A Multi-Account and Multi-Tenant Policy-Based Access Control (PBAC) Approach for Distributed Systems Augmented with Risk Scores Generation

Nicola Gallo[1] and Antonio Radesca[2]

***Abstract*— This paper introduces a Multi-Account and Multi-Tenant Policy-Based Access Control (PBAC) approach for distributed systems, formalized through the use of relational structures, relational algebra and operations. The approach is further enhanced by generating risk scores. As cloud computing became more prevalent, there was a growing demand for sophisticated software solutions, especially in Software as a Service (SaaS) offerings. These systems effectively handle data across diverse accounts, tenants, computing nodes, and networks. This approach introduces various problems. (1) First and foremost, permissions must be expressed in a straightforward manner to implement both Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC). (2) Secondly, the approach necessitates low-latency and high-availability for the evaluation of permissions. This becomes particularly intricate due to the inherent challenges posed by the PACELC theorem. Furthermore, in a distributed system with asynchronous operations, there is no assurance that the permissions initially allowing the action remain valid at execution time. It is crucial to mitigate this risk and maximize the likelihood of executing with the latest valid permissions at the time of execution. (3) Finally, risk assessment becomes intricate, especially as policies granting permissions grow complex. The approach must be capable of handling the complexity of these policies and effectively assess the associated risks.**

To address these problems, this paper introduces an architecture, relations, algorithms and an approach for generating risk classifications using scores.

I. Introduction

CLOUD computing has changed how companies develop software. Their focus is now on delivering easily scalable software solutions designed to leverage the capabilities of cloud computing. Many new software solutions were containerized and orchestrated with platforms such as Kubernetes. Additionally, an emerging trend that gained popularity was serverless computing.

Cloud native software solutions are fundamentally distributed systems, presenting challenges outlined by the CAP theorem. Software architects must carefully balance Partition Tolerance, Consistency, and Availability. This imperative has led to a shift in architectural styles to accommodate workloads that place less emphasis on strong consistency but prioritize availability through the implementation of Eventual Consistency.

A closer look at the CAP theorem shows that Eventual Consistency aligns with PA (Partition Tolerance and Availability). The CAP theorem [1] asserts that, in the presence of a network partition, a trade-off must be made between Consistency and Availability. Eventual Consistency, as an approach, allows a system to remain available during network partitions, deferring the achievement of full consistency until the partition is resolved. This ensures that the system is Partition Tolerant and Available even though data Consistency may take some time to be fully restored.

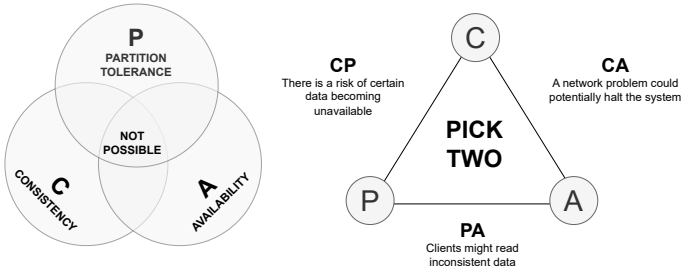

Fig. 1. CAP Theorem

Moreover, there has been a trend towards the adoption of asynchronous patterns utilizing event streaming processing such as Apache Kafka. These platforms are utilized to collect, store and process events representing asynchronous operations that lack a discrete beginning or end.

The Figure 2 illustrates a commonly employed cloud architecture. Clients undergo authentication with a centralized Identity Provider, which subsequently issues a token to the client. The client then uses this token to access the resources of the server. Two widely embraced protocols in this context are OpenID [4] and OAuth [5]. OpenID serves as an open standard for authentication, while OAuth functions as an open standard for authorization.

Within the scope of these protocols, JSON Web Token (JWT) [6] has gained notable significance. JWT offers a concise and URL-safe approach for exchanging claims between two parties. These tokens are encoded and digitally signed, enhancing their integrity and ensuring secure communication between the server and the client. Additionally, this token format streamlines and enhances the efficiency of verification, thereby contributing to the overall security and reliability of the authentication and authorization mechanisms in cloud-based systems.

Among these software solutions, access control stands out as a critical aspect. Models like Role-Based Access Control (RBAC) and Annotation-Based Access Control (ABAC) contribute to the rise of Policy-Based Access Control (PBAC) [7]

[1]Nicola Gallo, Software Architect nicola.gallo at nitroagility.com

[2]Antonio Radesca, Software Architect, antonio.radesca at nitroagility.com

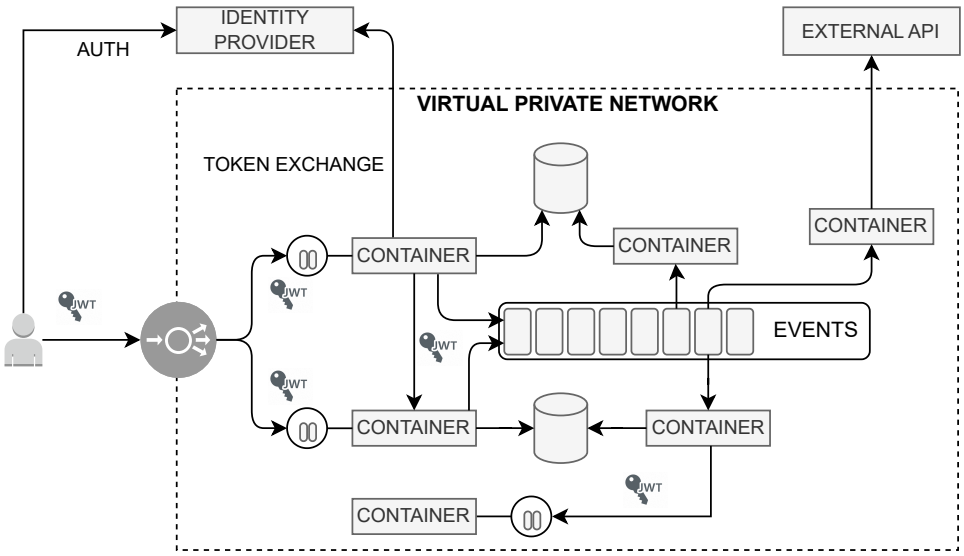

Fig. 2. Cloud Architecture

PBAC, or Policy-Based Access Control, is a method for managing system access control through dynamic policies. Instead of relying on permissions granted by static roles, PBAC employs rules and conditions to determine who can access specific resources. It offers a flexible approach, enabling organizations to customize access control based on specific needs. PBAC is recognized for its adaptability and capability to accommodate complex access requirements in diverse system environments.

PBAC policies must be linked to identities, where identities encompass users, and roles. However, this approach currently lacks standardization and exposes systems to various security threats. Therefore, the scope of this paper is to propose a standardized PBAC approach.

The primary consideration is that the Access Control layer should not be embedded within the software solution; instead, it should be provided as an external Access Control Solution (ACS). The main reason relies on the fact that Access Control is a software aspect that is too complex and error-prone to be implemented in each individual software solution. Ideally, the ACS should support the creation of separate accounts, enabling each organization to manage the access control of its software solutions within one or more isolated accounts as per Figure 3.

The ACS can be seen as a platform that enables organizations to create one or more accounts for modeling Access Control. Additionally, it must support the federation of these accounts to enable cross-account access control management when needed.

Taking a closer look, let's break down the following concepts for the ACS:

- **Account**: An account is the top-level entity in the hierarchy used for managing environment and systems isolation and configurations. It is identified by an AccountId and is associated with a single email address.
- **Identity**: The system can be accessed through identities, taking the form of Users, and Roles. In this context, a Principal is either a human user or a workload with granted permissions, responsible for authentication and initiating requests.
- **Tenant**: A tenant is an entity that accesses specific resources within an account. Ownership of resources can be divided or shared among tenants, and multiple identities can be associated with a tenant.
- **Project**: An account can be partitioned into projects, where each project represents a system.
- **Domain**: A project can be segmented into multiple domains using preferred techniques such as Domain-Driven Design (DDD), with at least one domain necessary for each project.
- **Resource**: Resources in a domain represent domain entities. While they may not directly align with system entities, they indicate logical resources crucial for defining access control policies.
- **Action**: Actions are operations that affect the state of resources.
- **Policy**: A policy is a collection of predefined rules evaluated to determine if a specific identity has been granted the right to perform a certain action.

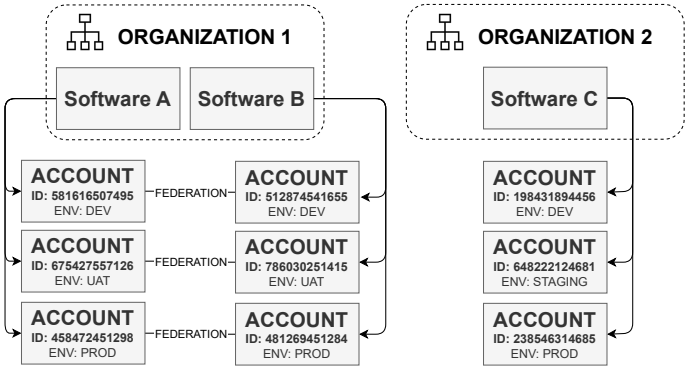

Fig. 3. Accounts

The paper aims to formalize an approach using relational structures, relational algebra, and operations. Additionally, it seeks to implement risk scores generation approach to help mitigate security risks.

This paper is organized as follows: In the next section (II),

the identified and defined problems are discussed. Section III outlines the proposed approach, followed by Section IV, where the generation of risk scores approach is introduced. Finally, Section V discusses the conclusions.

II. Problem Definition

An organization developing a Multi-Tenant software solution, such as a Software as a Service (SaaS), would typically adopt a Microservices Architecture [2] and likely design the model using a methodology such as Domain-Driven Design [3].

Furthermore, the organization must partition the software solution into separate tenants and identities (users and roles). Additionally, it is crucial to assign appropriate permissions to these identities by implementing and associating policies.

It is crucial to note that the Application Development Life Cycle requires distinct and isolated environments for each phase of the development. Therefore, it is imperative to establish multiple environments, such as Development (DEV), User Acceptance Testing (UAT), and Production (PROD).

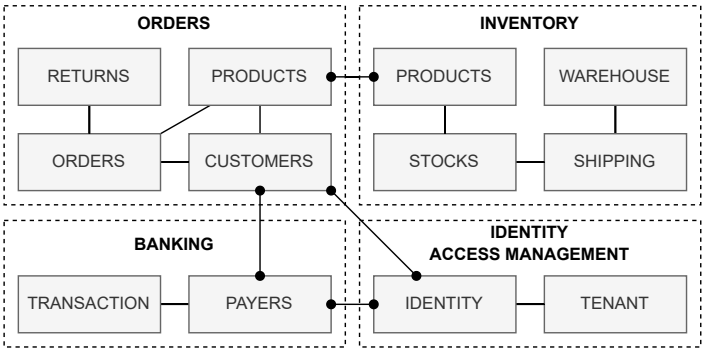

Fig. 4. DDD Bounded Contexts

Figure 4 illustrates an example of bounded contexts using the Domain-Driven Design (DDD) methodology for a hypothetical Software as a Service (SaaS) system that offers Order Management System (OMS) functionalities.

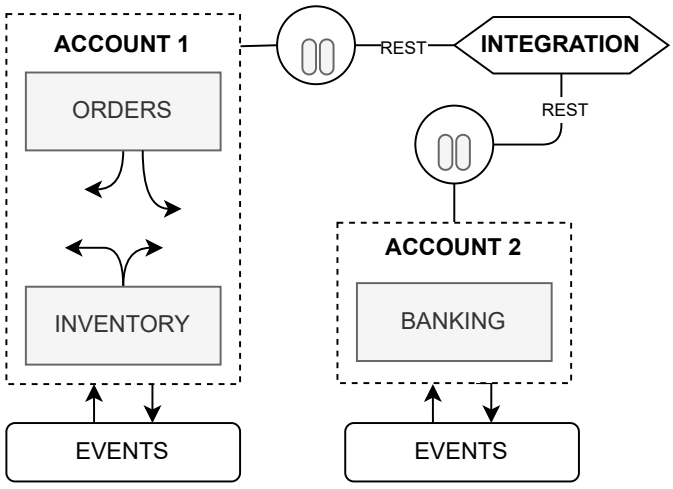

Fig. 5. Microservices Architecture

Meanwhile, Figure 5 depicts a microservices-based architectural solution that employs the proposed Access Control Solution (ACS) approach. This solution divides the software across two distinct yet federated accounts, enabling both isolation and integration.

Let D_{acs} be the database of the Access Control Solution (ACS) with a set or relations REL_{acs} and a set of attributes A_{acs}. REL_{acs} includes $ACCOUNTS$, $TENANTS$, $PERMISSIONS$ etc. A_{acs} includes attributes such as $AccountId$, $AccountName$, $PolicyName$ etc from the relations.

Let E be the set of the environments $E = \{e_1, e_2, \ldots, e_n\}$ and let AC represent the set of the accounts $AC = \{a_1, a_2, \ldots, a_n\}$ within the D_{acs} database.

It is crucial to note that the organization has to assign each account exclusively to a specific environment otherwise it would raise a security risk.

Definition 2.1 (Account Partitioning for Environments): Let AC be the set containing all existing accounts and let E represent the set containing all existing environments, Account Partitioning for Environments is defined as the operation of partitioning the set AC with E using the function f_{env} which is formally defined as:

$$f_{env} : AC \rightarrow AE | AE \subset AC \times E$$
$$f_{env}(a_i) = (a_i, e_m)$$

moreover, AE denotes a subset of $AC \times E$ resulting from the Account Partitioning for Environments operation on the set AC where for each account a_i, the function $f_{env}(a_i)$ generates an unique pair (a_i, e_m). Additionally $\forall a_j \in AC$ there exists a unique $e_n \in E$ such that $(a_j, e_n) \in f_{env}$. In conclusion, we can state the equivalence $AE \equiv f_{env}$.

The set AC_i is a subset of AC with the property $a_m \in AC_i$ and $a_m \notin AC_j$ where $i \neq j$ and so on, such that:

$$AC_i = \{a \mid a \in AC, \forall (a_i, e_m) \in f_{env}, (a_j, e_n) \in f_{env} : (a_i \in AC) \wedge (a_j \in AC) \wedge (e_m = e_n)\}$$

and:

$$\bigcup_{i=1}^{n=|AC|} AC_i = AC$$

and:

$$\forall ACi \in AC, ACj \in AC | (i \neq j \rightarrow ACi \cap ACj = \emptyset)$$

An organization developing the software solution, as illustrated in Figure 5, has to create multiple environments. Assuming three environments as per Table I (DEV, TEST, and PROD), each requiring two accounts (one for $ORDERS$, $INENTORY$ and one for $BANKING$), this results in a total necessity of six accounts as per Table II .

It is important to highlight that the $RELATIONS$ are not normalized in the database just for the sake of simplicity, making the explanation of the proposed solution easier to read.

TABLE I
THE RELATION ENVIRONMENTS

Env
DEV
TEST
PROD

TABLE II
THE RELATION ACCOUNTS

AccountId	AccountName	Env
581616507495	Orders Account	DEV
655814852128	Banking Account	DEV
669871239854	Orders Account	TEST
986167786642	Banking Account	TEST
951435799851	Orders Account	PROD
452917331579	Banking Account	PROD

The sets E and AC can be respectively related to the relations $ENVIRONMENTS$ and $ACCOUNTS$ as follows:

$$E = \pi\{Env\}(ENVIRONMENTS)$$
$$AC = \pi\{AccountId\}(ACCOUNTS)$$

The $ACCOUNTS$ relation has been populated through the Account Partitioning for Environments operation, as defined in Definition 2.1. This implies the existence of three sets, each of which is a subset of AC.

$$AC_{dev} = \{581616507495, 655814852128\}$$
$$AC_{test} = \{669871239854, 986167786642\}$$
$$AC_{prod} = \{951435799851, 452917331579\}$$

For the sake of simplicity, the following tables will display only the PROD account data to avoid confusion.

First step is to configure each account with Projects and Domains, as following:

- **Order Account**: A project named OMS-System and its two domains named Order and Inventory.
- **Banking Account**: A project named Banking-System and its domain named Banking.

TABLE III
THE RELATION PROJECTS

Project	AccountId
oms-system	951435799851
banking-system	452917331579

Each domain then needs to be further partitioned into resources and actions.

TABLE IV
THE RELATION DOMAINS

Domainid	Domain	Project	AccountId
1	orders	oms-system	951435799851
2	inventory	oms-system	951435799851
3	banking	banking-system	452917331579

TABLE V
THE RELATION RESOURCES

Resource	DomainId
product	1
customer	1
order	1
return	1
product	2
stock	2
warehouse	2
shipping	2
transaction	3
payer	3

To complete the configurations, it is necessary to create the $TENANTS$ and $IDENTITIES$ relations as per Table VII.

In summary, we've created accounts and environments, established the necessary relations between them. Following that, for each account, projects, domains, resources, and actions were generated. Lastly, configurations for tenants and identities were completed.

The Access Control Solution (ACS) helps clearly identify resources and actions by using a UUR (Universally Unique Resource) and UUA (Universally Unique Action).

A UUR is expressed as a string with the specific template:,

uur:{account}:{tenant}:{project}:{domain}:{resource}/{resource-filter}

Following is an example of a UUR for the "orders" domain in the project "oms-system" for the "default" tenant, targeting the resource "product" with ID "22":

uur:951435799851:default:oms-system:orders:product/22

A UUA is expressed as a string with the specific template:,

$$resource : action$$

Following is an example of a UUA for the action "get" on the resource "product":

$$product : get$$

Both UUR and UUA accept wildcard characters for matching wildcards.

The final step is a set of policies formed by combining the UUR and the UUA as per Table IX.

TABLE VI
THE RELATION ACTIONS

Action	Resource	DomainId
get	product	1
upsert	product	1
delete	product	1
get	customer	1
upsert	customer	1
delete	customer	1
get	order	1
upsert	order	1
delete	order	1
get	return	1
upsert	return	1
delete	return	1
get	product	2
upsert	product	2
delete	product	2
get	stock	2
upsert	stock	2
delete	stock	2
get	warehouse	2
upsert	warehouse	2
delete	warehouse	2
get	shipping	2
upsert	shipping	2
delete	shipping	2
get	transaction	3
upsert	transaction	3
softdelete	transaction	3
get	payer	3
upsert	payer	3
softdelete	payer	3

TABLE VII
THE RELATION TENANTS

Tenant	AccountId
default	951435799851
tenant1	951435799851
tenant2	951435799851
default	452917331579
tenant1	452917331579
tenant2	452917331579

The implemented software solution presents four distinct types of issues.

A. Policies latency and consistency

Evaluating policies on each computing node can be a resource-intensive operation, especially in scenarios with numerous policies. This process is susceptible to both latency and consistency issues, arising from network latencies between nodes and network partitioning.

B. Hacking of the Roles with Messages

In a distributed system, a common scenario involves publishing a message that is later consumed by a different node as per Figure 6. The node consuming the message should ideally have identical permissions to the one that published it.

However, if the consuming node has excessively broad permissions, it poses a security risk, potentially executing actions that were not intended to be permitted. On the other hand, excessively narrow permissions may prevent the node from executing actions it needs to perform.

TABLE VIII
THE RELATION IDENTITIES

IdentityId	IdentityName	IdentityType	Tenant	Account
1	email-1	USER	default	951435799851
2	email-2	USER	tenant1	951435799851
3	email-2	USER	tenant1	452917331579
4	email-2	USER	tenant2	951435799851
5	email-2	USER	tenant2	452917331579
6	platform-manager	ROLE	default	951435799851
7	platform-manager	ROLE	default	452917331579
8	orders-manager	ROLE	tenant1	951435799851
9	orders-user	ROLE	tenant1	951435799851
10	orders-manager	ROLE	tenant2	951435799851
11	orders-user	ROLE	tenant2	951435799851
12	banking-manager	ROLE	tenant1	452917331579
13	banking-user	ROLE	tenant1	452917331579
14	banking-manager	ROLE	tenant2	452917331579
15	banking-user	ROLE	tenant2	452917331579

These issues need to be addressed both at the message level and the identity level to ensure a balance between security and functionality in the distributed system.

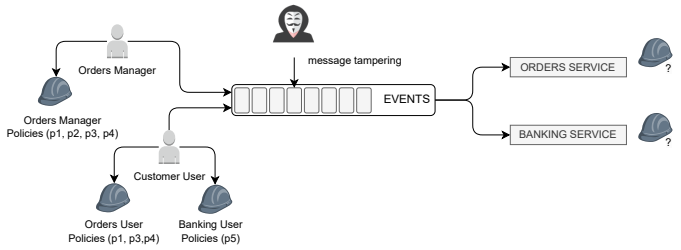

Fig. 6. Hacking of the Roles with Message

C. Hacking of the Roles with Messages using Policies is no longer Allowed

Especially in an Eventual Consistent system, the time it takes to process an action can be significantly delayed from when it was initially requested. This introduces another type of issue where an identity publishes a message that is later consumed by a different identity. However, the original identity that published the message no longer has the permission to execute that action as per Figure 7.

In such cases, the system should either discard the message without processing it or redirect the message to a different queue, necessitating an approval process.

Similar to the previous issue, this challenge can be tackled at both the message level and the identity level to maintain a balance between security and functionality in the system.

D. Risk of complex policies evaluation

Evaluating complex policies at runtime raises security concerns because unforeseen situations may lack protection, leading to a potential risk of an attack. Therefore, having a system in place to prevent such attacks is crucial, and one effective approach is to assess the system's security by generating risk scores.

TABLE IX
THE RELATION POLICIES

PolicyId	Effect	UUR	UUA
p1	allow	uur:951435799851:tenant1:oms-system:orders:product/*	product:get
p2	allow	uur:951435799851:tenant1:oms-system:orders:order/*	product:create
p3	allow	uur:951435799851:tenant1:oms-system:orders:product/*	order:get
p4	allow	uur:951435799851:tenant1:oms-system:orders:product/*	order:create
p5	allow	uur:452917331579:tenant1:banking-system:transation:*	transation:get
p6	allow	uur:452917331579:tenant1:banking-system:transation:*	transation:create
p7	deny	uur:452917331579:tenant1:banking-system:transation:*	transation:delete

TABLE X
THE RELATION IDENTITIESPOLICIES

IdentityId	PolicyId
2	p1
2	p3
2	p5
8	p1
8	p2
8	p3
8	p4
9	p1
9	p3
13	p5

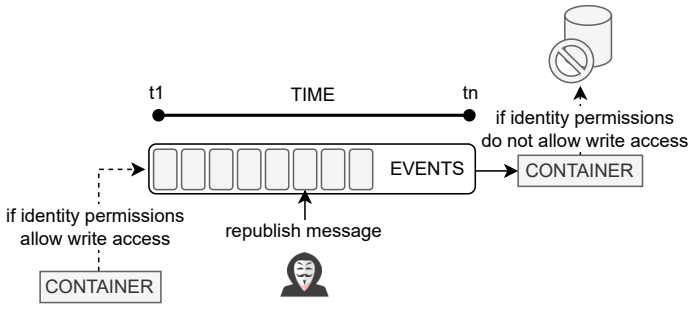

Fig. 7. Hacking of the Roles with Messages using Policies are no longer Allowed

III. POLICY-BASED APPROACH

IN this section the paper is going to solve three of the four main problems:

- Policies latency and consistency
- Hacking of the Roles with Message Tempering
- Hacking of the Roles with Messages using Policies is no longer Allowed.

A. Policies latency and consistency

The evaluation of a policy can be accomplished using an evaluation function, denoted as f_{eval}.

This function takes as input an identity, a resource, and an action. Subsequently, it evaluates whether the specified action can be performed on the given resource.

Definition 3.1 (UUR Matching Algorithm - $f_{uurmatch}$): The function $f_{uurmatch}$ is designed to take two Uniform Resource Identifiers (UURs) as input and returns a Boolean value.
Specifically, it returns True if the first UUR, denoted as $uur1$, is matched within the second UUR, denoted as $uur2$.
An UUR is considered to be matched if the two are identical or if $uur2$ has wildcards that permit the match of $uur1$.

Definition 3.2 (UUA Matching Algorithm - $f_{uuamatch}$): The function $f_{uuamatch}$ is designed to take two Uniform Action Identifiers (UUAs) as input and returns a Boolean value.
Specifically, it returns True if the first UUA, denoted as $uua1$, is matched within the second UUA, denoted as $uua2$.
An UUA is considered to be matched if the two are identical or if $uua2$ has wildcards that permit the match of $uua1$.

Definition 3.3 (Policy Evaluation Algorithm - f_{eval}):
Input:

- iid: identifier of the identity to be evaluated
- iurr: resource to be evaluated
- iaction: action to be evaluated
- P: set of the policies
- PI: set of the policies associated to the identities

Output: true if the action cen be performed on the input resource, false otherwise.
BEGIN:
$DP = \sigma_{IdentityId=iid,Effect='deny'}(PI \bowtie P)$
$DPA = \pi_{UUA}(\sigma_{f_{uurmatch}(iurr,UUR)}(DP))$
$\forall action_{DP} \in DPA$,
if $f_{uuamatch}(iaction, action_{DP}) is false$
return false

$AP = \sigma_{IdentityId=iid,Effect='allow'}(PI \bowtie P)$
$APA = \pi_{UUA}(\sigma_{f_{uurmatch}(iurr,UUR)}(AP))$
$\forall action_{AP} \in APA$,
if $f_{uuamatch}(iaction, action_{AP}) istrue$
return true

return false

A software solution necessitates invoking the f_{eval} function for each required policy enforcement. In the scenario where the solution must evaluate whether an action can be performed on a resource n times, the function has a time complexity $O(n)$.

This level of complexity would be acceptable only under the condition that the execution time for each function call is very short, in the range of a few milliseconds. However, each evaluation point would demand a call to a remote server to evaluate the policy, introducing significant latency to the overall solution and rendering the system unavailable in the event of network partitioning.

The time-consuming operations in the implementation of the Policy Evaluation Algorithm 3.3 are those executing queries, such as selections and projections. The only way to reduce this execution time and the risk of network partitioning is to minimize the number of remote queries and execute them locally.

This can be achieved by synchronizing the policies into the node and loading all of them into memory. While this approach would resolve the latency and network partitioning issues, it introduces the trade-off of eventually consistent data, and the node might experience periods where policies are out of synchronization as shown in Figure 8.

B. Hacking of the Roles with Messages

A software solution implementing asynchronous operations backed by messages would need to assess the permissions of the identity for the requested action on a resource. Upon successful evaluation, it would publish a message. On the receiving end, a consumer would read the message and execute the action.

Let's consider a scenario where the identity $email-2$ has been granted permissions by the policies $p1, p3, p4, p5$ and the

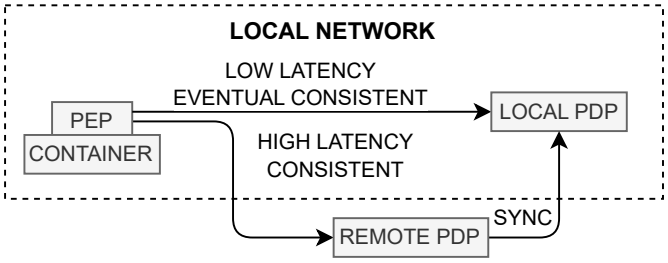

Fig. 8. Latency and Consistency for Policies Enforcement

identity requests to create an order, resulting in the publication of a message with the request to create an order for the identity $email-2$ which is granted by the policy $p4$.

The consumer node would need to perform the operation on behalf of the identity; therefore, the functional account used to execute the consumer would require the permission $p4$.

While it's possible to assign the permissions granted by the policies $p1, p3, p4, p5$ to the functional account, doing so would expose the system to multiple security risks:

- **Separation of Concern**: A consumer implementing OMS-System features should not implement Banking-System features.
- **Least Privilege Principle**: The consumer should adhere to the principle of least privilege, which is a concept that restricts a user's access rights to only what is strictly necessary.

It is correct to consider assigning only the permissions granted by the $p4$ policy to the functional account. However, a challenge arises as this consumer needs to handle all types of requests for the OMS-System.

As a result, granting the service account all the permissions from all OMS-System policies would be necessary. However, this would expose the system to security risks and would not adhere to the principle of least privilege.

Let's define $M1$ as the set of policies required to execute the action in the message m_1, denoted as $M1 = \{p_i, p_j\}$. Similarly, let's define $M2$ as the set of policies required to execute the action in the message m_2, denoted as $M2 = \{p_m, p_n\}$. The security risks would be defined as following:

- When executing m_1, security risks would be introduced by the policies that are not required to execute the operation, specifically $M2 - (M1 \cap M2)$
- When executing m_2, security risks would be introduced by the policies that are not required to execute the operation, specifically $M1 - (M1 \cap M2)$

This leads to the concept of a runtime policy evaluation context. This implies that when serving a request, the service account should be running in a runtime policy evaluation context that includes only the strictly required policies. In the previous usage example, when processing the message m_1, the runtime policy evaluation context should include the policies in $M1$, while

when processing m_2, the context should include the policies in $M2$.

This can be achieved by requiring the system to specify, in each message, the requesting identity and the role required to execute the message. On the other hand, the consumer would need to be granted permissions to impersonate the identity specified in the message by elevating its own identity to the specified role.

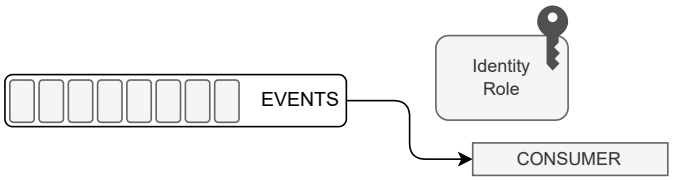

Fig. 9. Message processing

To prevent falling victim to a message tampering attack, the ACS solution requires signing messages and mandates that consumers verify the validity of each message.

Definition 3.4 (Message processing by Role - $f_{process}$):
Input:
- msg: message to be processed

BEGIN:
// Validate the signature of the message
$if f_{validate}(msg) is false$
return
// Elevate the identity to be executed as the role in the message
$if f_{elevate-role}(msg) is false$
return
// Process the message
$f_{business-handler}(msg)$

C. Hacking of the Roles with Messages using Policies is no longer Allowed

A software solution would need to evaluate the actions on resources for each identity request and subsequently execute the action on the system. However, during the interval between the evaluation of the action and its execution, the permissions associated with the identity might have changed. This becomes more pronounced in distributed systems utilizing asynchronous operations supported by messages.

The Message Processing by Role Algorithm 3.4 enables the handling of this type of issue as well. By requiring a consumer to elevate its identity to a specific role, it ensures that at the time of execution, the latest policies are applied to the role. Therefore, if there are changes since the point in time when the identity made the request, these changes would be reflected in the execution of the asynchronous operation.

IV. Risk Scores Generation

The Policy Evaluation Algorithm 3.3 is applied to a specific identity, and the final evaluation is the process to assess multiple policies at runtime.

This approach is highly flexible as it allows writing, composing, and evaluating policies. However, it poses a risk, as introducing a new policy could potentially expose the system to high risks. Therefore, it is crucial to define an approach that would calculate a risk score for each of the existing identities.

As depicted in Figure 10, a policy can be described as a triple, consisting of the following components:

- **Effect**: The effect of the policy can be either $ALLOW$ or $DENY$
- **Resource**: The resource target of the action to be performed
- **Action**: The action to be performed.

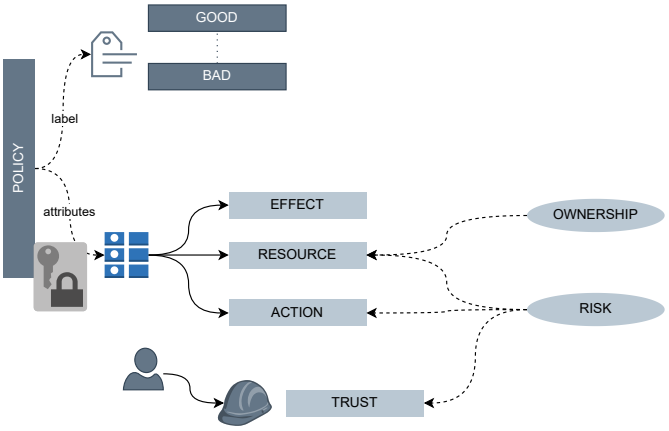

Fig. 10. Policy Risk Score Structure

Policies in the context of identity association can be further augmented with the following concepts:

- **Ownership**: A resource incorporates the concept of ownership, which in a multi-tenant context can be detailed as:
  - **Platform**: The owner of the software solution
  - **Tenant**: A tenant configured within the platform
  - **User**: A user registered within the platform.
  - **External**: An external resource, representing a third-party system that is integrated.
- **Trust**: This reflects the level of trust that the platform owner places in the specific type of identity
- **Risk**: Risk is a value ranging from 0 to 1, indicating the level of risk associated with a resource, action, or trust.

To assign a risk score to each identity, the proposed solution leverages a Machine Learning approach that assigns a score to a sample of policies and subsequently constructs a machine learning model. Establishing a procedure to create the dataset for building the machine learning model is essential.

Initially, the collection of all policies associated with the identity under assessment is required. For each policy, the creation of a Policy Risk Score Input rapresentation is mandatory, defined as follows:

- **Effect**: The effect of the policy can be either $ALLOW$ or $DENY$
- **Resource Ownership**: Owner of the resource. The domain is a set that includes the following values: $PLATFORM$, $TENANT$, $USER$, $EXTERNAL$

TABLE XI
POLICY RISK SCORE INPUT

Effect	Res. Ownership	Res. Legal Risk	Res. Revenue Risk	Res. Reputational Risk	Action Risk	Action Operation	Identity Trust Risk
allow	TENANT	1	0.8	0.7	0.8	DELETE	0.3
deny	USER	0.4	0.3	0.4	0.7	PUT	0.5
allow	TENANT	1	0.9	0.8	0.2	GET	0.4
allow	PLATFORM	1	0.9	0.8	0.2	POST	0.4

- **Resource Legal Risk**: Risk based on legal repercussions in the event of a security breach. The domain is $x \in \mathbb{R},\ 0 \leq x \leq 1$
- **Resource Revenue Risk**: Risk based on revenue repercussions in the event of a security breach. The domain is $x \in \mathbb{R},\ 0 \leq x \leq 1$
- **Resource Reputational Risk**: Risk based on reputational repercussions in the event of a security breach. The domain is $x \in \mathbb{R},\ 0 \leq x \leq 1$
- **Action Risk**: Risk based on how the action can alter the system. The domain is $x \in \mathbb{R},\ 0 \leq x \leq 1$
- **Action Operation**: Type of operation. The domain is a set that contains the following value: GET, $POST$, PUT, $PATCH$, $DELETE$, $SOFTDELETE$
- **Identity Trust Risk**: Risks based on the trust in the type of identity. The domain is $x \in \mathbb{R},\ 0 \leq x \leq 1$.

The values used to form the Policy Risk Score Input are assigned to each resource and action during the system configuration by developers. Subsequently, these values are employed to construct the Policy Risk Score Input.

Let's define $PRISK$ as the set containing a Policy Risk Score Input representation for each policy assigned to an identity. Each item belonging to $PRISK$ is associated with a score that can range between values classified as either $GOOD$ or BAD. The domain is $x \in \mathbb{R},\ 0 \leq x \leq 1$.

A dataset is constructed by selecting n identities and creating for each of them a $PRISK$ set associated with a risk score. Once the dataset has been built, it is necessary to construct a machine learning model that learns from the dataset and predicts the risk of a given identity.

The focus of this paper is to build the dataset; however, constructing the machine learning model is beyond the scope of this document. Ideally, it would be necessary to create multiple machine learning models and compare them to find the one that fits best.

This approach can be further extended through the use of ontologies. Developers configure resources and actions with values utilized to create the Policy Risk Score Input representation and then building the Machine Learning model.

When tackling a new business domain, it would be possible to create an ontology that maps the new domain and establishes links with the ontology of the initial configured domain. This would facilitate the inference of values for the resources and actions of the new business domain, and, more importantly, it would allow the utilization of the Machine Learning model to assess the new domain.

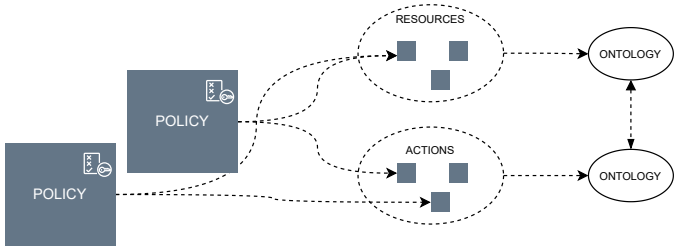

Fig. 11. Ontology Mapping

.

V. CONCLUSION

THIS paper introduces a novel approach to Multi-Account and Multi-Tenant Policy-Based Access Control (PBAC) in distributed systems and presents a Risk Scores Generation methodology. The proposed solution addresses various security challenges inherent in systems utilizing identities and runtime-evaluated policies, particularly in distributed environments, especially when adopting eventual consistency.

By combining Multi-Account and Multi-Tenant PBAC with the Risk Scores Generation approach, the presented solution offers a comprehensive strategy for tackling security concerns. It not only enhances the robustness of systems relying on identities and runtime policies but also provides a valuable framework for addressing security issues in distributed systems, particularly those navigating the complexities of eventual consistency.

This research contributes to the ongoing discourse on secure access control mechanisms in distributed systems and lays the foundation for further exploration and refinement of these innovative approaches.

REFERENCES

[1] Brewer, Eric. (2000). Towards robust distributed systems. PODC. 7. 10.1145/343477.343502.
[2] V. Velepucha and P. Flores, "A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges," in IEEE Access, vol. 11, pp. 88339-88358, 2023, doi: 10.1109/ACCESS.2023.3305687.
[3] V. Vernon, Domain-Driven Design Distilled. Reading, MA, USA: Addison-Wesley Professional, 2016.
[4] [OpenID.2.0] OpenID Foundation, "OpenID Authentication 2.0," December 2007.
[5] [RFC6749] M. Jones, D. Hardt, "The OAuth 2.0 Authorization Framework," October 2012.
[6] [RFC6750] M. Jones, D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", October 2012
[7] L. Zhi, W. Jing, C. Xiao-su and J. Lian-xing, "Research on Policy-based Access Control Model," 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing, Wuhan, China, 2009, pp. 164-167, doi: 10.1109/NSWCTC.2009.313.