

# RBAC and ABAC: Roles vs. Attributes

Sun, Aug 9, 6:00 PM EDT · <https://scottslab.io/posts/rbac-and-abac-access-control-models>



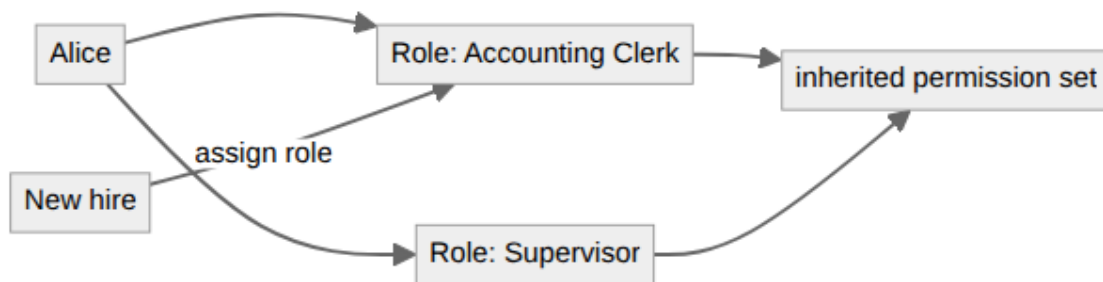
## TL;DR

Where MAC and DAC decide \*who holds the pen\*, RBAC and ABAC decide \*how you express the rules\*. Role-Based Access Control attaches permissions to roles and users to roles, so onboarding and bulk changes become a membership edit instead of a permissions safari. Attribute-Based Access Control goes finer, writing policy over user, resource, and situational attributes. That enables conditional access like "managers see salary data, but only after merit reviews close." Underneath all of it sits implicit deny: anything not explicitly allowed is blocked.

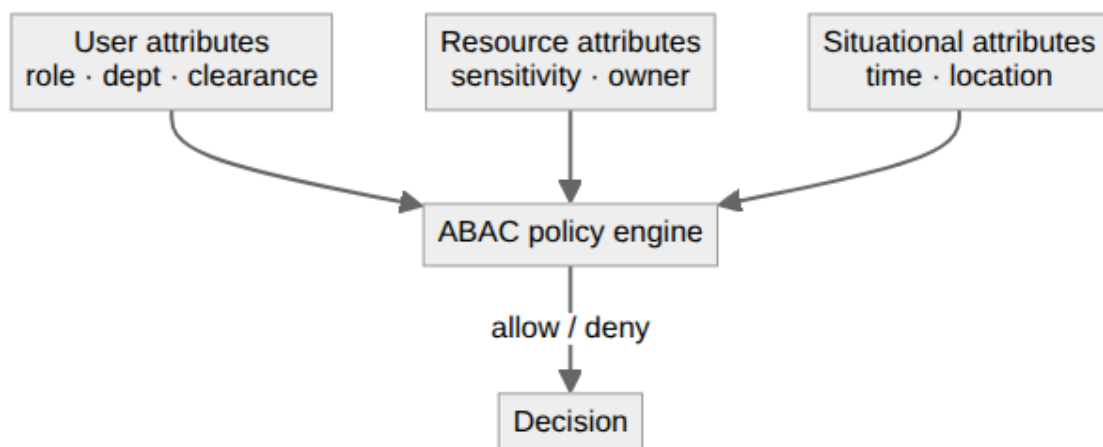
Once you've decided who sets permissions, you still have to express them. Doing it one user at a time doesn't scale. That's the job RBAC and ABAC split between them.

Role-Based Access Control assigns permissions to roles, then assigns users to roles; you never grant rights to a person directly. The payoff is onboarding and change management. A new accounting clerk gets the "accounting clerk" role and inherits its entire permission set instantly, and when the rules change you edit the role once instead of touching everyone who holds it. Roles stack, too. Give Alice both "accounting clerk" and "supervisor" and she inherits the union, say six permissions, automatically. This is also the honest answer to the shared-account temptation from earlier in the

series: roles and groups give you "everyone on the team can get in" without throwing away who-did-what.



Attribute-Based Access Control is the fine-grained evolution. Instead of "which role are you," it evaluates a policy over attributes: something about the user, something about the resource, and something about the situation. That last part is the point. ABAC does conditional access. You can write "managers may read salary data, but only after March 15, once merit increases are finalized," and the same engine treats physical location as just another attribute, so access can depend on where you actually are. It's more powerful and more work to author correctly; you're writing policy, not clicking checkboxes.



Whatever model expresses the rules, one default sits under all of them: implicit deny. Anything not explicitly permitted is blocked. That's the safe posture, because it fails closed. The clearest example is a firewall: traffic runs down the rule list, and if nothing matches, the connection is dropped. Build access that way everywhere: allow what you mean to allow, and let the absence of a rule be a no.

