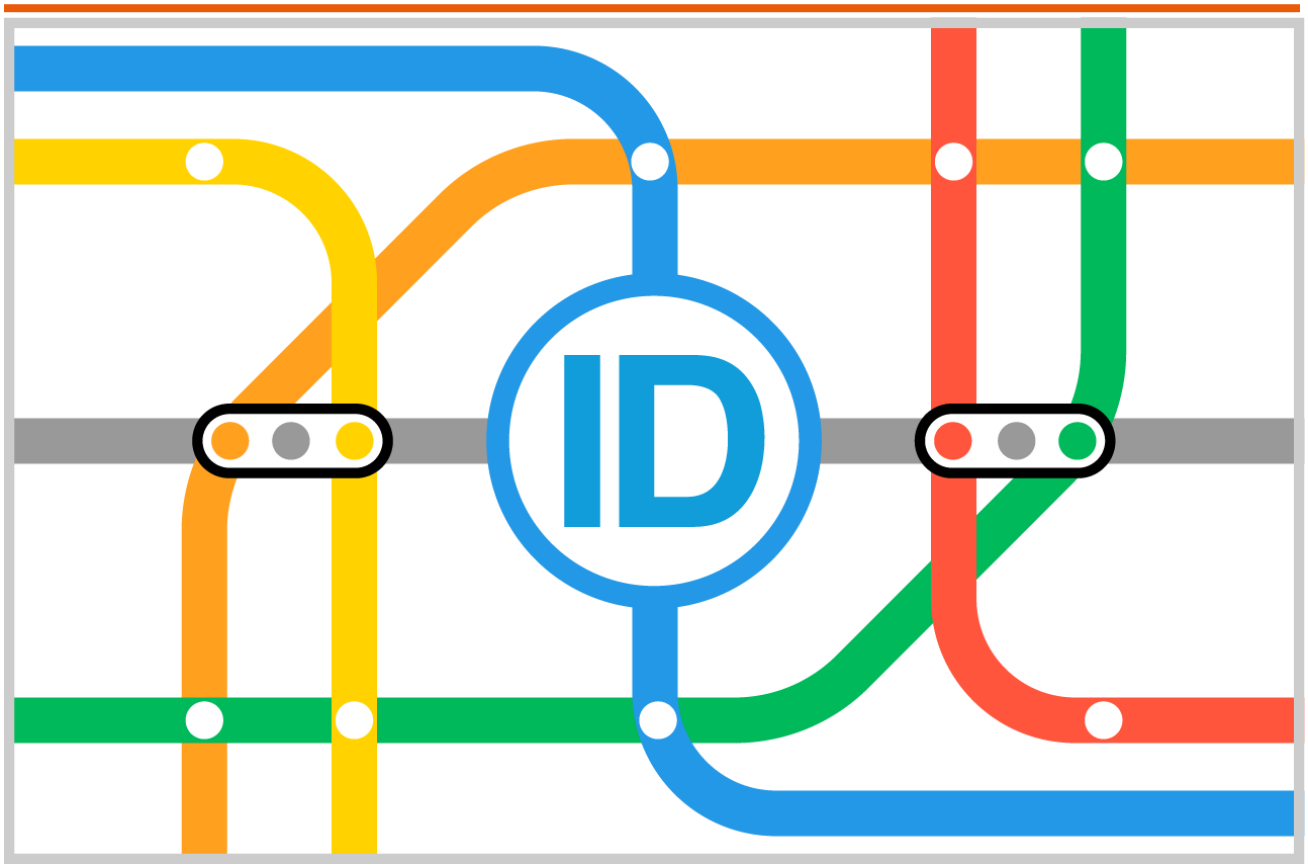
WHITEPAPER | FEBRUARY 2018

Best Practices in Enterprise Authorization V3:

The Next Generation Access Control (NGAC) Approach



empowerID The All-in-One Identity Management Platform

CONTENTS

OVERVIEW

The modern IT organization is a complex mesh of internally on-premise and externally hosted Cloud applications. A central concern that this situation creates is how best to secure access and control authorization for these applications. A long-standing debate in the IT community has been whether *Role-Based Access Control, or RBAC ("are-back") for short*—granting access to roles—or *Attribute-Based Access Control (ABAC)*—giving access via attribute-based rules—is a better model for authorization management. In addition to these two models, a new entrant dubbed Next Generation Access Control (NGAC) has emerged as a standard (ANSI/INCITS 526 2016 Edition, February 17, 2016).

The one thing that everyone can agree on is that, whatever the model, authorization logic should be created and maintained external to the application and not managed uniquely within each application. The reality of the IT application landscape is that many systems do not support centralized authorization control from either model. A hybrid model would leverage the best of each approach for defining policies, offer real-time decision making, and enforce permissions down onto legacy systems when required.

In this white paper, we discuss the RBAC versus ABAC models for authorization, lay out the benefits and weaknesses of each approach and offer a real-time RBAC/ABAC hybrid model following the new NGAC standard that retains the advantages of each while avoiding their major weaknesses.

INTRODUCTION TO RBAC AND ABAC

Role-Based Access Control or "RBAC" is a security and authorization model for securing access to computer resources. Almost all enterprises use RBAC to secure their systems. Sometimes referred to as "Role-Based Security," RBAC access is based on roles as defined by the business using them. In the RBAC model, roles are created and then sets of permissions for resources are assigned to the role. Users are then granted one or more roles to receive access to resources.

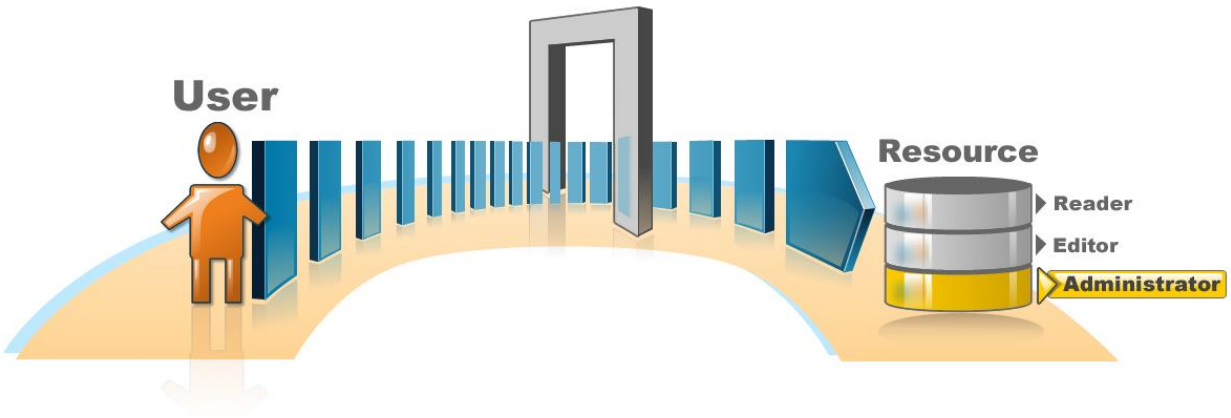


Figure 1. Role-Based access to permissions for resources

ABAC, on the other hand, stands for Attribute-Based Access Control. As suggested by the name, ABAC relies on user attributes for authorization decisions. ABAC policies are rules that evaluate access based upon four sets of attributes. These include: *Subject Attributes*, which are attributes concerning the person or actor being evaluated; *Resource Attributes*, which are attributes of the target or object being affected; *Action Attributes*, which describe the action to be performed on the *Resource*; and, *Environment Attributes,* which include attributes such as the time of the day, IP subnet, and others that do not relate to either the *Subject* or the *Resource*.

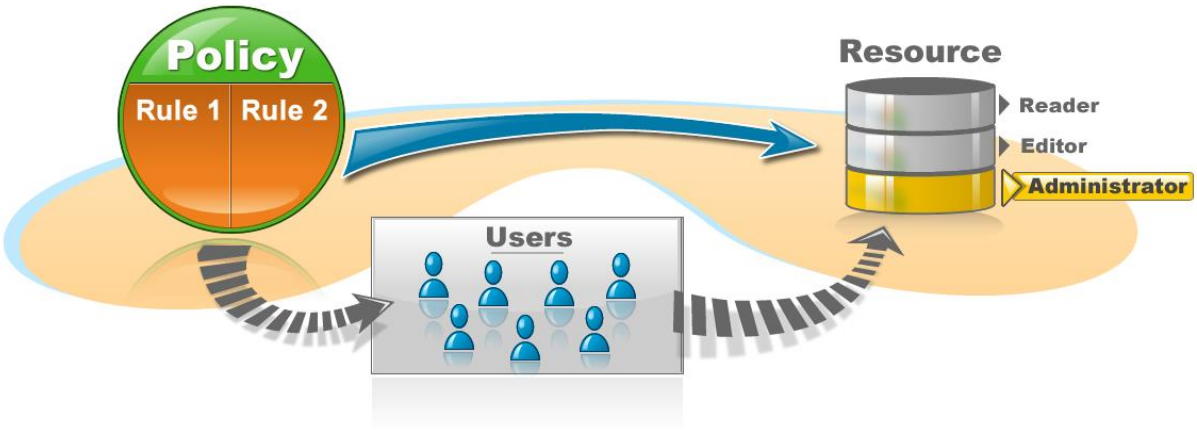


Figure 2. Simple logical view of attribute-based access to permissions for resources

RBAC VERSUS ABAC

Each of these models has its weaknesses and benefits. RBAC trades-off the initial effort of structuring roles for advantages in administration and auditability, whereas ABAC reverses those, allowing the immediate creation of rule policies, but complicating the management and audit of user permissions.

BENEFITS OF RBAC

In the RBAC model, the assumption is that controlling and maintaining access is easier since access is not directly assigned to users but bundled into assignments made to roles. Roles decrease the cost of security management and compliance auditing as they centralize access management into fewer assignments to be managed and audited. In an RBAC model, it is evident who has a role and what access the role grants. Furthermore, per a 2010 NIST study, RBAC implementation delivers significant ROI to companies through efficient provisioning and reduced employee downtime

RBAC'S Primary Benefits:

1. RBAC is deterministic. An RBAC approach makes it easy to know who has access to what at any moment in time.
2. RBAC is more direct and easier to visualize. Security admins can visualize the actors and resources they will affect when creating or modifying a policy.
3. RBAC is inherently auditable. With RBAC assignments it is simple for business owners to certify or attest to access granted, as the consequences of that access are visible. This visibility contrasts with ABAC where a "before the fact audit" is not possible and the effects of a rule are not easy to grasp.
4. RBAC can be simpler than ABAC. For example, with RBAC, bundles of access can be directly assigned to a user. To do this in ABAC requires the creation of a new rule.

WEAKNESSES OF RBAC

The precise nature of the RBAC model can also be considered the source of many of its weaknesses as it is more static than ABAC and doesn't consider the contextual information when making an access decision. When managing access with RBAC, each new situation must be considered against all existing roles to determine how best to deliver the access. Should current roles be modified or new roles be created? Handling all the access requirements encountered in a typical enterprise can lead to the creation of an excessive number of roles (aka "role bloat" or "role explosion.")

RBAC'S Primary Weaknesses

1. RBAC requires advance knowledge of the Subjects and Resources and typically does not support making on-the-fly contextual decisions.
2. An RBAC-only approach can result in a huge number of roles to accomplish fine-grained authorization.
3. Resource Owners must know something about the roles and their intended purpose to grant access to those roles accurately.
4. Resources must be organized into collections to facilitate delegation.
5. Given a substantial number of roles and collections of resources, a correspondingly large number of delegations would need to be created and managed.

Poorly Implemented RBAC

RBAC's greatest weakness is that it is almost universally oversimplified in its implementation by application developers. The RBAC model defines the concept of a Subject, a Role, and a Permission much the same as ABAC's Subject, Resource, and Action. In the RBAC model, a Permission is the combination of an "Action" for a "Resource" in ABAC terminology (e.g., CustomerA.delete). However, most implementations of RBAC, as written by application developers, limit themselves to simple checks for role membership without any concept of permission. This usage downgrades roles into being simply groups to maintain collections of users with the central role management system unaware of the permissions they grant in the applications themselves.

The pseudocode below is an example of this simplistic "IsInRole" approach:

```
if (user.IsInRole(StandardEmployee)
   this.application.show(Employee.PublicProfile)

else if (user.IsInRole("HRManager") || (user.IsInRole("SalesManager") ||
(user.IsInRole("AuditManager")
   this.application.show(Employee.PublicProfile + Employee.AbsenceReport)
```

This oversimplification is so widespread that it has become synonymous with RBAC and even promoted as a standard practice by many vendors. In this model, application permissions are enforced solely internal to the application and hidden from the role management system. This internal enforcement makes it impossible to manage or audit the actual access being granted by

roles, opening the possibility for security vulnerabilities and abuse. Also, as we can see in the pseudocode example above, any time the rules for who may perform an action change, a new role needs to be created and added to the hardcoded list of roles checked by the application's IsInRole statements. Also, when a role's permissions become more restrictive, all application code referencing them would need to be removed and recompiled. With each new change to policy, this process would be repeated. The IT team would need to revisit the application, investing significant resources each time the smallest change occurs. Even more importantly, writing role checks into source code lacks the transparency necessary for maintaining a useful audit trail. As there is no clear external link between the Absence Reports resource shown in our example code and the roles granting access to it, assignments like this cannot be audited. Over the passage of time, exactly what resources each role can access becomes lost to everyone.

BENEFITS OF ABAC

The key advantage of ABAC is that it does not allow application developers to oversimplify their authorization source code by hardcoding a static list of roles. ABAC forces them to centralize all authorization decisions and call out at runtime for a decision based on the *Subject*, *Resource*, *Action*, and *Environment* request attributes. Another key benefit is ABAC's sometimes simpler nature, which can make it easier to understand how a rule grants access to a resource. RBAC seems foreign to many users, and the levels of abstraction can be difficult for an IT team, especially during the early phase of its adoption.

Other main advantages of ABAC include flexibility—almost anything can be represented as a rule-based query as long as the necessary data is available. A rule evaluated at runtime in a login session can make use of contextual information, even information passed in via SAML claims. In contrast, a standard RBAC engine does not evaluate this type of information when delivering the role membership for a user to the application.

ABAC Primary Benefits

1. ABAC enforces centralized management of authorization policies.
2. ABAC makes it easy to specify access rules as simple queries.
3. ABAC rules can be extremely fine-grained and contextual.
4. ABAC rules can evaluate attributes of Subjects and Resources that are not inventoried by the authorization system.
5. ABAC rules need less maintenance and overhead because they do not require the creation or maintenance of the structure on which an RBAC model depends (e.g., roles and resource locations.)

WEAKNESSES OF ABAC

One of the biggest challenges with ABAC is that the just-in-time evaluations of its rules often results in a disconnect or lack of an auditable link between the rule-based policies and the resources (aka assets) they protect. ABAC policies are inherently non-relational, with access rules defined in text-based policies. This model contrasts significantly with the relational nature of RBAC in which users are related to roles, which in turn relate to permissions for resources. This lack of a relational concept is at the core of many of ABAC's weaknesses. XACML policies in ABAC are also less efficient as multiple policies must be taken into consideration to render a decision as policies are not computed and then combined into a final decision.

A simple example using a rule that grants access to an application based on attribute values and group membership highlights some of the shortcomings of ABAC. In this example, the rule authorizes access only to members of "Group C" who have a Job Title that contains "sales" in the HR system **and** has the appropriate employee safety training status in the corporate training system.

This example illustrates the flexibility of ABAC where can be created rules with almost any syntax without requiring the complex relationships needed for similar RBAC assignments. However, this rule may not perform acceptably in real-time taking longer than would be acceptable for the application to receive a decision. The challenge lies in how the protected application is completely disconnected from the rule and from the conditions within the rule that specify group membership and attribute values. Moreover, as the following shows, there is also a disconnect between the group used in the rule and the application to which it grants access.

Standard audit questions arising from this scenario that would be extremely difficult to answer, if at all include:

1. As an application owner, how would I see which users have access to my application and how they each were granted access? How will application owners be trained to read and interpret the rules as well as how to research all the source information? How would the application owner determine who matches the rule, given that the origin of the attribute values could come from another system or be contextual to a single login session?
2. How would the application owner grant direct access to a user in an ABAC model?
3. As a delegated admin who can add or remove users from Group C, how would I know that adding users grants them access to Application A? With ABAC, no relationship exists between the group and the access it gives as there would be with an RBAC model.
4. As an auditor, how would I audit how access is granted to the application and who is giving the access? Was the person who added the user to Group C the same person who gave the access to Application A? Or was it the training staff member who changed the user's status to "Authorized"?
5. What security weight should be associated with the Job Title and training status fields on a user? From how many sources could this information be edited and by whom? Potentially many people could be unintentionally performing "Entitlement Management" on a day-to-day basis.

6. How does an auditor monitor the transitions when users are granted and revoked access to sensitive applications dynamically? No log would be able to track persons who had access on a Monday versus a Friday because access would only be evaluated at runtime.

ABAC's Primary Weaknesses

1. ABAC makes it extremely difficult, if not impossible, to perform a "before the fact audit" and determine the permissions available to a specific user. Potentially, a huge number of rules might need to be executed, and in the same order in which the system applies them, to successfully determine access. As a result, it could be impossible to determine risk exposure for any given employee position.
2. ABAC can lead to a "Rule Explosion" (somewhat in the same way as RBAC can create a "Role Explosion) as a system with N number of attributes would have 2^N possible rule combinations.
3. ABAC systems (which don't pre-calculate the net result of access rights) can be unacceptably slow to answer authorization queries unless rules are kept extremely simple and do not access data from multiple source systems.

THE "LAST MILE" PROBLEM OF BOTH RBAC AND ABAC

The Last Mile problem is a colloquial phrase originating in the telecommunications industry to describe how the final delivery of their service to the end consumer proves proportionally more challenging and expensive. RBAC and ABAC systems both encounter this problem as only a small number of existing applications or systems support using an external source for authorization decisions. An example of this can be seen by examining Office 365 mailboxes.

Office 365 mailboxes maintain ACL-based permissions, controlling who may read the emails contained within a mailbox, who may send as the owner of the mailbox, and who may access and manage the owner's calendar for scheduling purposes. There is no practical way in which to insert an external authorization system into this equation and have the email system call out for real-time centralized decision-making for mailbox access regardless of whether the authorization system relies on RBAC or ABAC.

One might conclude that the problem outlined above would be limited to legacy ACL-based permissions systems such as mailboxes and file shares. It is not, however, as can be seen when we examine the much more modern and sophisticated authorization models employed within Amazon AWS. The Amazon AWS authorization model relies on extremely granular and flexible ABAC-like policies stored as JSON documents. Security admins can precisely define which operations or actions can be performed for which AWS services and the specific resources within these services. The policies include the flexibility of the ABAC model but also contain some of its inherent lack of auditability. Again, as with our mailbox example, the AWS authorization system does not "call out" for decision making from a centralized ABAC or RBAC-based system.

So, given these inherent limitations in much of the IT application landscape, how do ABAC and RBAC vendors maintain relevancy for the permissions they define and manage centrally? In many cases, there is nothing they can do to enable these systems to perform external real-time authorization. There are, however, a few commonly employed methods to allow centrally managed authorization policies to be enforced in these types of systems.

Reverse Proxy Web Access Management (ABAC or RBAC)

Web Access Management (WAM) is a web-centric Identity Management strategy that focuses on enforcing authorization policies for web applications by controlling access to the URLs or pages within the application. WAM systems control this access by either placing an agent on the web servers or by placing a Reverse Proxy server in front of the web servers. In both cases, all calls to the application are intercepted, and the authorization system determines the decision to allow access. This model works for both ABAC and RBAC systems, but only provides coarse access control to the URLs themselves and not to the data or actions being performed within the application.

Claims-Based (ABAC or RBAC)

Many modern web applications support centralized authentication whereby the user signs into a trusted identity provider, which then delivers the identity of the user and information about them as “claims” to the application. The application trusts this information and uses it to verify the user’s identity, leveraging the claim information to control access within the application itself.

A sophisticated example of this can be seen in the AWS authorization model. AWS supports receiving AWS IAM “Roles” as claims. These roles are in turn mapped within AWS to the authorization policies we discussed above.

This mapping allows a centralized system that is tied into the federated authentication process to “control” the authorizations within AWS. Again, AWS does not call out to the authorization system for real-time decisions, and the control is not granular in that the central system. It merely sees these roles as group memberships to pass along and is unaware of the access they provide. The claims-based authorization model can be supported by either an ABAC or RBAC system.

Enforcement Using Provisioning Engine (RBAC Only)

A final option for controlling application permissions based on centrally managed policies is to “push” or “enforce” them down onto the system using a provisioning engine. This involves management of access policies centrally using RBAC, since contextual on the fly decisions are not possible in this case, and then pushing down the resultant “calculated” access into the external systems. In our Office 365 mailbox example, this would involve an RBAC-based system calculating all the role assignments or policies granting mailbox permissions for each mailbox and then triggering a provisioning system to grant or revoke the permissions via PowerShell with Office 365. This model relies on a unique type of RBAC within a Big Data-like engine that can continuously calculate who has which permissions to what external system resources, even through policies that use inheritance and query-based rules.

HYBRID RBAC AND ABAC APPROACHES

As we can see, we live in an imperfect world where non-traditional methods are often required to manage permissions in IT systems. RBAC and ABAC are two ways of managing authorization policies. Moreover, while both have overlapping qualities, individually each one cannot cover all the necessary aspects of access control. So why choose between the two?

For optimal, dynamic support of an IT organization's needs, systems supporting the richness of the RBAC relational modeling system with the flexibility and contextual nature of ABAC offer the best solution. Also, WAM and a provisioning engine are required to enforce the results in these policies in many IT systems. The debate over which is the better access control system is not-applicable to *EmpowerID*'s Identity and Access Management system because *EmpowerID* incorporates these aspects while solving many of the weaknesses found in each approach.

Types of RBAC/ABAC Hybrid Models and their strengths and weaknesses:

Attribute-Centric

In an Attribute-Centric hybrid model, Roles are treated as just another attribute, with permission sets assigned to attribute-based policies. In contrast with conventional RBAC, a role is not a collection of permissions but rather another attribute named "role." This model is the same as the "IsInRole" approach whose significant weaknesses we have previously discussed. The main drawback to the Attribute-Centric model is the loss of RBAC's administrative ease when determining risk exposure for any employee position due to the lack of the relationship between the role and the access it grants when treated as another attribute.

Role-Centric

In a Role-Centric hybrid model, permission sets are assigned to Roles and attribute-based policies are added to constrain RBAC. Constraint rules incorporating attributes can only reduce the permissions available to users, not expand them. Additionally, some of ABAC's flexible qualities are lost because roles still constrain permission sets. The system, however, retains RBAC's qualities for determining the maximum set of permissions that are user-attainable.

Dynamic Roles

In a Dynamic-Roles model, permission sets are assigned to Roles just as they are in the Role-Centric hybrid model. The Roles themselves, however, are assigned to users dynamically by attribute-based policies. This model supports the use of complex attribute-based permissions assignments that automatically assign users to roles.

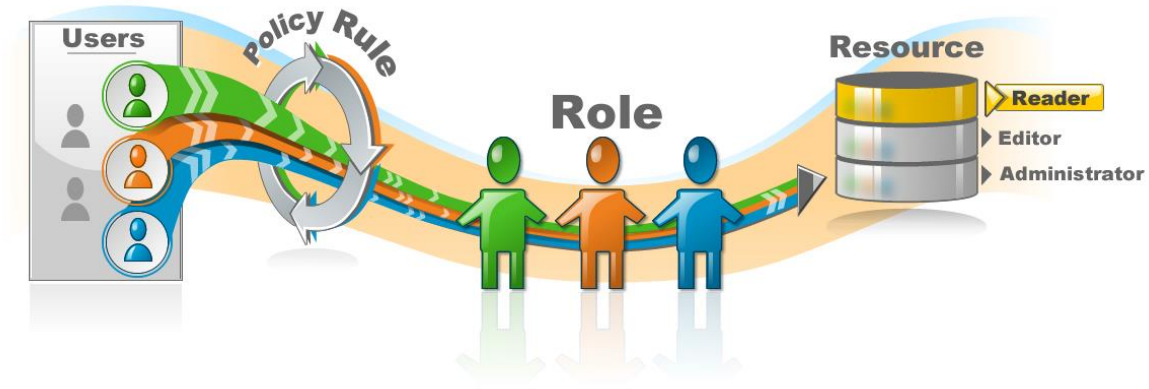


Figure 3. Dynamic roles where attribute-based policies automate role assignment

THE EMPOWERID NEXT GENERATION ACCESS CONTROL (NGAC) MODEL

Over the last ten years, EmpowerID has employed a unique model that maintains the best of each of the types of authorization modeling and enforcement described above. A new standard independently authored by the NIST dubbed Next Generation Access Control (NGAC) matches the concepts upon which the EmpowerID Authorization Engine was built.

Per the NIST, "*NGAC diverges from traditional approaches to access control in that it defines a generic architecture that is separate from any particular policy or type of policy. NGAC is not an extension or adaption of any existing access control model, but rather a redefinition of access control in terms of a fundamental and reusable set of data abstractions and functions. NGAC provides a unifying framework capable of supporting current access control approaches as well as novel types of policy that have been conceived yet never implemented due to the lack of a suitable means of expression and enforcement.*"

The key concept behind NGAC is that access rights to perform operations against resources or objects are acquired through relationships referred to as associations. The standard document states that NGAC is inherently more efficient than XCAML policies because authorization decisions are not based on multiple computed and then combined local decisions but rather based on the net result of multiple policies based on relationships existing within a single database. This aspect also allows NGAC to enforce dynamic separation of duties rules which are not fully achievable with XACML. A last key feature mentioned is NGAC's support for "before the fact audit" which is the ability to see who has which access to a resource at any time not just during the real-time evaluation of a policy set.

All the key features outlined in the NGAC standard were design principles built into the previously labeled EmpowerID RBAC/ABAC Hybrid model. EmpowerID's sophisticated role and relationship

modeling allows security architects to model the organization and its structure and policies, including segregation of duties policies to prevent undesired combinations of access. A flexible ABAC-based system supports acting as a centralized real-time decision point for applications that can call the EmpowerID API for authorization decisions. The ABAC engine enhances or modifies the decisions calculated by the powerful RBAC engine, allowing their use only when greater flexibility or contextual information is required. ABAC policies are made much more powerful by the inclusion of the pre-calculated access results that the engine derives from complex RBAC policies that account for inheritance and even attribute-based queries.

For systems that do not support external real-time authorization, EmpowerID provides out-of-the-box support for the Claims, WAM, and Enforcement-based approaches.

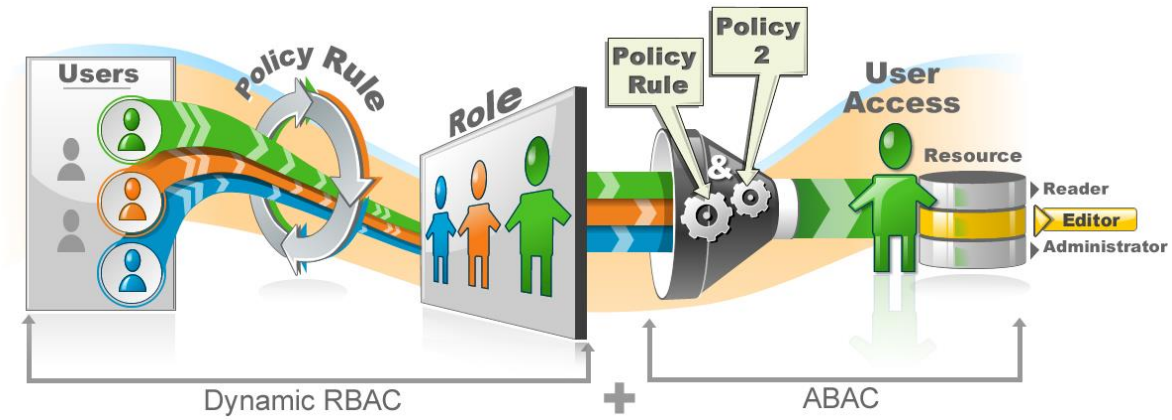


Figure 4. The EmpowerID hybrid RBAC/ABAC “NGAC” access control model

For modeling role-based permission policies, EmpowerID offers a 3-tiered RBAC model with a Business Role tier, a Functional Role tier, and a Technical Role tier.

The top tier or Business Role tier is an implementation of polyarchical RBAC in which a hierarchical Business Role is describing a user’s position in the organization is combined with a hierarchical organizational location that represents which portion of the organization or in which context the user performs their Business Role. EmpowerID represents these as two hierarchical trees with people assigned to the Business Role and a Location combination.

The middle tier is comprised of Functional Roles, known in EmpowerID as Management Roles, represents teams, cross department collaborations, or projects that people might be working on. This tier is more flexible than the Business Role tier in that Management Roles are less tied than Business Roles to a Person’s job description within the organization.

They act in the middle layer as a type of flexible glue to support access requests, role ownership, and many types of temporary access.

The bottom tier is comprised of Technical Roles, which are known in EmpowerID as Access Levels. Access Levels are the system or application-specific roles used to connect the policies in EmpowerID to the actual permissions those policies grant to resources contained within external systems or applications. An example of an Access Level would be the Mailbox Publishing Editor Access Level, which would grant permissions to a mailbox delivered as ACLs within Office 365. Access Levels can grant these “Rights” within external systems and “push” them out via the provisioning engine.

Access Levels may also grant operations which are used when an IT system or application supports external authorization. Operations are synonymous with the Actions of the ABAC world except that they are always about a specific resource of a type as defined within EmpowerID. For example, a user might be granted the “View” operation in an Access Level for all Employee Absence Reports in a department (i.e. Location) or a specific Absence Report. A special Access Level named Member grants the user group membership for groups or applications roles in external managed systems.

EmpowerID optimizes RBAC-based authorization by leveraging the “Role-Centric” and “Dynamic Roles” approaches. EmpowerID is Role-Centric in its use of RBAC objects like roles for the bulk of its permissions assignments.

EmpowerID makes heavy use of Dynamic Roles to automate getting users into and out of roles based on information changes in critical authoritative enterprise systems. Users can be automatically assigned to roles by falling into query-based groupings known as Query-Based Collections. Query-Based Collections themselves are also RBAC actors in the EmpowerID model and can be directly assigned permissions as a type of dynamic role. Query-Based Collections also automate the resource side of RBAC by creating dynamic attribute-based collections of resources for receiving delegations. These collections of resources can use any metadata available for the resource, allowing collections of secured resources to be grouped by security tagging or other information.

SOLVING “ROLE BLOAT” / “ROLE EXPLOSION”

One of the fundamental weaknesses cited in the RBAC approach is that its inflexibility leads to the creation of too many roles to create the authorization policies needed by an organization. Often organizations with an inflexible RBAC system will end up managing thousands of roles and be forced to build roles for each simple access case.

EmpowerID has solved the Role Explosion problem in two ways. First is the hybrid nature of the system allowing auditable RBAC policies to be used for the bulk of access management, with ABAC rules being used to handle complex scenarios or when contextual information should be evaluated. The ABAC rules are not required to perform most access control, only to enhance it in

cases where it makes sense to avoid the administrative burden of creating scores of unnecessary roles.

The second way EmpowerID has solved the Role Explosion problem is in the design of its Business Role tier. As mentioned above, the EmpowerID Business Role tier is a polyarchical RBAC model consisting of a Business Role and an Organizational Location tree.

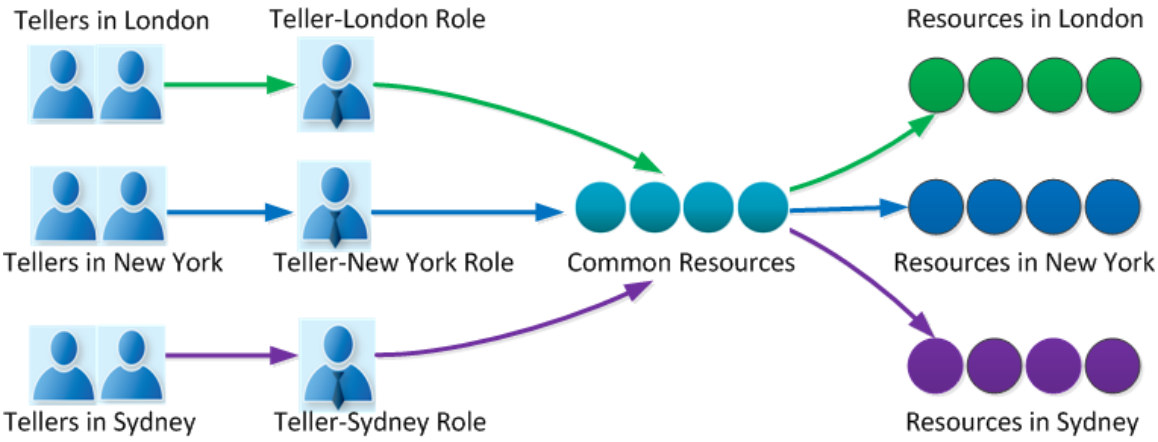
This dual hierarchy allows for access to be assigned based on a combination of what a person does in an organization (their Business Role) and where that person works (their Location).

This intersection of Business Roles and Locations allows for a much smaller "role footprint" than is possible with most approaches to RBAC allowing for more precise assignment of resources to multiple users with the same role.

As an example, let's consider how both models address managing resource access needs for an employee common to banking institutions, the Teller.

It is understood that most tellers perform the same tasks using the same types of resources. So, using RBAC to manage the access to those resources sounds relatively straightforward: You create a "Teller" role, assign all tellers to the Teller role, and then assign to the Teller role the resources tellers need. At this point, everything appears fine. Only one role is needed. However, what if the banking institution has branches located in multiple cities, regions and even countries? Although each teller needs access to a shared pool of resources, by their location, each would also need resources outside the shared pool. Tellers in New York would need access to resources specific to New York, but not London or Sydney; tellers in London would need access to resources in London, but not New York or Sydney; and, tellers in Sydney would need access to resources in Sydney, but not New York or London. In this case, using the same Teller role for all tellers is problematic because doing so would create a "super role," giving each teller access to resources beyond their scope, presenting too great a security risk and violating the concept of minimum privilege. So how does RBAC address this?

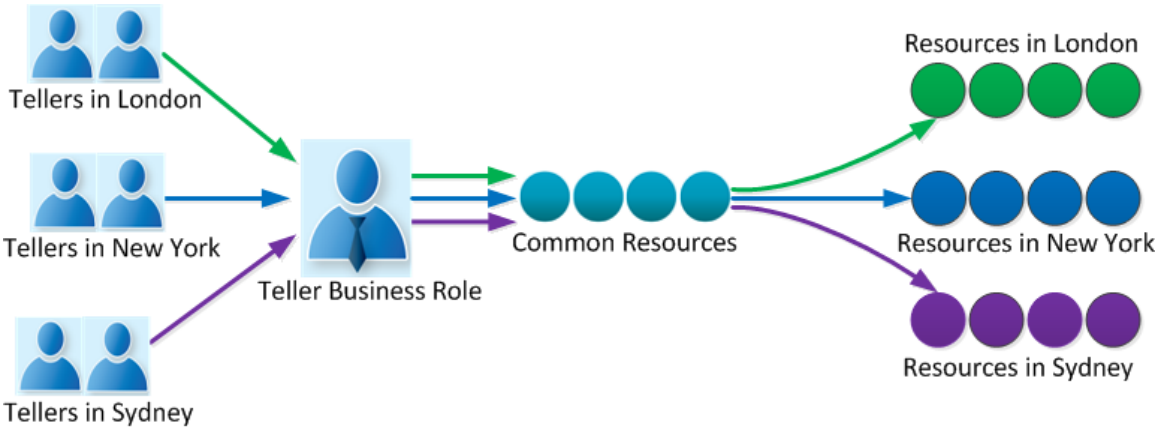
Many typical RBAC implementations create separate roles for each teller location. Thus, tellers in London, New York, and Sydney would have their own roles, such as is depicted by the below image.



While this may appear to be an adequate solution, creating separate roles for each teller location can quickly become unmanageable, leading to role bloat and eventual confusion over which roles have access to which resources. Simply envision tellers working for the same banking institution with branches located in every major city throughout the world.

As mentioned above, the polyarchical model of EmpowerID allows you to significantly reduce an organization's role footprint by allowing resources to be assigned to users based on a combination of their Business Roles and Locations.

This approach makes it possible to address the above role bloat problem using only one Teller Business Role — without needing to create a "super role." This is possible because EmpowerID allows you to differentiate the resources assigned to a Business Role based on the location of those resources. In this way, tellers in London, New York, and Sydney can all have the same Business Role without accessing each other's resources, as shown by the below image.



SOLVING RBAC OVERSIMPLIFICATION (“ISINROLE”)

To recap, our previous example of the over simplification of RBAC using the “IsInRole” approach can be seen in the pseudocode below:

```
if (user.IsInRole(StandardEmployee)
   this.application.show(Employee.PublicProfile)

else if (user.IsInRole(“HRManager”) || (user.IsInRole(“SalesManager”) ||
(user.IsInRole(“AuditManager”)
   this.application.show(Employee.PublicProfile + Employee.AbsenceReport)
```

In this model, application permissions are enforced strictly within the application and hidden from the role management system. This make it impossible to manage or audit the actual access being granted by roles, opening the possibility for security vulnerabilities and abuse.

EmpowerID activates the full power of real-time centralized RBAC and eliminates the need to make hard-coded IsInRole() checks by allowing developers to simply define what needs to be protected, not who can have access to it.

Approaching access control from this perspective yields a highly flexible and granular framework by which explicit controls can be placed on every resource being protected for each type of action. Returning to our Absence Reports example, under this model we would write the code as follows:

```
if (hasAccess(“View”, “Employee.PublicProfile”)
   this.application.show(“Employee.PublicProfile”)

if (hasAccess(“View”, “Employee.AbsenceReport”)
   this.application.show(“Employee.AbsenceReport”)
```

As can be seen, this pseudo-code does not define a role. It simply checks to see if the user is authorized to view the report, showing or not showing it based on the results of that check. This call would be made live to EmpowerID via its API, and the decision returned by the EmpowerID server

would be based on the combined logic and power of all RBAC-based delegations in conjunction with any ABAC rules. This model allows organizations to assign and remove the ability to view the report to or from any user at any time without disruption or the need to rewrite the code block. This model also supports the real-time nature of ABAC as attributes and contextual information can be passed to the decision-making logic or retrieved at runtime by the EmpowerID system. Examples of these might include Spending Limit, Operation or Action, Out of Office Status, Organization Emergency Status, IP Address, authentication type, risk score, and others.

As we can see, combining an RBAC Big Data Engine with ABAC runtime decisions is superior to the limited “IsInRole” approach and provides the benefits of system agility that can respond to changes and provide centralized auditability.

CONCLUSION

There is no need to decide between RBAC and ABAC for enterprise authorization as both are required to meet all an organization’s authorization needs. EmpowerID’s combined hybrid model built in alignment upon the tenets of the Next Generation Access Control (NGAC) standard, retains the advantages of RBAC while integrating the increased flexibility offered by the ABAC approach and adds a new Big Data Analytics dimension to Enterprise Authorization

For more information on the EmpowerID authorization model visit: http://docs.empowerid.com/