Entitlement Strategy — RBAC, ABAC, RuBAC & Permissions for AI Agents

From role gates to policy bound to data. How RBAC (role-based), ABAC (attribute-based), and RuBAC (rule-based) access control actually relate; why role-based perimeters — WordPress roles, AWS IAM, Azure RBAC — stop at the door; and how the entitlement plane extends the same discipline to a subject the incumbents were never designed for: the AI agent.

1 · Thesis

Every mainstream permission system answers one question: *who may open the door?* The questions that decide real-world outcomes are different: *what does this specific subject hold, right now, on this specific object, under which terms — and how do we take it back?* The entitlement plane answers those. Policy is **data** (purchase-granted roles carrying scoped capability caps), enforcement is at the **data plane** (per-asset envelope encryption; possession of bytes is worthless without a grant), revocation is an **operation** (rotate one key, every outstanding copy dies), and every decision leaves **evidence** (a hash-chained ledger, verifiable against a published key at `/.well-known/jwks.json` by anyone, with no account and no trust in the platform).

2 · The door-check pattern: three incumbents, one failure joint

	WORDPRESS ROLES	AWS IAM	AZURE RBAC / ENTRA
Model	Roles = flat bundles of global capabilities (`edit_posts` = all posts)	Identity policies over control-plane APIs; ABAC via tags exists but governs *resources*, not app users	Role assignments at scopes (subscription → resource); ABAC conditions narrow (e.g. blob storage)
Granularity stops at	Site / post-type; per-object requires plugins enforcing in PHP at render time	The AWS resource boundary; your application's users and objects are "your problem" — hence Cedar	The Azure resource boundary; app roles are coarse claims stamped into a token
The naked file	`wp-content/uploads` = public URL, zero policy	S3 presigned URL = bearer access, irrevocable until expiry	SAS tokens = same bearer pattern
Revocation semantics	Role change ≠ recall of anything already served	Policy change ≠ recall of presigned URLs	Assignment change ≠ recall of issued SAS/tokens mid-lifetime

The common structure: all three are **perimeter authorization over containers**. None bind policy to the data itself; all three have weak revocation. The industry has already conceded the gap — AWS shipped **Cedar / Verified Permissions**, Google published **Zanzibar** (now OpenFGA), and **OPA** exists, all because role gates do not compose into application-level authorization. Each of those is a policy-as-data engine. The entitlement plane is the same conclusion, plus the step none of them take: cryptographic enforcement at the asset.

3 · The layered model is the standard, not a workaround

A role gate at the door with attribute rules underneath is the architecture the standards bodies recommend:

- **NIST SP 800-162** (Guide to Attribute Based Access Control) frames RBAC as a *special case* of ABAC in which the evaluated attribute is "role." The role check is the first layer of the general model — not an alternative to it.
- **Kuhn, Coyne & Weil, "Adding Attributes to Role-Based Access Control"** (IEEE Computer, June 2010) — by authors of the RBAC standard itself — endorses the *role-centric hybrid*: roles set the ceiling of what a subject may ever do; attributes constrain what they can do right now, on this object, in this context.

Where RuBAC fits. Rule-based access control (RuBAC) evaluates condition rules — time windows, network, object state — against a request. In NIST's framing it is not a rival model but the policy half of ABAC: attributes describe the subject, object, and context; rules decide. A mature stack therefore layers all three: an RBAC gate at the door, RuBAC condition rules in the policy engine, and attributes carrying the per-subject, per-object facts those rules consume.

> Purchase-granted roles carrying scoped capability caps are the Kuhn role-centric hybrid, implemented: the purchase grants the ceiling (RBAC); the caps, entitlement state, and consent snapshot are the attributes (ABAC); the evaluation of them per object and per request is the rule layer (RuBAC).

4 · The entitlement plane, concretely

- **Grant, not file.** Assets are sealed in per-asset envelopes (AES-256-GCM). Delivery is server-side decryption against the caller's entitlement, per request. There is no public URL to leak, cache, or scrape.
- **Policy as data.** Entitlements are records — purchase-granted, scoped, per-subject, revocable — evaluated at request time. Changing policy is a state change, not a redeploy.
- **Revocation heals.** Rotating an asset key kills every outstanding envelope copy at once. Refund-driven or abuse-driven revocation is one recorded operation.
- **Decisions leave evidence.** Every mint, grant, denial, download, and revocation lands in a hash-chained ledger; certificates are Ed25519-signed and third-party-verifiable against the published JWKS. This is the accountability record GDPR expects and the evidence artifact CRA Article 14 reporting presumes.
- **Coexistence, not replacement.** The plane sits beside WordPress, AWS, and Azure estates and feeds them: those systems keep governing their containers; the entitlement plane governs the objects and subjects they cannot see.

5 · Agent & AI gating: the subject the incumbents never modeled

Role systems assume a human who logs in occasionally and holds standing permissions. An AI agent is the opposite subject: it acts continuously, delegates, and cannot safely hold a role-wide grant — an agent with "Editor everywhere, forever" is an incident report with a timestamp. Agent authorization requires exactly the properties the entitlement plane already has:

- **Agents are first-class subjects** in the same engine — no parallel permission vocabulary. Agent capability is gated by an explicit cap (`caps.a2a`), granted and revoked like any entitlement.
- **Mandates, not sessions.** Autonomous purchase runs under an AP2-style mandate: a scoped, time-boxed, consent-anchored authorization for a *specific* action envelope (`create_mandate` → `agentic_checkout`). The mandate is the ceiling; caps constrain within it — the Kuhn hybrid applied to machines.
- **Identity travels encrypted.** The acting party is resolved server-side from a decrypted JWE token — never from a claim the agent asserts about itself. An agent cannot name its own principal.
- **Consent is an input, not a footnote.** Agent actions evaluate the subject's consent snapshot at decision time, and the snapshot is signed into the resulting certificate.
- **Every agent action is ledgered.** Agent-initiated grants, purchases, and denials land in the same hash-chained record as human actions — one audit surface, machine and human alike.

- **Tool surfaces are gated.** Agent-facing interfaces (MCP tools) expose only operations the caller's entitlements permit; the tool list is a projection of the grant, not a menu plus a prayer.

> Principle: **agents get entitlements, never roles.** A role is standing power; an entitlement is a scoped, revocable, evidenced grant. Standing power plus autonomy is how AI incidents happen.

6 · Terminology precision

- **MAC — claim the property, not the label.** Classical MAC (Bell-LaPadula lineage; SELinux, classification lattices) means centrally assigned labels no subject can alter. This plane is not label-lattice MAC — creators set sharing states on their own objects. What it *does* have is MAC's defining property: **mandatory, non-discretionary enforcement** — no subject, human or agent, can extend access they hold; possession of bytes confers nothing; revocation executes without the holder's cooperation. NIST SP 800-162 notes ABAC policy can be enforced in exactly this non-discretionary manner.
- **RABAC, not "RBDAC."** Roles whose effective permissions vary dynamically with context is a named, citable model: *RABAC — Role-Centric Attribute-Based Access Control* (Jin, Sandhu & Krishnan, 2012). Purchase-granted roles with runtime-evaluated caps are RABAC with dynamic entitlements. Invented acronyms next to real ones weaken the citable stack.
- **Three control types in the runners — named correctly.** File delivery is *rule-based access control* over entitlement attributes (per-request rules deciding on live grants). Task runners are *least-privilege service principals* under scoped credentials (blast radius bounded by what the credential can reach). Deploy guards, dry-run gates, and publish locks are *workflow gating* — operational governance, not access control. Delegated runs inherit the subject's entitlements, never the platform's.

7 · The application-scope boundary: RBAC-only stacks vs out-of-scope modules

The deeper split is not which acronym a stack implements but *where its enforcement lives*. WordPress, the cloud IAMs, and the modern Next.js stack (Sanity for content, Better Auth or Clerk for identity) all enforce inside the file system or application scope — the app is the guard. The entitlement plane's modules live outside both: policy as data at the edge, enforcement cryptographic at the asset. When the application is compromised, only one of these designs still holds.

	WORDPRESS	AWS IAM / AZURE RBAC	NEXT.JS + SANITY + BETTER AUTH / CLERK	ENTITLEMENT MODULES — OUTSIDE FILE-SYSTEM & APPLICATION SCOPE
Where policy lives	App code + DB, inside the install	Cloud control plane (infra scope)	App middleware + role claims in the vendor's token	Policy-as-data at the edge, independent of any app deployment
Enforcement point	Template/render code	Cloud API perimeter	Route guards & server components — whatever code remembers to check	Every request at the edge, plus cryptographically at the asset
Object granularity	Role → site-wide capabilities	Resource / container	Org & role claims; per-object = custom app code	Per-subject × per-object × per-request entitlement
Re-delegation	Files & links forwardable	Presigned URL / SAS = bearer access	Token holder asserts role; sessions replayable in scope	Non-discretionary: no holder can extend access
Revocation	Role edit ≠ recall of anything served	Policy edit ≠ recall of issued URLs	Session revoke stops future logins; served data is gone	Key rotation kills every outstanding copy at once

App compromised — then what?	Game over: the app was the guard	Infra intact; data within granted scope exposed	Game over: enforcement was compiled into the app	Modules unaffected; the attacker holds ciphertext
AI agents	No model	Service principals with standing power	API keys with standing power	Mandates + `caps.a2a` : time-boxed, consent-anchored, ledgered
Evidence	Logs, if configured	Control-plane trails only	Vendor dashboards	Hash-chained ledger; public JWKS verification

To be fair to the incumbents: Sanity, Better Auth, and Clerk solve *authentication* and role assertion well. The gap is that everything after the door — object authorization, delegation, revocation, evidence — remains application code, in application scope, sharing the application's fate.

8 · Strategy summary

LAYER	QUESTION IT ANSWERS	MECHANISM
RBAC gate (door)	Who may enter at all?	Roles — including the incumbents' (WP/IAM/Azure), which coexist
Capability caps (ABAC)	What can this subject do here, now?	Purchase-granted, scoped caps evaluated per request (NIST 800-162 / Kuhn hybrid)
Agent mandates	What may this *machine* do, for whom, until when?	AP2 mandates + `caps.a2a` + JWE-resolved principal + consent snapshot
Data-plane crypto	What happens if every layer above fails?	Envelope encryption; leak yields ciphertext; rotation = revocation
Evidence	Can anyone prove what happened?	Hash-chained ledger; Ed25519 certificates; public JWKS verification

9 · References

- NIST SP 800-162, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations* — https://csrc.nist.gov/pubs/sp/800/162/upd2/final (PDF: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-162.pdf)
- D.R. Kuhn, E.J. Coyne, T.R. Weil, "Adding Attributes to Role-Based Access Control," *IEEE Computer* 43(6), June 2010 — https://csrc.nist.gov/files/pubs/journal/2010/06/adding-attributes-to-rolebased-access-control/final/docs/kuhn-coyne-weil-10.pdf (DOI 10.1109/MC.2010.155)
- NIST ABAC project overview — https://csrc.nist.rip/projects/abac/
- NIST SP 800-207, *Zero Trust Architecture* (policy enforcement point model) — https://csrc.nist.gov/pubs/sp/800/207/final
- X. Jin, R. Sandhu, R. Krishnan, "RABAC: Role-Centric Attribute-Based Access Control," MMM-ACNS 2012 — https://link.springer.com/chapter/10.1007/978-3-642-33704-8_8
- Amazon Cedar / Verified Permissions; Google Zanzibar (USENIX ATC 2019) / OpenFGA; Open Policy Agent — the industry's policy-as-data concessions.