Proceedings of GICITE, Berlin, Germany (6-7th November, 2026)

WINS *Proceedings*

ISSN: 3153-8304

DOI
10.XXXX/ICXXX.2027.000

Proceeding Paper

Identity and Access Management for AI Agents in Clinical Platforms

Jobinesh Purushothaman Manakkattil

Oracle Corporation

Redwood City, California, USA

jobinesh@gmail.com

***Abstract*—**AI agents in clinical platforms read and write protected health information. Many platforms run them under a shared service account, which hides which agent acted, ignores the clinician whose authority the agent uses, and gives hospitals no control over who may use an agent. This paper presents an identity and access management model for AI agents in multi-tenant clinical platforms. The model has four parts: a distinct identity for each agent in each tenant, a delegation token that carries both the clinician and the agent, a layered access rule with an organization default and group and user exceptions, and an agent lifecycle tied to tenant entitlement. We describe the model as applied on a production healthcare platform and report four design lessons.

***Keywords*—**AI agents, identity and access management, delegated authorization, access control, multi-tenant platforms, healthcare information systems

Published: 6 November 2026

I. Introduction

Clinical systems have mature access control for people. A clinician signs in, receives a role, and opens a chart under rules the hospital has reviewed. AI agents do not fit this pattern. An agent that drafts a visit note or prepares a patient summary is software, but it works with a clinician's patient data and often at the clinician's request. It is neither an ordinary user nor an ordinary back-end service.

The stakes are practical. Agents in clinical settings are used for documentation, summarization, coding support, patient messaging and similar tasks, and each of these touches records that are protected by law and by hospital policy. A single misrouted call can expose a chart to someone who should not see it, and a single missing identity can make it impossible to say afterward which software did so. Multi-tenancy makes the problem harder. A platform provider serves many hospitals, each with its own departments, user directories and policies, and each expects to decide for itself which agents its staff may use. An identity model that works for one hospital's internal tools therefore does not carry over unchanged to a shared platform.

Many platforms give every agent one shared integration account. This is simple, and it fails in three ways: the platform cannot tell agents apart, the agent's access is not limited by the clinician it serves, and the hospital cannot allow an agent for one department but not another. The HIPAA Security Rule requires access control and unique user identification for systems that hold electronic protected health information [1]. A shared account works against both requirements.

This paper aims to define a practical identity and access management (IAM) model for AI agents in multi-tenant clinical platforms, where a tenant is one customer organization such as a hospital. It makes three contributions: five requirements for agent IAM, a four-part model that meets them, and design lessons from a production deployment. A companion paper covers the audit side of the same platform [11].

TABLE I
Building Blocks from Related Work and What Each Provides

RBAC [2]	Permissions assigned to roles
ABAC [3]	Decisions from subject, resource and environment attributes
Break-the-glass [4]	Emergency access with review afterward
Zero trust [5]	Every request authenticated and authorized
OAuth 2.0 [6]	Delegated authorization for applications
Token Exchange [7]	One party acting for another
JSON Web Token [8]	Claims carried in signed form

II. Related Work

Role-based access control (RBAC) assigns permissions to roles and remains the base of most clinical systems [2]. Attribute-based access control (ABAC) decides from attributes of the subject, the resource, and the environment [3]. Break-the-glass mechanisms allow emergency access with review afterward [4]. All three assume a human user who acts directly. Zero trust architecture requires every request to be authenticated and authorized, whatever its network origin [5]. OAuth 2.0 defines delegated authorization for applications [6]. OAuth 2.0 Token Exchange adds a standard way to express that one party acts for another [7], and JSON Web Tokens carry these claims in signed form [8]. Recent work proposes authenticated delegation for AI agents [9] and agent identifiers to improve visibility [10]. These works supply the building blocks but do not show how a multi-tenant clinical platform should combine them. This paper addresses that gap.

Table I lists these building blocks.

Two further observations from this literature shape the design. First, the standards were written with applications and human users in mind, and they leave open how an autonomous or semi-autonomous program should be named, limited and revoked. Second, most proposals for agent identity consider a single organization, whereas a clinical platform must keep each tenant's agents, users and rules separate. The model in this pa-

© 2026 The Authors. Published in the Proceedings of GICITE 2026.

TABLE II
THE FIVE REQUIREMENTS FOR AGENT IAM AND HOW EACH APPROACH MEETS THEM

Requirement	Shared service account	Per-agent credentials	Proposed model	Part of the model that meets it
Distinct identity	×	✓	✓	Agent identity per tenant
Bounded delegation	×	×	✓	Delegation token with scopes limited to the clinician
Tenant control	×	×	✓	Layered access rule set by hospital administrators
Timely revocation	×	×	✓	Short token expiry; lifecycle tied to entitlement
Explicit unattended operation	×	×	✓	Token naming the agent's own tenant-scoped account, with no actor claim

per therefore reuses the standards where they fit, namely signed tokens, token exchange and an explicit actor claim, and adds the tenant-level pieces they do not supply: a per-tenant identity, a layered rule that tenant administrators control, and a lifecycle linked to what the tenant is entitled to use.

III. Requirements

Agent IAM in a clinical platform must meet five requirements. A shared service account meets none of them, and per-agent credentials alone meet only the first.

Table II compares a shared service account, per-agent credentials and the proposed model against these requirements.

1. *Distinct identity.* Each agent has its own identity in each tenant. No two agents share a credential.
2. *Bounded delegation.* An agent that acts for a clinician never has more access than that clinician.
3. *Tenant control.* Hospital administrators decide who may use each agent, without help from the platform provider.
4. *Timely revocation.* Disabling an agent or removing a user takes effect within a known, short time.
5. *Explicit unattended operation.* An agent that runs with no clinician present is identified as such and authorized separately.

These requirements follow established security thinking. The principle of least privilege, that every program and user should operate with the smallest set of privileges needed for the task, was set out for protection mechanisms in computer systems decades ago [12]. Bounded delegation applies the same principle to agents: an agent's authority is not its own, and it should be no wider than the authority of the person who asks it to act. Timely revocation and tenant control follow from a second idea, that the party responsible for the data should be able to withdraw access without asking a vendor. Distinct identity is a precondition for all of these, since a rule can only be applied to an agent that can be named. Together, the five requirements give a checklist that a platform team can use to review an existing design before adopting the model.

IV. Proposed Model

The model has four parts, enforced by an authorization service on every invocation. Fig. 1 shows the request flow.

Each part answers one or more of the requirements. The agent identity gives each agent a name that is unique within a tenant. The delegation token carries that name together with the name of the clinician, so that downstream services can apply the clinician's limits. The layered access rule lets tenant administrators decide who may invoke each agent. The lifecycle ties the agent's ability to obtain tokens to what the tenant is entitled to use. The authorization service combines the four on every invocation, and a request that fails any check does not reach the agent's tools or the patient data behind them. The following sections describe the parts in turn.

V. Agent Identity

Every agent, whether built by the platform provider or by a customer, is registered in a tenant and receives its own identity there. The identity is created once, at registration, and tokens are obtained later, at run time, from a central identity service. Because the two steps are separate, an administrator can disable an agent or rotate its credential without redeploying the agent.

Registration should record who owns the agent, which tenant it belongs to, what it is for and what it is allowed to request. Digital identity guidelines from NIST describe how identities are proofed, how credentials are bound to them and how authentication is assured [13]. Those guidelines were written mainly for people, but several of their ideas carry over: credentials should be issued by a central service, should be protected from disclosure, and should be replaceable without changing the identity they represent. For an agent this means that secrets are held by the platform and not copied into agent code, and that a credential can be rotated on a schedule or after a suspected exposure. Because an identity is scoped to one tenant, an agent that a customer builds for one hospital cannot present itself in another, and the same provider-built agent appears as a separate identity in each tenant that uses it.

VI. Delegation Token

When a clinician invokes an agent, the identity service issues a signed JSON Web Token [8] that names both parties. The clinician is the subject (*sub*). The agent is the actor (*act*), following OAuth 2.0 Token Exchange [7]. The token also carries the tenant, the granted scopes, and a short expiry. Scopes are limited to what the clinician may already do, so the agent cannot exceed the clinician.

Table III contrasts the two forms of token.

An unattended agent has no clinician. Its token names the

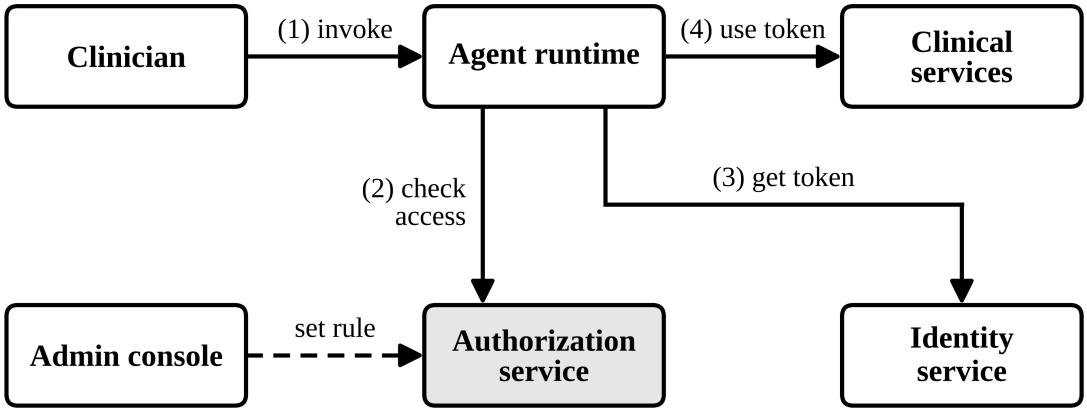


Fig. 1. Request flow. Steps (1) to (4) run on every invocation; the dashed arrow is set up in advance.

TABLE III
CONTENTS OF THE DELEGATION TOKEN

Claim	Clinician-invoked	Unattended
sub	The clinician	The agent's own tenant-scoped account
act	The agent	Absent
Tenant	The tenant	The tenant
Scopes	Within the clinician's own access	Authorized separately
Expiry	Short	Short

agent's own tenant-scoped account as the subject and has no actor claim. Services can tell the two cases apart from the token alone. Downstream services pass both identities on instead of replacing them with their own.

Scopes should be specific. A scope such as read access to all patient records is too broad for an agent that summarizes one visit. OAuth 2.0 Rich Authorization Requests allow a request to describe the type of access, the resources and the actions in structured form instead of a fixed list of scope strings [14], which suits clinical data where the same action may be permitted for one patient or encounter and not for another. A platform can use this to narrow the token to the patient or encounter in the current clinical workflow. Narrow tokens reduce what is exposed if a token is intercepted or if the agent is misled. Two further points deserve attention. The token's lifetime should be short enough that removing a clinician's access takes effect quickly, and long enough that a normal task finishes without constant renewal. And every service that receives the token should check the audience, the expiry and the signature itself and not assume that an earlier service has done so.

VII. Layered Access Rule

Each agent in a tenant has one access rule. The rule holds an organization default, either allow or deny, plus an allowed list and a denied list. Lists hold users and flat groups, defined locally or synchronized read-only from the clinical system.

The rule is evaluated in a fixed order. A user setting overrides a group setting, and a group setting overrides the organization default. When a user belongs to groups that disagree, deny wins. Table IV shows four cases.

Three layers cover the common cases. A hospital can enable an agent for everyone, pilot it with one department, or block one person, all with the same rule.

A fixed evaluation order matters more than the particular order chosen. An administrator who reads the rule must be able to predict the outcome for any user, and an auditor must be able to reproduce a past decision. Letting deny win when a user's groups disagree errs toward protecting data, and the user-level override gives an administrator a way to correct a single case without restructuring groups. Synchronizing groups read-only from the clinical system keeps one source of truth for who belongs where, so the access rule refers to groups instead of copying them. Local groups remain useful for pilots, for example a small group of early users of a new agent. Teams should also decide in advance what happens when a group disappears from the source system, and should prefer a result that denies access until an administrator reviews the rule.

VIII. Enforcement

An authorization service evaluates the rule each time an agent is invoked. It allows the invocation only when three checks pass: the agent is enabled in the tenant, the access rule allows the user, and the requested scopes fall within the clinician's own access. A denial carries a reason, such as a disabled agent, a group exclusion, a missing rule, or the absence of a clinician. Decisions are cached at the enforcement point for a configurable window, two minutes by default. The window is the upper bound on revocation delay. A shorter window gives faster revocation and more load on the authorization service. The platform also shows administrators which users have access to each agent, and why. The view is computed from the live rule with the same logic the authorization service uses, so it cannot drift from runtime behavior [11].

Fig. 2 shows the three checks in order.

Enforcement should be centralized, because separate checks inside each agent drift apart over time. A central authorization service gives one place to change the rule, one place to test it and one source of reasons for denials. The enforcement point should fail closed: if the service cannot be reached and no cached decision is available, the invocation is denied. The cache window needs the most thought. A two-minute window is short enough for most administrative changes, but an emergency removal of

TABLE IV
EVALUATION OF THE LAYERED ACCESS RULE

Org. default	Group setting	User setting	Result	Reason
Deny	Allow in one group	None	Allow	Group exception
Allow	Allow in one group, deny in another	None	Deny	Deny wins
Allow	Deny	Allow	Allow	User overrides group
Deny	None	None	Deny	Default applies

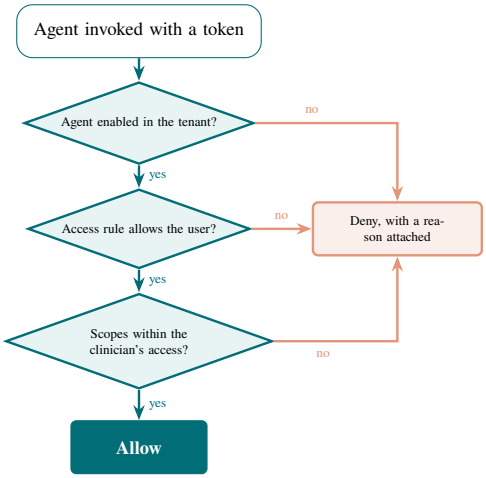


Fig. 2. The three checks applied by the authorization service on every invocation.

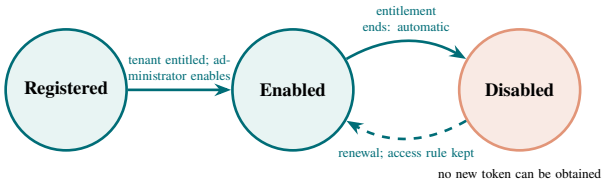


Fig. 3. Agent lifecycle in a tenant, tied to entitlement.

a user may need an explicit cache clear, and a platform should offer that option. Latency is the other cost. Because every invocation pays for the checks, the service should be fast, and its response time should be measured and reported, which is part of the future work described below.

IX. Lifecycle and Entitlement

An agent moves through three states in a tenant: registered, enabled, and disabled. A tenant must be entitled to an agent before an administrator can enable it. When the entitlement ends, the platform disables the agent automatically. The access rule is kept for a later renewal, and a disabled agent cannot obtain a new token.

Fig. 3 draws the lifecycle as a state diagram.

Tying the lifecycle to entitlement benefits the hospital as well as the provider. When a contract changes, no one has to remember to switch off each agent by hand, and no agent keeps running on a lapsed agreement. Keeping the access rule through a disabled period means that a renewal restores the earlier configuration instead of forcing administrators to rebuild it. Two cases need care. An agent that a customer built should be disabled when the customer's own developer account is closed, and an agent retired for good should have its identity and credentials removed, with its records retained as policy requires so that earlier activity can still be reviewed.

TABLE V
DESIGN LESSONS AND WHAT TEAMS SHOULD DO

Lesson	Action
Identity propagation is the hard part	Plan first for services that replace the original clinician and agent with fresh tokens
A management layer helps	Route agent runtimes through one agent management service
Administrators want the answer	Provide a view of users with access
Denials are useful signals	Attach reasons to denied requests

X. Application and Design Lessons

We applied the model on a multi-tenant production healthcare platform that hosts both provider-built and customer-built agents. Four lessons stand out.

Table V restates the four lessons with the action each implies.

1. *Identity propagation is the hard part.* Defining the token was quick. Changing services to carry both identities took far longer, because many issued fresh tokens that replaced the original clinician and agent [11]. Teams should plan for that work first.

2. *Put a management layer between agents and the identity provider.* Agent runtimes and the agent control plane call one agent management service, which talks to the identity infrastructure. The identity provider can then change without touching agent code.

3. *Administrators want the answer, not the rule.* The view of users with access was the feature administrators used most [11].

4. *Denials are useful signals.* Reasons attached to denied requests exposed agents invoked from workflows nobody had documented [11].

Healthcare platforms also have to work with the standards used to connect applications to electronic health records. SMART on FHIR, for example, uses OAuth 2.0 to let third-party applications obtain access to clinical data in a standard way [15]. The model in this paper is compatible with that approach: the delegation token plays a similar role to the access grant, and the tenant rule decides whether the agent may ask for one at all. Platforms that already support these standards can reuse their authorization servers and scope vocabularies, and then add the agent-specific parts described here, namely the actor claim, the per-tenant identity and the lifecycle. The lessons above suggest a sensible order of work: define the token, change the services to

carry both identities, add the management layer, and only then build the administrator views and denial reasons that make the system understandable to the people who run it.

XI. Discussion and Limitations

The model limits damage rather than preventing every failure. An agent misled by a crafted input can still act, but only within the clinician's access, the granted scopes, and the tenant rule. Short token expiry and the cache window bound how long revoked access lasts. A compromised identity service is out of scope, and synchronized group membership is only as current as the synchronization. The paper reports one platform, and its lessons are qualitative rather than from a controlled comparison. It does not address whether agent output is correct or whether a clinician reviewed it.

Prompt injection is the most direct threat to this design. Research on LLM-integrated applications has shown that instructions hidden in content that a model retrieves can steer the model to take actions its user never intended [16]. An agent that reads a patient message or an outside document is exposed in this way. The identity model cannot stop the agent from being misled, but it limits what a misled agent can do: it holds only the clinician's access, narrowed to the task, for a short time, and every action it takes is attributed to both parties. Layered defenses are still required, including input handling, output review and confirmation before consequential actions such as writing to a chart. A related limit is that the clinician's own access may be broader than a given task needs. Narrowing scopes by task reduces this gap, but it requires work to define what each agent needs. Finally, the model assumes that agents call services through the platform. An agent that reaches data by a path that bypasses the authorization service is outside its protection, so the network and service design must make the platform the only route.

XII. Conclusion

AI agents in clinical platforms need their own identity and access model. The proposed model combines existing standards into four parts: a distinct agent identity per tenant, a delegation token that names the clinician and the agent, a layered access rule, and a lifecycle tied to entitlement. Future work will measure authorization latency and revocation delay and will evaluate the model on other platforms. Teams adopting the model can start with the lowest-effort steps, which are to give each agent its own identity, retire shared accounts and attach reasons to denials, and then move to full delegation tokens and the lifecycle.

Acknowledgment

The author used Claude (Anthropic) for drafting, language editing, and preparing Fig. 1.

References

[1] U.S. Department of Health and Human Services, "HIPAA Security Rule," *45 C.F.R. Part 164, Subpart C*.

[2] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, (1996), pp. 38–47.

[3] V. C. Hu et al., "Guide to attribute based access control (ABAC) definition and considerations," *NIST SP 800-162*, (2014).

[4] A. Ferreira et al., "How to break access control in a controlled manner," *Proc. IEEE CBMS*, (2006), doi: 10.1109/CBMS.2006.95.

[5] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," *NIST SP 800-207*, (2020).

[6] D. Hardt, Ed., "The OAuth 2.0 authorization framework," *IETF RFC 6749*, (2012).

[7] M. Jones et al., "OAuth 2.0 token exchange," *IETF RFC 8693*, (2020).

[8] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," *IETF RFC 7519*, (2015).

[9] T. South et al., "Authenticated delegation and authorized AI agents," *arXiv:2501.09674*, (2025).

[10] A. Chan et al., "Visibility into AI agents," *Proc. ACM FAccT*, (2024), pp. 958–973.

[11] J. P. Manakkattil, "Making AI agents accountable in clinical systems," *Proc. IEEE HMSS*, (2026), to be published.

[12] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proc. IEEE*, vol. 63, no. 9, (1975), pp. 1278–1308.

[13] D. Temoshok et al., "Digital identity guidelines," *NIST SP 800-63-4*, (2025), doi: 10.6028/NIST.SP.800-63-4.

[14] T. Lodderstedt, J. Richer, and B. Campbell, "OAuth 2.0 rich authorization requests," *IETF RFC 9396*, (2023).

[15] J. C. Mandel, D. A. Kreda, K. D. Mandl, I. S. Kohane, and R. B. Ramoni, "SMART on FHIR: a standards-based, interoperable apps platform for electronic health records," *J. Am. Med. Inform. Assoc.*, vol. 23, no. 5, (2016), pp. 899–908.

[16] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," *Proc. ACM Workshop on Artificial Intelligence and Security (AISec)*, (2023), pp. 79–90.