations, we choose 16+3=17.

Here is an illustration of measuring the D&A norm of a principal's IAM tuple:

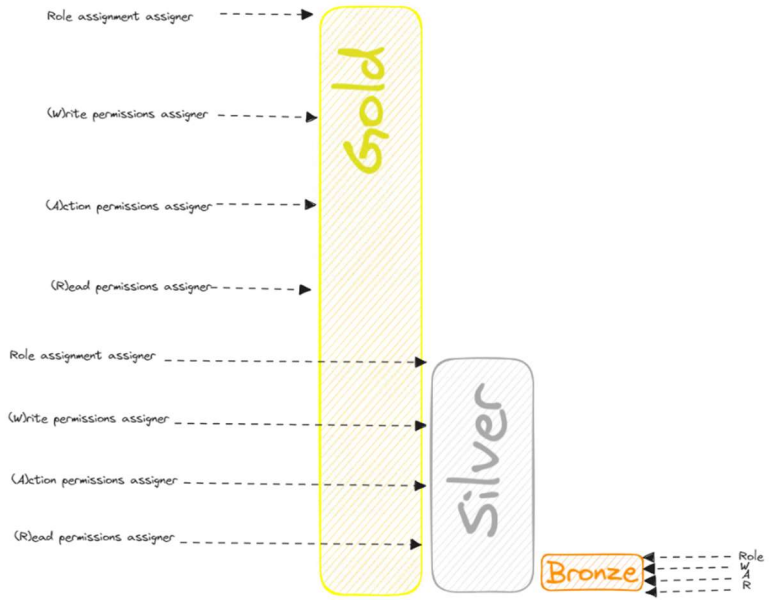
Assignees	Assignee may assign roles	Assignee may assign W ops	Assignee may assign A ops	Assignee may assign R ops
GOLD	+192	+48	+12	+3
SILVER	+128	+32	+8	+2
BRONZE	+64	+16	+4	+1

From the above table, it should be obvious that the maximum norm is 192+48+12+3=255 corresponding to an IAM role granting all permissions to arbitrary principals. This corresponds to the default built in **Owner** role.

The D&A norm of the built-in **User Access Administrator** role is 192. The very high value attached to this role is consistent with the facts that user access administrators can elevate their privileges to perform arbitrary control plane operations in a scope (which we don't care), and that they can grant such arbitrary powers to uncontrolled third parties.

A corporate **User Access Auditor** is likely to be assigned D&A norm 0 and WAR norm 4 (an operational, control plane permission to read role assignments in Microsoft.Authorization**).**

Takeaway: for any two principals, it is possible to compute the distance of their respective IAM roles, regardless of the scopes they manage.



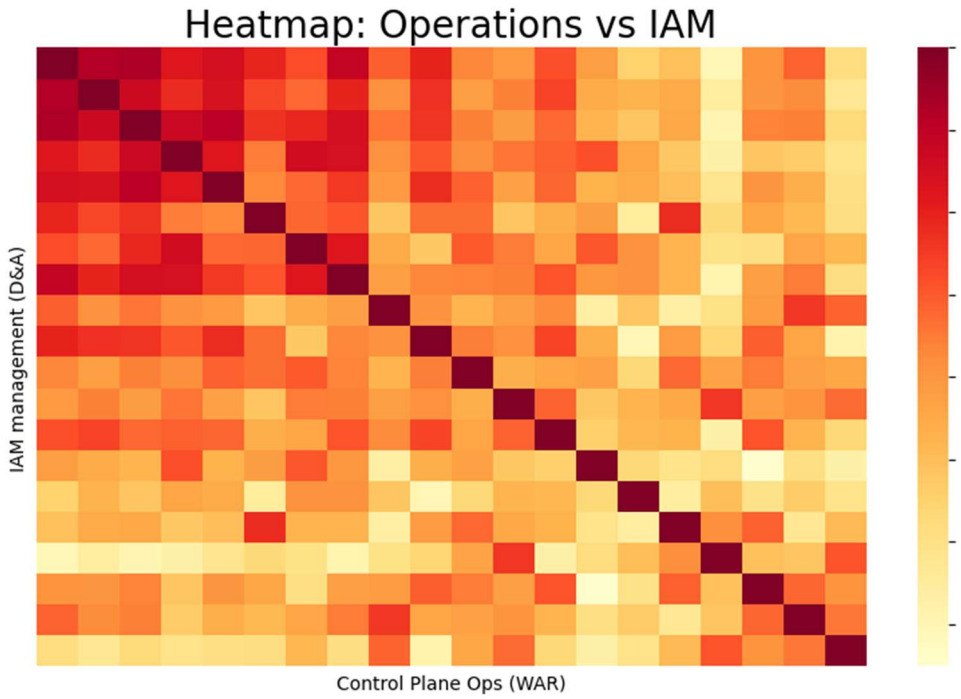
Separation of concerns in the L1-minus norm of the D&A distance

Solving the long-standing intrication problem in Azure

Up until the preview of role assignment delegation in 2023, building a D&A distance would have made little sense, because in practice only users with Owner or User Administrator roles have the right to assign roles, and these two built-in roles are themselves highly privileged:

- IAM roles and control plane operations roles are intricated;
- Segregation of these administrative duties is difficult.

With the help of the WAR and D&A distances, we may set up a heatmap to illustrate the typical correlation of IAM roles and administrative roles in an Azure tenant, it would look like this:



Intricated ops rights and IAM rights in Azure (artist view, not actual data)

As Azure customer adopt roles delegation, attribution of IAM permissions will become gradually more nuanced: the heatmap "concentration of power" is going to spread out. The D&A distance will become a prime instrument to witness de-intrication factually and with great accuracy

Distance based de-escalation

Consider the two major IAM sources of truth in the Microsoft Cloud ecosystem:

1. Entra, the golden source of principals, role definitions and role assignments. This directory can be queried using Microsoft Graph API and, for Azure RBAC, may also be queried with Azure Resource Graph API;
2. Log Analytics Workspace, the ground truth of (W)rite and (A)ction control plane operations issuance and outcome. LAW's resource provider may be queried using the Azure Resource Manager API.

Let's introduce two silhouettes of Azure principals:

- The outer silhouette is the WAR norm of a principal's extent of Azure rights considered from Entra's source of truth;
- The inner silhouette is the WAR norm of aprincipal's extent of Azure rights considered from LAW.

The foundational ideas of distance-based de-escalation can be expressed as follows:

- Sources of truth being authoritative, any misalignment between any two of them must be treated as a discrepancy in the model,
- A discrepancy is a gap between the outer and inner silhouettes of a principal. Discrepancies arise because Entra tolerates wildcards and arbitrary scope assignments, whereas LAW sees named operations scoped at resource level.
- Since we can measure silhouettes using the WAR norm, we have a formal and reliable way to measure the breadth of the gap using the WAR distance,
- The gap defines a de-escalation range as the interval between an absolute lower bound (the inner silhouette), and an absolute upper bound (the outer silhouette),
- A de-escalation tactic is the process of choosing, for any principal, a target silhouette as a point within de-escalation range. The de-escalation effort of this principal is the difference between the outer silhouette and the target silhouette, a direct measurement the WAR distance
- A de-escalation strategy is the process of prioritizing de-escalation for all principals, by ordering outer silhouettes in descending criticality and constraining de-escalation efforts using IT security risk reduction objectives and financial constraints,

In Azure, it is possible to implement a formal IAM de-escalation strategy because, for any principal:

1. the outer and inner silhouettes can be determined using the L1-minus WAR norm,
2. the de-escalation range can be calculated as the WAR distance between outer and inner silhouettes,
3. the de-escalation effort can be set by picking any point belonging to the de-escalation range.

Note that the inner silhouette defines the absolute lower bound of the target silhouette. In practice, it is unreasonable to attempt setting the target silhouette to the inner silhouette because the latter always operates at resource scope by definition of operations in Azure activity logs.

The WAR distance provides a clear edge when dealing with many principals, or when many privileged roles "look the same", because one can leverage the natural ordering of distances to streamline de-escalation at the scale of the enterprise:

- Quantity: principals can be grouped into risk-based bins, ranging from high-risk (high WAR norm) to low risk (low WAR norm);
- Quality: roles which look similar are often hard to qualify: which role is more risky than which other is prone to subjectivity, inconsistency and interpretation. The WAR distance provides an unambiguous measurement.

SPN rights de-escalation

Automation accounts is one of the most overlooked population when it comes to enforcing strong RBAC. In Azure, Service account SPNs and Managed Identities are no exception. The risk of unmanaging automation accounts is amplified by the fact that they are usually very numerous, and they are generated automatically: in most corporations, they typically scale with the number of assets and subscriptions.

There is one feature, however, that IAM professionals can leverage when dealing with automation accounts: their deterministic behavior. This is really the most outstanding feature of automation.

Machine Learning excels at solving scalability, deterministic problems. We exploit this fact to de-escalate Azure SPNs.

Principles of scalable SPN rights de-escalation

Corporations commonly deploy hundreds, if not thousands, of SPNs to automate their Cloud operations. When determining RBAC distances, it is therefore critical to leverage this deterministic behavior and to reason at the grain of whole SPN clusters, not individual clusters.

As we explained in the first section, one may aggregate all roles of a principal into a silhouette to get a holistic view of the principal's control plane rights : this takes the form of a single measurement, the WAR norm.

Let's expend this idea further and consider groups of SPNs: if we can define clusters of similar SPNs, we may generate a cluster silhouette that maximizes the individual silhouettes of a whole SPNs population.

Takeaway: For SPNs, the WAR distance is calculated at cluster level.

This is line of thoughts doesn't hold for human principals, because business-driven role models are implemented in a *top-down fashion* with Entra groups membership. Automation accounts, however, are rarely business driven, they are driven by technological needs to circumvent purely technical limitations. In that context, going *bottom-up* makes a lot of sense.

K-Means

K-Means is an unsupervised machine learning algorithm used for grouping multi-dimensional vectors which are close to one another.

We can consider unsupervised machine learning algorithms as categorization "black boxes" which take inputs called features, and which output a single result called the prediction.

In our case, to group accounts by RBAC similarity, we must find some SPN-related features that we shape into the form of a multi-dimensional vector (one for each SPN). Then, we inject each vector into K-Means to "categorize" it. The output category is a cluster ID.

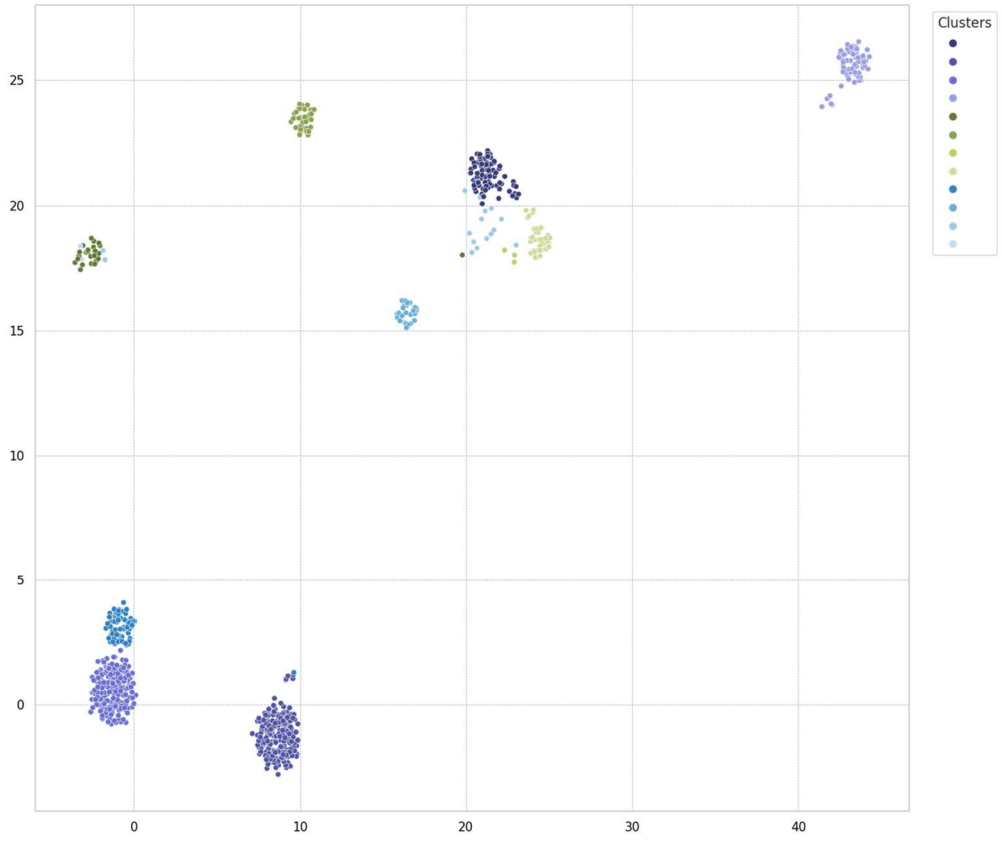
Features determination

There are many ways to find suitable features for Azure SPNs grouping. It is under active research. What seems to work well is by looking at all the role assignments of an SPN and pick the scope, the WAR permissions and their associated resource providers.

The various resource providers can then be one-hot-encoded before being added to the multi-dimensional vector. The scopes and WAR permissions do not need to be encoded since they are well-defined categories:

- Scopes belong to the Tenant, Management Group, Subscription, Resource Group, Resource, Subresource categorial set;
- Permissions belong to the Write, Action and Read categorial set. (Write can be further subdivided into Superadmin and not superadmin for a better resolution).

To illustrate the efficiency of k-means, the picture below is a 2D UMAP projection of a k-means clustering involving hundreds of SPNs.



UMAP projection of 12 clusters of Azure SPNs

A great feature of K-means is that we can pre-set the number of categories that we want.

Since we don't know the optimal number of categories that we need for SPNs clustering, we use a steepest descent algorithm to find an optimum. This optimum is usually rather large, it may be reduced by visually analyzing PCA or UMAP projections, preferably 3D projections.

Projections are also useful to analyze the relative importance of each feature to determine potential dimension reductions. In the case of SPNs clustering, however, using our methodology we haven't found performance issues related to dimensionality, even when a large number of SPNs is fed into the system.

Calculating a cluster silhouette

Once clusters have been stabilized following the techniques mentioned in the previous section, it becomes possible to build a silhouette for each K-means cluster, which is called a condensate, using the following algorithm:

For each cluster ID,

Initialize the cluster silhouette: ∅

For each SPN in this cluster,

1. Retrieve the SPN silhouette
2. Aggregate it to the cluster silhouette by using the same aggregation formula (consistently take the maximum scopes sw, sa and sr at which a W, A or R permission operates in each role assignment)

The cluster's outer silhouette (as observed from the golden source of RBAC roles managed in Microsoft Entra) is simply the WAR norm calculated over the aggregated individual SPN silhouettes.

From the outer silhouette, it is a straightforward matter to define a desired silhouette by reasoning along each WAR partition axis in sequence, see where we stand, and where we want to go.

For example, imagine the outer silhouette of a cluster is 938:

- Along the W axis, the cluster currently enjoys management group level modification rights (900) Is that legitimate? For a global CD/CI platform, it might make perfect sense, but to the very least all wildcards should be removed (500). For an application bot, de-escalating modification rights to resource group level (300) could be much more sensible.
- Along the A axis, the cluster enjoys subscription level permissions (35). In most situations, this can be lowered to resource group (30) with little efforts, or even in many cases to resource level (20).
- Along the R axis, the cluster enjoys read access to the management group (4). This should be consistent with W needs, so we could end up sticking to 4 if the cluster SPNs are used by a global CD/CI platform, or to 3 for an application bot.

The desired silhouette of this cluster could stand anywhere between 323 (application bot usage) and 534 (global CD/CI platform).