SECUREAUTH
Welcome to Better Identity.

HOT TOPIC

AGENTIC AI

INVITING SaaS AI—WITHOUT INVITING TROUBLE

Cross-boundary AI integration introduces risk—but it doesn't have to. This piece outlines how to securely connect SaaS-hosted AI agents using OAuth 2.1, Dynamic Client Registration, and trust registries, enabling scalable, policy-enforced access without sacrificing control, auditability, or Zero Trust posture.

ARCHITECTING IDENTITY FOR AGENTIC AI: PART 4

This is **Part 4** in the series **Architecting Identity for Agentic AI**, a technical guide for software architects securing AI systems with OAuth, OIDC, and trust frameworks. Designed for architects building scalable, policy-driven identity across enterprise and cross-boundary environments.

BY LUKASZ RADOSZ

SVP of Engineering, SecureAuth

With over two decades in identity and access management, Lukasz brings deep expertise in authentication, authorization, and API security to the SecureAuth team. A champion of open standards like OAuth and OIDC, he's an advocate for modern identity architectures across machine identity, open banking, and transactional access control.

Introduction

In this fourth installment of our series on agentic AI system patterns, we address the **External SaaS Pattern**—integrating SaaS-hosted AI capabilities with enterprise systems **securely**. As organizations seek to invite powerful AI services into their architecture, they must ensure they're not also inviting security and privacy troubles. This article focuses on two cross-boundary scenarios and how to manage them using **OAuth 2.1**, **Trust Registries**, and **Dynamic Client Registration (DCR)** with signed software statements. The goal is to equip software architects to safely expose or consume AI functionality across organizational boundaries without sacrificing control, privacy, or auditability.

The External SaaS Integration Challenge

Modern AI agents often need to interface across enterprise and SaaS environments. In an **External SaaS Pattern**, either the AI agent or the "MCP" server (the AI Model Context Protocol server that provides tools/services to agents) is hosted outside the enterprise. We consider two scenarios:

1. Internal AI Agent → External SaaS MCP

An AI agent running inside the enterprise network needs to connect to an external SaaS-hosted MCP server (a service providing AI tool APIs or context).

2. External AI Agent → Internal MCP

An AI agent running in a SaaS platform (outside the company) needs to access an internal MCP server or enterprise API to perform actions on behalf of the enterprise.

Each scenario presents security hurdles. The enterprise must ensure that only trusted, authorized agents and services communicate, that sensitive data remains protected in transit, and that all access is auditable. The integration must scale to potentially many agents and services—manual credential handshakes won't cut it when dozens of AI agents spin up or down daily. We need standardized, automated, **OAuth 2.1 based solutions** to manage authentication and authorization at scale.

OAuth 2.1: A Secure Baseline for AI-to-SaaS Integration

OAuth 2.1 provides a modern, hardened baseline for delegating access securely across systems. It isn't a new protocol but a security-focused evolution of OAuth 2.0 that removes unsafe practices and mandates proven ones. By using OAuth 2.1, architects ensure that only robust flows (e.g. authorization code with PKCE, no implicit grants) are used and that latest best practices are followed. In fact, industry initiatives like the Model Context Protocol (MCP) spec lean heavily on OAuth 2.1 for AI agent authorization.

Key benefits of OAuth 2.1 for our scenario include:

- **Standardized AuthZ flows:** Agents obtain access tokens to act on resources, with explicit scopes and consent. No sharing of passwords or API keys—instead, short-lived tokens with least privilege.
- **Removal of legacy risks:** Insecure flows (like the implicit grant) are eliminated and secure defaults (PKCE, refresh token rotation, etc.) are built-in.
- **Proven interoperability:** OAuth is vendor-neutral and widely supported, making it ideal to bridge disparate systems (enterprise and SaaS).
- **Audit and consent tracking:** OAuth flows naturally log which client (agent) accessed what, and can integrate with consent dashboards for users or admins.

In both scenarios, we will use OAuth 2.1 as the foundation: the AI agent will act as an OAuth client and the MCP (or API) as a resource server protected by an OAuth authorization server. The twist is ensuring trust in an otherwise open OAuth registration model – that's where **Trust Registries** and **Dynamic Client Registration** come in.

Trust Registries for Controlled Onboarding

Trust Registry in this context means an authoritative list or service that keeps track of which external parties (SaaS agents or services) are trusted to integrate with your system (and vice versa). Instead of allowing any unknown AI service to connect, both the enterprise and SaaS provider maintain a controlled roster of trusted partners, including the identity and metadata of each AI client or service. This registry could be run by the enterprise itself or by a neutral body (in regulated ecosystems, a central directory often fulfills this role).

Think of a Trust Registry as a "permissioned phonebook" for OAuth clients and servers. When an external AI agent or SaaS MCP wants to integrate, it must first be onboarded into the registry (usually via a vetting or agreement process). Once listed, that agent or service can dynamically establish OAuth credentials with the counterpart system, proving its identity via the Trust Registry. The registry typically stores details like organization identity, software application IDs, public keys or certificates, and what level of access is allowed.

How Trust Registries enforce control:

- **Pre-vetting:** Only organizations/apps that have been vetted (e.g. contract in place, security review, compliance check) get an entry in the registry. Unknown parties are rejected.
- **Credential binding:** The registry might issue digital credentials (like a certificate or a signed token) to the trusted partner. This credential will later be used in OAuth flows to prove "I am an approved partner."

- **Dynamic verification:** When an OAuth client from a partner registers or requests access, the authorization server checks the presented credentials against the Trust Registry. If the client isn't in the registry or the credential is invalid, the request is denied.
- **Revocation:** If a partner is compromised or should no longer have access, removing them from the Trust Registry (and revoking their credential) will prevent any new token issuance or registrations from them. This gives a central kill-switch for access.

In summary, the Trust Registry concept lets us automate client onboarding **without throwing the doors wide open**. It provides the necessary guardrails so that only invited SaaS AI or services can connect, and that their identity is verified by a source we trust.

Dynamic Client Registration (DCR): Self-Service Onboarding for OAuth

Dynamic Client Registration (RFC 7591) is a protocol that allows OAuth clients to register themselves with an authorization server on the fly, by calling a standard **/register** endpoint with a set of metadata. Instead of manually pre-registering every client (and issuing client IDs/secrets out-of-band), DCR enables **programmatic onboarding**. This is vital in an agentic AI context where you may not know ahead of time all the AI agents or services that will need to connect – they might be spun up dynamically, or belong to external organizations.

However, naive use of DCR (open to anyone) would be dangerous—a rogue app could register and obtain client credentials if the endpoint isn't locked down. That's why we combine DCR with the Trust Registry and strong authentication of the registration request.

Signed Software Statements for Trusted DCR: The solution adopted in many ecosystems is to require the registering client to present a **software statement**—a JWT containing the client's details, **signed by a trusted authority**. The authorization server will verify this JWT using a known public key (the trust anchor). If the signature and details check out, it knows the client is legitimate per the trust registry. Essentially, the software statement is like a letter of introduction from the Trust Registry, saying "this client is bona fide."

According to RFC 7591, a software statement is "a digitally signed JSON Web Token (JWT) that asserts metadata values about the client software." The trust relationship the auth server has with the issuer of this JWT determines whether the registration is accepted. In practice, this means if the JWT is signed by a known authority (e.g. the enterprise's partner registry or an industry directory), the OAuth server will trust the contained client data and complete the registration.

Real-world analogs make this concrete. In **OpenBanking UK**, third-party providers (TPPs) must first register with the Open Banking Directory and obtain a software statement (and an SSL certificate). When a TPP wants to integrate with a bank's APIs, it uses that software statement in a DCR request to the bank. The bank's authorization server validates the JWT signature against the Open Banking Directory's public key and only then issues client credentials. Moreover, the request is sent over mutual TLS with the TPP's certificate, adding an extra layer of authentication. This ensures **only authorized, identified fintech apps** can connect to bank APIs—a random app can't just show up and register. In fact, the OpenBanking spec mandates that the AS (bank) **must not register a client** unless the software statement is verified and trusted.

Similarly, in **Australia's Consumer Data Right (CDR)**, accredited data recipients obtain short-lived SSAs (Software Statement Assertions) from a central Register to include in their DCR calls. Data holder APIs are required to reject any client registration that isn't accompanied by a valid, signed SSA. The rule is explicit: "Data Holders MUST NOT register Data Recipient software products unless they present a verified registration request with a verified SSA." Only after successful dynamic registration does the external app get a client ID and secret to use in OAuth flows.

Dynamic Registration Flow with Trust Registry (Overview)

To illustrate how DCR, trust registry, and software statements come together, let's walk through the onboarding of a new SaaS AI integration in our enterprise context:

1. Trust Onboarding
Out-of-band, the enterprise and the SaaS provider agree to integrate. The SaaS provider (or its specific AI agent application) is added to the enterprise's Trust Registry. This might involve the provider proving who they are (through business agreements or even exchanging a digital certificate). The trust registry generates a **software statement JWT** for the SaaS provider's application or the provider supplies one from an agreed authority.

2. Discovery
The new AI agent or MCP (client) discovers the authorization server metadata of the other party. For example, an external agent discovers the enterprise OAuth 2.1 endpoints (authorization, token, registration URLs) via a **.well-known endpoint**, or an internal agent finds the SaaS OAuth endpoints.

3. Dynamic Client Registration
The AI client submits a POST /register request to the authorization server, including its metadata (redirect URIs, contact info, desired scopes, etc.) and the signed software statement. This request itself must be authenticated – commonly by using an initial credential:

- For external clients, this could be a mutual TLS connection using the client's certificate issued via the trust framework, or a JWS signed with a private key that the server can verify against the software statement's key reference.
- This ensures the entity calling /register is indeed the one the software statement was issued to (preventing token replay by others).

4. Validation
The authorization server validates the software statement JWT:

- Checks the issuer (**iss**) to ensure it's the trusted authority (e.g. "OpenBanking Ltd" or the enterprise's registry).
- Fetches the issuer's JWKS (public key set) and verifies the JWT signature.
- Verifies claims like expiration, audience, software ID, etc., possibly checking that the software ID or organization matches an approved entry in its trust registry database.
- If any check fails, registration is denied. If all good, the client is considered trustworthy per the registry.

5. Client Provisioning
The authorization server creates a new client record for the AI agent/MCP. It assigns a **client ID** and generates credentials (either a client secret or sets up the client's public key for JWT-based auth). The response to the client contains the client ID, and possibly the client secret if using a shared secret.

6. Registration Outcome
The AI agent now has OAuth credentials to use. Both parties have a record of this client in place. The client might cache the credentials for re-use until it expires or is rotated.

At this point, the SaaS AI agent or service is onboarded in a controlled manner—it has the necessary OAuth client identity within the enterprise (or vice versa), and that identity was only issued because it was in the trust registry.

Let's visualize the onboarding and subsequent token flow for each scenario.

Scenario 1: Internal AI Agent → External SaaS MCP (Outbound)

In this scenario, an **internal AI agent** (inside the enterprise network) needs to call an **external SaaS-hosted MCP** (to utilize external AI tools or data). The enterprise must ensure outbound calls are secure and controlled—e.g., the agent should only access the SaaS MCP endpoints with proper authorization, and the SaaS should trust that the call is from a legitimate client (not a random attacker).

The flow typically goes as follows, assuming the SaaS supports OAuth 2.1 and DCR for clients (many SaaS APIs do allow client registration via console or API):

- The enterprise's AI agent (or its developer) discovers the SaaS OAuth endpoints (either via documentation or an OAuth discovery document).
- If the SaaS requires registration, the agent (as an OAuth client) **registers dynamically** at the SaaS authorization server. The enterprise might supply a software statement if the SaaS has whitelisted the enterprise via a trust program (optional step). Otherwise, the SaaS might allow open self-service registration with basic verification.
- The SaaS authorization server creates a new client for the agent and returns a client ID (and secret or certificate requirements).

- At runtime, when the agent needs to call the SaaS MCP, it performs an OAuth flow to get a token. Often this could be a client credentials grant (if the agent acts as itself with a pre-arranged trust), or an authorization code grant if acting on behalf of a user (where the user would authenticate and consent to the SaaS access).
- The agent obtains an access token from the SaaS auth server, with scopes that allow the needed operations on the MCP.
- The agent sends the API requests to the SaaS MCP, presenting the access token in the Authorization header.
- The SaaS MCP validates the token (locally if it's a JWT or by querying its auth server) and then serves the request, returning results to the agent. All actions are logged.

Security considerations: The enterprise should maintain an allow-list of approved external SaaS MCPs that its agents are permitted to use (this list is effectively a trust registry of external services). The OAuth client registration ensures the external service knows which specific client (the enterprise's agent) is calling—the client ID represents the enterprise agent, and can be scoped to certain permissions. No internal data is directly exposed; the agent only sends what's necessary and receives results under the SaaS's token protections. The enterprise can audit what tokens were issued and what data was pulled via logs on the SaaS side (often accessible via the SaaS's developer portal).

Below is a sequence diagram summarizing the key interactions for Scenario 1:

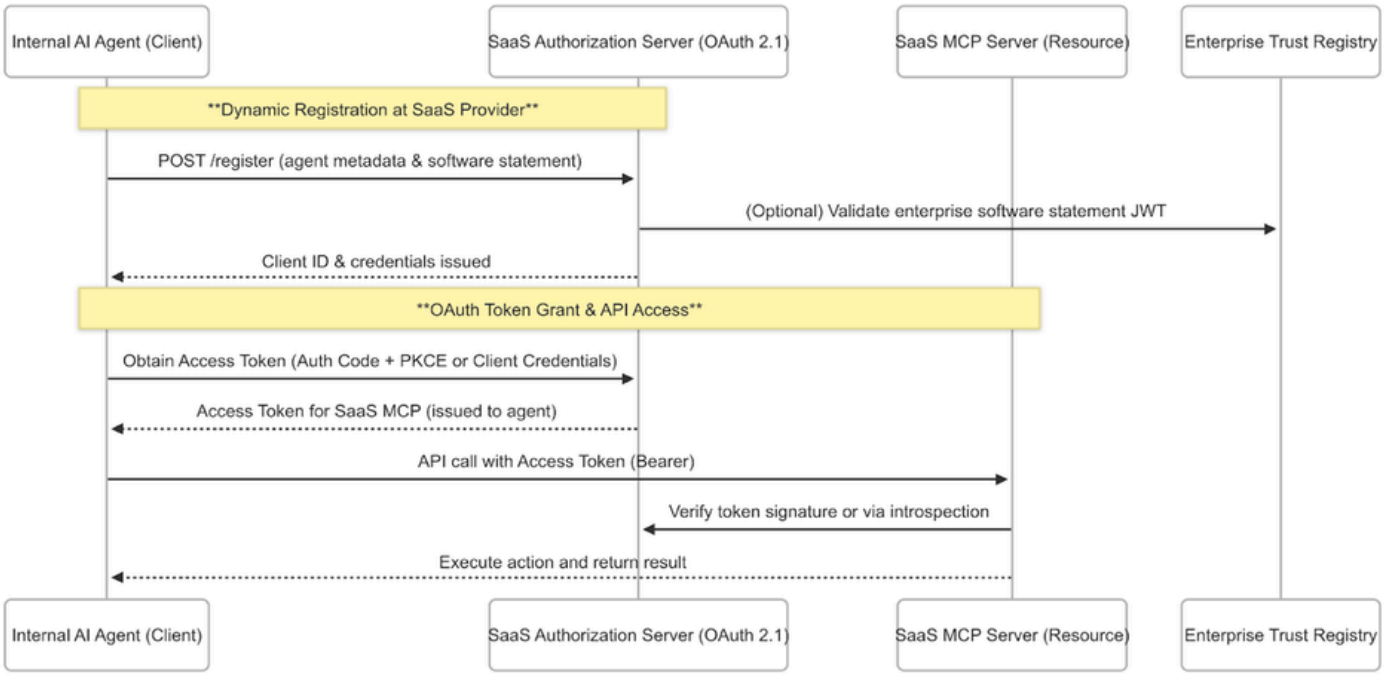

Diagram: Internal agent dynamically registers with external SaaS (possibly presenting a signed software statement to prove enterprise identity). The agent then performs the OAuth flow to get a token and call the SaaS MCP API securely.

In many cases, the external SaaS might not require a custom software statement—it could simply allow the agent to register since the SaaS trusts any client up to the point of requiring user authorization or API keys. But for deep enterprise integration (e.g., a SaaS that is an AI co-pilot explicitly for your company), the SaaS could issue your enterprise a software statement or expect one from you to verify the client belongs to an allowed organization. The optional Trust Registry check in the diagram represents that kind of arrangement.

Scenario 2: External AI Agent → Internal MCP (Inbound)

This scenario flips the perspective: a **SaaS-hosted AI agent** needs to call into an **enterprise's MCP** server or APIs. Here the enterprise is the one hosting the protected resource, and the external agent is the OAuth client. This is analogous to a third-party app trying to access a company's API – a high-risk operation without proper controls. The enterprise must strictly authenticate the external agent and only allow it to do what's permitted.

We address this by leveraging the same OAuth 2.1 + DCR approach, now with the enterprise authorization server in control. The external agent must register as a client and obtain tokens to access internal APIs. Crucially, the enterprise uses the Trust Registry to decide which external agents can even register.

The flow for Scenario 2:

- **Trust establishment:** Before any tech calls, the external SaaS provider (hosting the agent) is onboarded in the enterprise's Trust Registry. For example, ACME Corp (enterprise) approves "Contoso AI Agent SaaS" as an authorized partner. ACME might issue Contoso a signed software statement JWT and share its OAuth endpoints. Alternatively, Contoso might have an authority sign a statement that ACME's systems trust.
- **Dynamic client reg:** The external AI agent (or its controlling service) calls ACME's **/register** endpoint on the enterprise authorization server, including the software statement. The call is made over a secure channel (e.g., mutual TLS or signed JWT auth) so that ACME can confirm the caller's identity matches the statement.
- **Validation:** ACME's auth server verifies the software statement signature against the trust registry's public key. If valid and the metadata is acceptable, it creates a client record for Contoso's agent. Now Contoso's agent has a client ID (and maybe a client secret or an associated public key if using private-key JWT auth).
- **Token request:** When the external agent needs to actually perform an action (say retrieve some data or invoke a command via the MCP API), it uses its client credentials to obtain an **access token** from ACME's token endpoint. This could be a client credentials grant (if it acts as a machine client on behalf of an agreed service account), or possibly an authorization code grant if acting on behalf of an enterprise user (in which case a user would go through a consent prompt to authorize the agent's request).

- **Access enforcement:** The agent calls the enterprise MCP API, presenting the access token. The MCP server validates the token—since it trusts the enterprise's authorization server, it might check the token's signature (if JWT) or call an introspection endpoint to ensure the token is active and issued to the expected client.
- The MCP then executes the allowed action and returns the result to the external agent. The entire transaction is authorized under the token's scopes/roles, and logged. The enterprise can audit which external agent (client ID) accessed which resource at what time.

Below is a sequence diagram for Scenario 2:

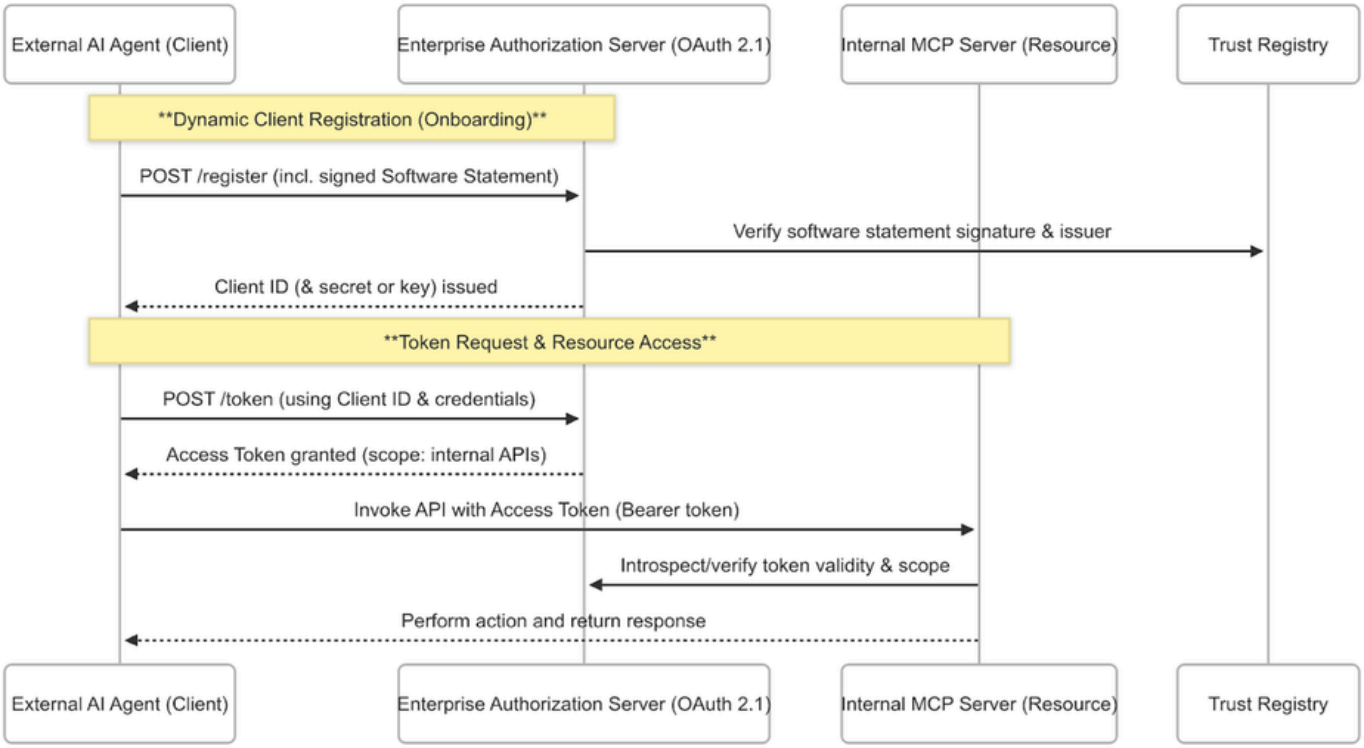

Diagram: Internal agent dynamically registers with external SaaS (possibly presenting a signed software statement to prove enterprise identity). The agent then performs the OAuth flow to get a token and call the SaaS MCP API securely.

This pattern mirrors the approaches used in open banking and data-sharing ecosystems. For instance, before Contoso's agent ever gets a token to access ACME's data, it had to prove its identity and authorization via the registration step. If Contoso was not in ACME's trust registry, the registration would be rejected outright. Even after registration, the access token issued can be constrained – e.g., ACME might issue a token that only allows read-only actions, with an audience restricted to specific APIs. Everything is cryptographically verified and auditable.

Real-World Analogies: Open Banking & CDR

To reinforce the concepts, it's useful to compare with analogous architectures in finance, where third-party integrations are common and highly regulated:

Open Banking UK

Banks (resource servers) expose APIs to third-party apps (clients) under strict security. The OpenBanking Directory acts as a **trust registry**—only regulated apps get listed and receive digital certificates and software statements. During onboarding, a third-party app uses a software statement signed by OpenBanking Ltd. to dynamically register with a bank's authorization server. The bank verifies the statement using the directory's keys and only then issues OAuth client credentials. All OAuth requests between the app and bank must use mutual TLS with the directory-issued cert, ensuring the caller is an approved party. The outcome is an ecosystem where **any bank can trust any registered app** without one-off arrangements, because the common trust registry (directory) vouches for them. This is very similar to our scenario 2, just industry-wide.

Australia Consumer Data Right (CDR)

Likewise, data holder companies (banks, energy providers, etc.) integrate with accredited data recipient apps. The central CDR Register is the trust authority. It issues a short-lived **Software Statement Assertion (SSA)** for each app. Data holders require this JWT in the DCR call and will not register a client without a valid SSA. The SSA includes the app's metadata and is signed by the Register's private key (using a strong algorithm like PS256). The data holder fetches the Register's JWKS (public keys) to verify the signature and ensure the SSA hasn't expired. If all checks pass, the app gets onboarded and can then request tokens to access consumer data (with the consumer's consent). The trust registry concept here is embodied by the CDR Register plus the legal accreditation behind it.

These real systems prove that combining OAuth with a trust framework scales to large ecosystems. They address the same fundamental challenge we have with SaaS AI integration: how to let in external code to sensitive APIs, or let internal code call out, without giving up security or control. The solution pattern is clear—**establish mutual trust via a registry and use OAuth flows for delegated access.**

Benefits: Security Without Sacrificing Agility

By applying the External SaaS Pattern with OAuth 2.1 and trust-managed DCR, an enterprise can achieve:

Agile onboarding of AI partnerships—new AI agents or services can be integrated in minutes rather than weeks, thanks to standardized registration and token exchange. (No hard-coding credentials or custom one-off auth mechanisms.)

Maintained Zero Trust posture: Even though we open our systems to external calls, every interaction is authenticated, authorized, and scoped. Access tokens carry fine-grained permissions and are short-lived, reducing risk.

Centralized oversight: The Trust Registry gives a single pane of glass to manage who/what is allowed. Administrators can review, approve, or revoke partner access in one place, and those changes propagate to the OAuth layer in real-time.

Auditability and compliance: Each token issuance and API call leaves an audit trail (client ID, user, scopes, time). This is critical for compliance and for investigating any anomalies. It also allows setting up alerts—e.g. if an external agent is suddenly requesting unusual scopes, you can detect and intervene.

Privacy protection: Users or data owners can be looped into the authorization process. If an AI agent wants to act on a user's behalf, standard OAuth consent screens can be presented to inform the user and get explicit approval. The user grants only the access needed and can revoke it later via a portal. This keeps transparency in how external AI is accessing internal data.

Importantly, all of this is achieved in a **vendor-neutral** fashion. We have not tied the design to any specific product—OAuth and JWT standards ensure that whether you use Azure AD, AWS, SecureAuth, Ping Identity, or an open-source OAuth server, the pattern can be implemented. The trust registry could be a simple database or a full-fledged directory service depending on scale, but the concept remains the same.

Note: In this article, we focused on explicit trust lists and signed statements to federate identity of clients. In the next part of this series, we will explore emerging standards like **OpenID Connect Federation** for automating these trust relationships. That will further reduce manual coordination by using hierarchical trust and signed metadata. But even without full federation, the approach outlined above is a powerful way to securely integrate SaaS AI today.

Conclusion

Inviting a SaaS-hosted AI agent into your enterprise (or vice versa) need not equate to inviting chaos. By leveraging OAuth 2.1 with Dynamic Client Registration—fortified by a Trust Registry and signed software statements—software architects can create a secure handshake between internal systems and external AI services. We can allow the innovation of agentic AI across organizational boundaries without sacrificing control or oversight. The External SaaS Pattern thus enables a careful balance: frictionless integration for trusted partners, but significant roadblocks for anything untrusted. In an era of rapidly proliferating AI capabilities, this pattern ensures your enterprise can say "yes" to SaaS AI—on your terms, with security, privacy, and accountability built in.

About SecureAuth

More security shouldn't mean more obstacles. Since 2005, SecureAuth has helped leading companies simplify identity and access management for customers and employees—creating experiences that are as welcoming as they are secure.

SecureAuth is redefining authentication for the modern enterprise. Today's evolving threat landscape demands innovative, adaptive security solutions. As the first-to-market provider of continuous facial authentication, we go beyond the initial authentication to deliver ongoing security throughout the entire session. Our mature AI-driven risk engine delivers dynamic—and often invisible—authentication, making you more effective than ever at eliminating threats while ensuring frictionless, secure access for employees and customers.

Welcome to Better Identity.

SECUREAUTH