Policy-Based Access Controls

By Mary McKee

For a primer on access controls, please see André Koot's Introduction to Access Control.

To comment on this article, please visit our GitHub repository and submit an issue.

Table of Contents

Abstract

The natural evolution of access controls has caused many organizations to adopt access management paradigms that assign and revoke access based on structured and highly reproducible rules.

One such paradigm is known as Policy-Based Access Control (PBAC), which is most differentiated by two key characteristics:

1. Where other access control paradigms often optimize for ease of granting user access to all relevant resources, PBAC optimizes for ease of extending resource access to all applicable users.

2. PBAC facilitates the evaluation of context (time of day, location, etc.) in granting access to a protected resource. Context is used to express who may access a resource and the conditions under which that access is permissible.

Shifting the focus of access controls from the user to the resource allows PBAC systems to be particularly resilient against shifts in organizational structure or regulatory obligations. The inclusion of context (such as an authorized user's location or device) allows for additional security controls to be expressed and extended within resource permissions themselves, ensuring that all facets of access control are contained and auditable within a single structure.

Because PBAC accommodates a very precise expression of who may access a resource and under which circumstances, it lends itself to the automation of access provisioning and deprovisioning in a way that provides ease of management as well as increased security and adaptability.

Introduction

Access control systems provide security and privacy around organizational assets, but they need to be engineered for rapid shifts in technology, regulatory obligations, and organizational structure to do so effectively. Strategies that worked well for limited and largely on-premises information technology infrastructure fare poorly in this age of cloud services, federated identity, and advanced cyber threats.

Most early access management systems utilize what we now refer to as **Discretionary Access Control** (**DAC)**. With DAC systems (such as access control lists), administrators manually assign privileges to users according to their understanding of need, appropriate use, and organizational rules. As DAC systems grow in users, resources, administrators, and/or age, their reliance on ad hoc management leads to inconsistencies in application and understanding of access. As inappropriate access

often goes unnoticed and insufficient access creates visible business challenges, DAC administrators are increasingly incentivized to be liberal with authorizations and conservative with access cleanup. As a result, DAC is often too costly, too inconsistent, and too inflexible for modern needs.

Contemporary access control systems aim to promote consistency and efficiency by granting access to resources through structured rules. Perhaps the best-known model for abstracting access control so that permissions are based on rules is known as **Role-Based Access Control (RBAC)**. Through RBAC, permissions are associated with "roles" which are assigned to users. This model is effective in ensuring that users with the same responsibilities are consistently granted the same permissions and encourages governance by requiring that roles and their associated permissions be defined before they can be used. Further, RBAC is suitable for use in federated authorization scenarios where resource permissions depend on the information provided by an external user authority. While these are improvements over DAC, RBAC permissions are not resilient against shifts in responsibility structure within an organization and are limited in how permissions can be defined. These drawbacks, covered later in this article, make it difficult for RBAC systems to ensure that users do not have more access than they need to perform intended business functions (also known as the *principle of least privilege*[i]).

Policy-Based Access Control (PBAC) is a more robust paradigm for managing permissions through structured rules in federated or non-federated contexts. While the RBAC model intentionally bundles permissions, PBAC builds on a concept known as **Attribute-based Access Control (ABAC)** to avoid interdependence of permissions and provide more fine-grained access controls based on information like a user's employment status or job code. PBAC builds on the ABAC model by accommodating the expression of acceptable contexts for access beyond user authorization, increasing an organization's latitude for defining, adapting, and automating access controls.

The discrete nature of PBAC permissions allows these systems to accommodate precise, intuitive, and incremental change to controls as business and regulatory needs emerge without compounding maintenance over time, maximizing potential for a "least privilege" architecture.

Terminology

- **Access control system** – a structure that manages and helps enforce decisions about access within an organization.

- **User** or **Subject** – a person or entity who may receive access within an access control system.

- **Resource** or **Object** – an asset protected by access controls, such as an application, system, or door.

- **Action** – a protected operation available for a resource, such as “view”, “edit”, or “submit”.

- **Permission** – a statement of authorization for one or more subjects to perform one or more actions on one or more objects.

- **Context** – conditions under which an action on a resource is authorized for a subject, such as time of access, location of access, or a compliance state.

- **Federated access controls** – an access control architecture that accommodates separation of user/subject authority and resource/object authority.

- **Discretionary access control** – a pattern of access control system involving static, manual definitions of permissions assigned directly to users.

- **Role-based access control** – a pattern of access control system involving sets of static, manual definitions of permissions assigned to “roles”, which can be consistently and repeatably associated with users with common access needs.

- **Attribute-based access control (“ABAC”) / Claims-based access control (“CBAC”)** – a pattern of access control system involving dynamic definitions of permissions based on information (“attributes”, or “claims”), such as job code, department, or group membership.

- **Policy-based access control** – a pattern of access control system involving dynamic definitions of access permissions based on attributes (as in ABAC) as well as context for authorized access.

- **Principle of least privilege** – an information security best practice ensuring that users in an access control system do not have more access to resources than is necessary for their intended activities.

- **Segment** – a grouping of subjects that may be useful for authorizations, such as full-time employees, undergraduate students, IT administrators, or clinicians.

- **Abstraction** – the practice of identifying and isolating repeated aspects of operations or business logic so that they can be maintained in one place and referenced in many places.

PBAC vs. RBAC: A Comparison

To better understand PBAC structures, it may be helpful to examine how they differ from RBAC.

While the primary lens of RBAC permissions is the user, the primary lens for PBAC permissions is the resource.

RBAC asks, “what types of users do I have, and what may they do in my environment?”. Controls are constructed with **subjects** (who is getting access), **permissions** (what is being accessed or used), and **roles** (what permissions can be assigned to a subject)[ii]. This looks like:

Subject		**Role**		**Permission**
Ada	as	Editor	may	Modify Documents

PBAC asks, “what types of resources do I have, and who/how may they be used or managed?”. Controls are constructed with **subjects** (who is getting access), **actions** (what behavior is being discussed), **objects** (what resource is being accessed or used), and **context** (environmental or other parameters defining acceptable access)[iii]. This looks like:

Object		**Action**		**Subject**	**Context**
Documents	may be	Modified	by	Those with “Editor” job code	On managed devices

Both of these examples abstract subjects to ensure that all editors are granted the necessary permission. In the RBAC example, Ada acquires the permission because she has been assigned to the “Editor” role through a manual or automated process. In the PBAC example, Ada acquires the permission because the subject definition matches her identity information, though the subject definition could also be a manual process, such as the assignment of a group membership.

To make the most apples-to-apples comparison, imagine that an RBAC system adds Ada to an “Editor” role and a PBAC system adds her to an “Editor” group membership that is referenced in access policies. Though these actions may seem nearly equivalent, the PBAC architecture offers the following advantages: the flexibility to support different situations (context), the ability to discretely handle changes without impacting other permissions (modularity), and the capacity to handle real-time permission evaluation (symmetry).

Context

Ada’s employer may be subject to legal or compliance concerns that affect how resources may be accessed. For example, when national security regulation (such as export controls) restricts access from certain types of devices, relevant PBAC policies can be amended to include this stipulation.

If the company requires some form of training before resources can be accessed, this too can be articulated as context. A "certification status" attribute can be maintained for Ada based on records referenced from within or outside the authorizing organization. Ada's permissions can require that this status is current at the time of access. Instead of laborious audit processes or managing infrastructure to revoke and reassign permissions as compliance states change, Ada's access is automatically blocked when she is not compliant with training and automatically restored when she re-certifies her training. Similarly, if Ada must consent to terms and conditions for the access she has been granted, PBAC context can ensure that this has occurred in advance of any interaction with the resource.

For security reasons, Ada may be expected to only access company resources from safe-listed network spaces or with multi-factor authentication requirements that are more stringent than those of users with lesser permissions. As the accountable stakeholders define such requirements, these too can be codified in and enforced through context definitions.

While it may be possible to implement these types of controls outside of access policies, doing so creates challenges. Verifying compliance at the time of assignment of a permission rather than at the time of access can leave gaps where someone who has fallen out of compliance can still access the resource. Further, imposing limitations on access that are expressed outside an organization's access control system hampers administrators' ability to authoritatively speak to whether a user will be able to access a resource. This makes it much more challenging to answer everyday business questions about authorizations, such as whether all recipients of an email will be able to access a resource referenced in the message before it is sent.

Modularity

Because permissions granted by PBAC policies are not inherently interconnected as they are with RBAC, they are highly modular and easier to manage with confidence. When an organization needs to add, remove, or modify controls on a resource, policies for that resource can be adapted exactly as needed without impact on other resources.

When permissions are bundled together, as in RBAC, accommodating new business scenarios requires a broad analysis of existing permission groupings. Often, administrators are forced to choose between a "close enough" access bundle that carries unneeded permissions with it or a proliferation of bundles that becomes increasingly difficult to understand and maintain.

For example, if senior leadership at Ada's company selected her to edit sensitive briefings for their investors, it is likely that she would need access atypical for editors. An RBAC system admin charged with granting this access is likely to consider solutions such as:

- Giving all editors the access Ada now needs, thus over-privileging other editors.
- Granting Ada a senior leadership role in addition to the editor role, thus over-privileging Ada.
- Creating a new role for permissions specific to this need, setting a precedent of provisional role creation that undermines access governance and can cause the number of organizational roles to snowball.
- Re-engineering roles to offer a cleaner solution for this business scenario.

Organizations with these kinds of use cases will not find it practical to re-engineer RBAC roles each time exceptional access is needed, promoting an increasingly lackadaisical approach to access management.

Symmetry

When there is a divergence between criteria for granting access and criteria for revoking access in a system, it is common for the system to accumulate permissions that were at one time appropriate but would not be allowable under current policy. PBAC systems evade this problem by defining allowable circumstances for access in a way that can be referenced in real-time.

Since PBAC is an extension of ABAC, PBAC controls easily accommodate fully or partially automated access based on attributes. An institution may wish to automatically grant access to any current employee of a company, or any employee who works at Office X, or any employee who works at Office X and is not currently on personal leave.

By automating how access is assigned, it becomes trivial to automate continuous monitoring of permissions, revoking any that are no longer allowable under current rules. This creates symmetry between provisioning and deprovisioning of access, saving time and ensuring an environment free of undesirable remnant permissions.

PBAC is Practical

One of the most common concerns about PBAC is that it may be overly complex for what seems to be a simple access management scenario.

While PBAC policies may initially seem heavier than other strategies, their added flexibility over traditional approaches benefits modest operations as much as large enterprise settings. Time saved with streamlined RBAC roles can be quickly lost if the business impact of modifying a role (or its many associated permissions) is unclear. This can disincentivize active and responsible management of access controls and hamper growth in an organization of any size.

To illustrate how PBAC can be preferable even in a small organization, consider the following scenario:

JE Plumbing starts as a small business comprised of five plumbers and an owner who handles all administration.

Thanks to an excellent reputation and growing customer base, the owner is able to expand the staff to twenty plumbers who are supported by a business manager, three sales representatives, and two finance specialists.

Over time, JE Plumbing sees an opportunity to expand the company's coverage area and offerings. To accomplish this, they set up two new locations overseen by two new business managers (one of whom was an internal promotion from a finance specialist position). They grow their residential plumber staff to seventy-five and hire twenty-five commercial plumbers. Finance and sales positions are replicated across the two new offices, growing that team from two to six. A dedicated marketing specialist is hired to cover all three sites.

An RBAC approach to this problem might start with two roles: an admin role for the owner and a technician role for her staff. As the company grows, a business manager might be trusted with the admin role, but new roles would need to be created for the sales and finance specialists. After doubling from two to four roles, the role count doubles again as the company splits the technician role into commercial technician and residential technician, splits the sales and marketing role into distinct roles, formalizes roles for business managers and customer service, and retains the original admin and finance roles.

Though this example looks at JE Plumbing's development at three points in time, it is unlikely that the company would implement such broad shifts overnight. To preserve security through incremental shifts in responsibility, a small business making strategic organizational adjustments with limited working capital should consider the absence of a role not included in this exercise: that of a full-time IT professional available to perpetually re-engineer RBAC roles and adapt each system utilizing them.

A PBAC approach would start by looking at what resources JE Plumbing needs to secure: work orders, customer information, invoices, inventory, employee personal and licensing information, payroll data, and expense reports. Though responsibility for these functions changes as the company adds staff, the functions themselves remain the same. If the company expanded the nature of their business in addition to the scale, permissions could easily be added to support the new functions without interfering with existing functions.

This simple shift from expressing access controls from user-focused to resource-focused allows for access control complexity to grow linearly rather than exponentially. As a result, JE Plumbing is able to adapt authorizations in step with organizational shifts without managing a ballooning number of roles.

In addition to being more sustainable, PBAC also creates opportunities for the company to reduce risk by setting the context for access. For example:

- When technicians can see all customer information, customers are at risk of privacy violations, and the company is at risk of an employee exfiltrating that information to help them start their own competing company. Perhaps technicians need to see addresses to navigate to job sites but only need to see information associated with open jobs assigned to them. Customer service may need to see phone numbers and email addresses for all customers but may not need address information.

- Only technicians making rounds need access to job information from out of the office, so restricting other users' access to internal IP addresses is an easy way to reduce the cyberattack surface for the company's systems.

- Overexposure of work order information encourages employee speculation about how the business is being run, which can result in misunderstandings or inappropriate disclosures about operational practices.

- When technicians can be assigned to jobs at a business manager's discretion, there is a risk of a technician being sent on the job with a lapsed license. Policy-based permissioning can require valid licensing before job assignment can occur.

Although organizations with modest access management needs may initially choose to forgo PBAC features such as context limitations on access policies, committing early to PBAC architecture for access controls allows for an organic and natural maturation of access management rules over time - whether it be to accommodate more users, more resources, and/or a more sophisticated security or risk management posture.

When RBAC is Preferable

This article has primarily compared policy-based access controls to role-based access controls due to the prominence of RBAC as an access control strategy.

Some IAM professionals may be interested in implementing PBAC controls but must work with systems that can only support RBAC. In these cases, it is sometimes advantageous to rethink institutional roles in terms of resources or specific work functions rather than permission bundles that will be difficult to adapt over time. As long as an RBAC system accommodates multiple roles for a user, it should be possible to achieve some advantages of PBAC (like modularity) within that system.

When choosing between RBAC and PBAC, it may be helpful to consider that PBAC can be constructed to behave like RBAC more reliably than the reverse. For example, an organization that prefers to think in terms of "roles" may choose to represent group

memberships as such, assigning those groups to many resource permissions to the same end effect - one action results in the application of a defined set of permissions. Conversely, options for applying a notion of context to RBAC permissions are often limited.

While the increased flexibility and scalability of PBAC make it a strong choice for protecting sensitive resources, it does require a degree of system-level thinking that may be less approachable for casual users of an access management system. Systems with fairly static access controls and those that delegate access management to end-users rather than administrators (such as those where content creators can authorize collaborators) may find that the approachability of RBAC is more valuable than the robustness of PBAC.

Implementing PBAC

The key to building any successful access control environment is managing abstraction thoughtfully. Too much abstraction, and your access control landscape becomes unintuitive to manage, inviting errors. Not enough, and your environment is too brittle to evolve with business requirements.

To maximize return on your investment in policy-based access controls, consider the following guiding principles:

Build Reusable Components

Managing abstraction in PBAC means isolating parts of your policies that may be applicable to other policies. The most obvious place where this applies is with user segmentation.

For example, if you are constructing a policy to say that:

Object		**Action**		**Subject**	**Context**
User profiles	may be	Updated	by	Business managers	For full-time employees

“Business managers” and “full-time employees” are very likely to be used again in other policies. Thus, creating a definition for these segments that can be used by one or more policies is wise.

The ideal way to avoid these definitions becoming too granular and rigid is through access management system implementations that allow for set logic - particularly intersections (membership in set A AND set B), unions (membership in set A OR set B), and complements (membership in set A, BUT NOT set B).

To expand on the previous example, if the policy above requires the following update:

Object		**Action**		**Subject**	b
User profiles	may be	Updated	by	Business managers at the Detroit office	For full-time employees ***at the Detroit office***

The best way to solve this problem is usually[iv] to keep your definitions of "business managers" and "full-time employees" and add a third: "Detroit office employees." The subject of your policy can then be updated to apply only to users who are business managers AND Detroit office employees.

If your system cannot perform these types of operations, you may have to have different definitions for "Business Managers" and "Business Managers at the Detroit Office." The more specific and nuanced your groups get, the more difficult the system will be to maintain, but good naming conventions can help future policy writers understand the nuances between segments, such as "Employees - All," "Employees - Detroit," and "Employees - Durham."

This approach makes it possible to achieve the same ease of assigning a permission to a group of individuals as you might in RBAC, with the benefits of avoiding interdependence between permissions, being able to cleanly segment objects as well as subjects and being able to easily segment things other than people (such as device identifiers, IP address ranges, document classifications, etc.)

While it's often best to have policies reference an externally managed segment/group/set of users/resources so that multiple policies can use the same set (possibly in different ways), policies should be directly referenced by the resources they support. Having abstraction here only creates room for divergence between the expectation and reality of access management.

Reconcile Static Access Controls

While PBAC is traditionally used for real-time authorization to resources, it is sometimes necessary to integrate access policies with provisioning mechanisms for resources incapable of referring to access policies in real-time and depend on static, internal permissions.

Resources that cannot directly reference PBAC policies are prone to accumulating vestige permissions that are no longer allowable under current organizational access rules. For example, employees who transfer between departments may need some new permissions on a resource and only a subset of their previous ones. Deprovisioning access only when a user leaves an organization or at an administrator's discretion burdens administrators with negotiating appropriate

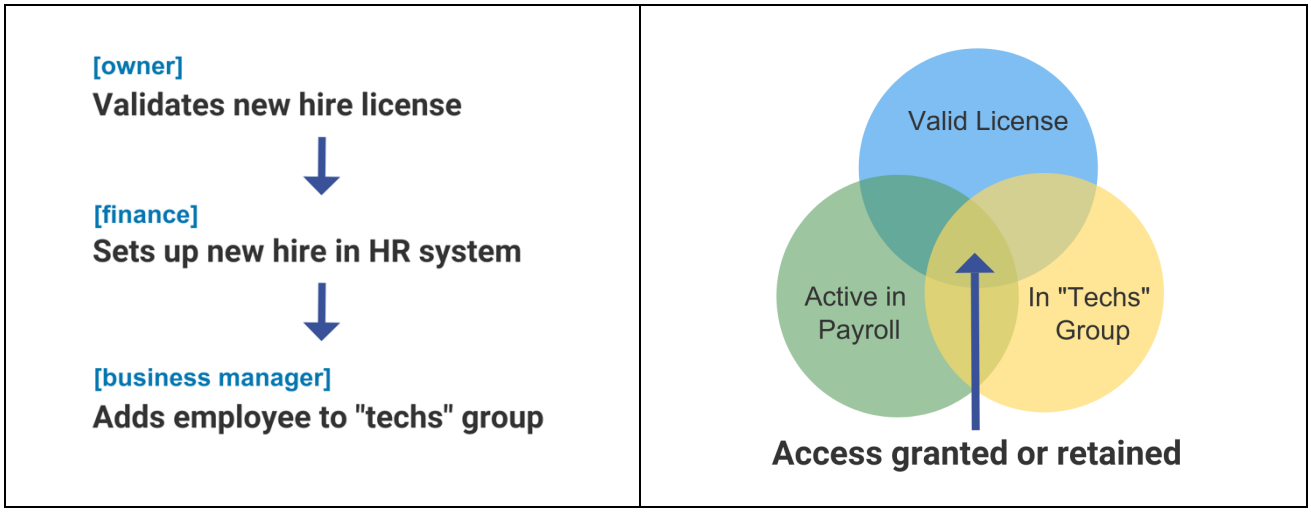
transition plans and often frames such discussions as a matter of a user's trustworthiness rather than organizational policy.

Whether or not it is feasible for a PBAC environment to automate *provisioning* of access in this setting, it should always be feasible to at least partially automate its revocation. Rather than spending time manually removing access that is no longer needed, access management overhead is much better allocated to defining policies that will result in the immediate revocation of access that is no longer authorized, and that will support the organization through blurrier access transitions, such as grace periods or sponsored and time-limited policy exceptions.

Legacy permissions in more static access control settings are sometimes the result of a faulty automation process rather than administrator inaction. These legacy permissions usually accrue because deprovisioning processes are triggered by *events* (e.g., an employee is terminated) rather than *states* (e.g., a user is not an active staff member). Rather than monitoring for a user being hired into position A, or promoted into position A, or transferred into position A, it is better to monitor for compliance with a PBAC policy that determines whether a user is currently in position A. This approach both streamlines policy expression as well as evades problems brought about by process changes that are not always communicated to access administrators. In this way, Human Resources can introduce or change internal processes about how a user comes to be in position A without inadvertently affecting access provisioning.

Business processes are often developed with flowcharts, which are event-driven. As a result, access management systems are often implemented in an event-driven way. To workshop access rules that can be turned into robust PBAC policies, consider dropping the flowchart arrows and working only with circles representing conditions. Arranging these circles as a Venn or Euler[v] diagram allows for a discussion of acceptable conditions for access that will result in cleaner and more robust policies.

Event-based Permission Design	**State-based Permission Design**
Looks like: Flowcharts **Results in:** Rigid and sequential workflows, point-in-time validation, complicated deprovisioning logic.	**Looks like:** Overlapping circles **Results in:** Flexible and parallel workflows, continuous validation, harmony between provisioning and deprovisioning.



There are several reasons why event-based provisioning decisions undermine the best intentions in an access control system:

- There are generally many more events that can lead to states than states themselves, creating unnecessary complexity.

- Access management teams are much more likely to be informed of changes to relevant states (e.g., employment, company policy, business functions) than to changes to events (e.g., how many processes can be used to hire staff, changes to the company network, infrastructure upgrades, etc.). When departmental processes shift in ways that affect detection of events driving access, access management teams become responsible for investigating the resulting inconsistencies and cannot be confident that their systems function as intended.

- Events occur at a point in time, which makes them more difficult to audit for appropriateness. For example, someone might have access through a legacy process that has since been revised (and should retain access) or because of a network glitch at the moment deprovisioning was attempted (and should lose access). Without a current policy to compare against, it becomes very difficult to determine whether existing permissions are appropriate, further eroding trust in the system.

Defining access in terms of observable states rather than events allows for access rules to be expressed as PBAC policies, facilitating continuous reconciliation of controls and eliminating legacy access even in systems that cannot reference those policies in real-time. This approach also supports the evolution of access controls through comparison of access allowable under an existing policy vs. a proposed policy.

Preserve Privacy

More academic guidelines around PBAC may refer to “policy decision points” (PDPs) and “policy enforcement points (PEPs). The central authority determining whether someone is eligible for access (the PDP) may expose the criteria for its decision without revealing why a specific individual was included or excluded from the policy. This allows the downstream resource (the PEP) to enforce institutional access policies without propagating sensitive information across systems.

In more practical terms, consider access management for a scientific instrument subject to federal law requiring all users to be either a citizen or legal permanent resident of their country, and additionally with a clean background check performed within the last three years. By setting up the instrument (the P**E**P) to **enforce** policy based on the **decision** made by the access management system (the P**D**P), the managing organization can avoid sending citizenship, immigration status, and background check results to the instrument. In this way, the organization can meet its legal obligations without propagating sensitive user data across the resources it oversees (or, in federated contexts, across organizational boundaries), significantly reducing exposure of this information.

Support Governance and Audit

A good access control system will allow auditors and business owners engaged in access governance to understand existing precedents in organizational access controls, analyze how they may need to be extended or modified, and ascertain the business impact of proposed changes.

When designing a PBAC system, it is important to make sure that subjects, actions, objects, and contexts are stored in a way that makes it straightforward to report on access from any of these perspectives. Business owners and auditors should have easy access to reports that answer questions about access users have, users able to access resources of interest, and allowable contexts for any actions defined for a resource.

The expressiveness of PBAC permissions makes it realistic to define all access considerations within policies. This flexibility is advantageous over implementing additional security measures (such as IP restrictions) outside of an organizational access control system. It allows for a single source of truth about circumstances under which access is allowed.

Being able to report on permissions in this way facilitates the examination of current rules for access to a resource. Good reporting may also include users who currently meet these criteria. Though PBAC is often used in federated contexts where identity (and other contextual) information for all potential users is not available to the resource administrator, such user reports can be helpful for spot-checking, especially in the context of a proposed change. Reports on who would gain or lose access under a proposed policy support business owners and auditors in refining controls to best facilitate organizational needs and security.

Conclusion

Access control systems promote and protect an organization's mission through changes in users, personnel, responsibilities, organizational structure, and legal obligations. Most failures with access management are due to a system architecture that is too manual to scale or too brittle to change at the pace of business without costly and time-consuming re-architecture efforts.

While it is common to try to optimize access control systems for efficiency in *granting* access, a truer measure of a robust access control system is how reliably it can *revoke* access. Policy-based access controls support the security principle of least privilege by offering logical symmetry between access assignment and revocation. By defining requirements for access, even manually assigned access can be dynamically verified for validity, and automatically revoked or reported as soon as that access becomes invalid under current policy.

Engineering access controls from a resource-first perspective and adding a notion of context to these controls allows PBAC systems to maximize resource security over convenience of access assignment. While these systems can initially be more complex than other approaches, the atomic nature of policies and their relative resilience against buildup of legacy permissions makes for a system that is much more maintainable over time as compared to more limited rule-based access management systems like RBAC.

Author Bio

Mary McKee works as Director of Identity Management and Security Services at Duke University, where she studied Computer Science as an undergraduate and was subsequently hired as a web application developer. Her interest in abstraction and interoperability brought her to Identity and Access Management and subsequently, Information Security.

Acknowledgments

The author would like to thank André Koot and Andrew Hindle for their thoughtful responses to earlier versions of this article, and Heather Flanagan, Christienna Fryar, Dave Wible, and Mary Ellen Wible for their feedback and support with its development.

---

[i] "Least Privilege," https://us-cert.cisa.gov/bsi/articles/knowledge/principles/least-privilege (accessed February 10, 2020)

[ii] "Role-based access control," https://en.wikipedia.org/wiki/Role-based_access_control (accessed February 10, 2020)

[iii] "Attribute-based access control," https://en.wikipedia.org/wiki/Attribute-based_access_control (accessed February 10, 2020)

[iv] The examples in this section are meant to illustrate optimizing for set math capability within a context where both the identity provider (or user attribute store) and the service provider (or resource to be protected) exist within a common environment, and does not extend to federated contexts where a service provider may be interacting with one or more externally controlled identity providers. It is, however, worth noting that PBAC (/ABAC/CBAC) can easily accommodate these externalities.

[v] "Euler diagram," https://en.wikipedia.org/wiki/Euler_diagram, (accessed February 25, 2020)