Don’t Give Your Agent The Master Key!

Ask permission, not forgiveness.

We’re all racing to make agents smarter.

What decides whether any of this ships isn't intelligence.

It’s Identity!

Most people treat identity as the thing that slows agents down.

You can't hand an agent more autonomy until you can constrain it.

The constraint is the enabler.

The defining challenge of the agent era isn't making agents smarter.

It's making them trustworthy.

Who is this agent?

What did the human consent to?

What is this agent authorized to do?

What has this agent done before?

Can the agent prove all of this on every call?

Who is this agent?

What did the human consent to?

What is this agent authorized to do?

What has this agent done before?

Can the agent prove all of this on every call?

Who is this agent?

What did the human consent to?

What is this agent authorized to do?

What has this agent done before?

Can the agent prove all of this on every call?

Who is this agent?

What did the human consent to?

What is this agent authorized to do?

What has this agent done before?

Can the agent prove all of this on every call?

Who is this agent?

What did the human consent to?

What is this agent authorized to do?

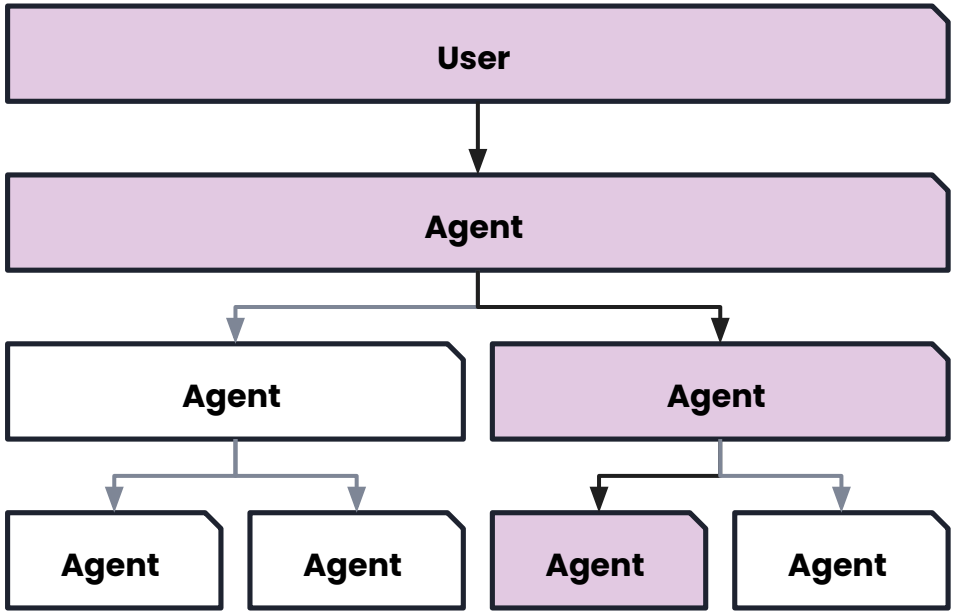
What has this agent done before?

Can the agent prove all of this on every call?

Our Vision

Every action traces back to a human decision.

Scoped. Time-bound. Revocable.



API Keys

Fine in a demo, a breach with a countdown timer in production!

```
// tools/send-email.ts

const sendEmail = tool({
  description: 'Send an email on behalf of the user',
  parameters: z.object({ to, subject, body }),
  execute: async ({ to, subject, body }) => {
    return fetch('https://gmail.googleapis.com/…', {
      headers: {

        // one key. all users. all scopes. forever.
        X-API-Key: process.env.GMAIL_API_KEY`,
      }, method: 'POST', body: JSON.stringify({ to, subject, body}),
    });
  },
});
```

x	Leaked key	→	Attacker gets everything the agent had
x	No user context	→	API keys authorize applications, not users
x	Over-scoped token	→	Prompt injection becomes data exfiltration
x	No human gate	→	Agent does a sensitive action nobody approved

Agents have their own identity.

They act for users, on the record.

Token Vault

Routine tool calls with user-scoped credentials.

Client-Initiated Backchannel Auth

High-stakes actions that need explicit human approval.

Agents as Principals

A first-class identity for the agent itself.

On-Behalf-Of Authentication

Delegated tokens that record who acted, for whom.

01

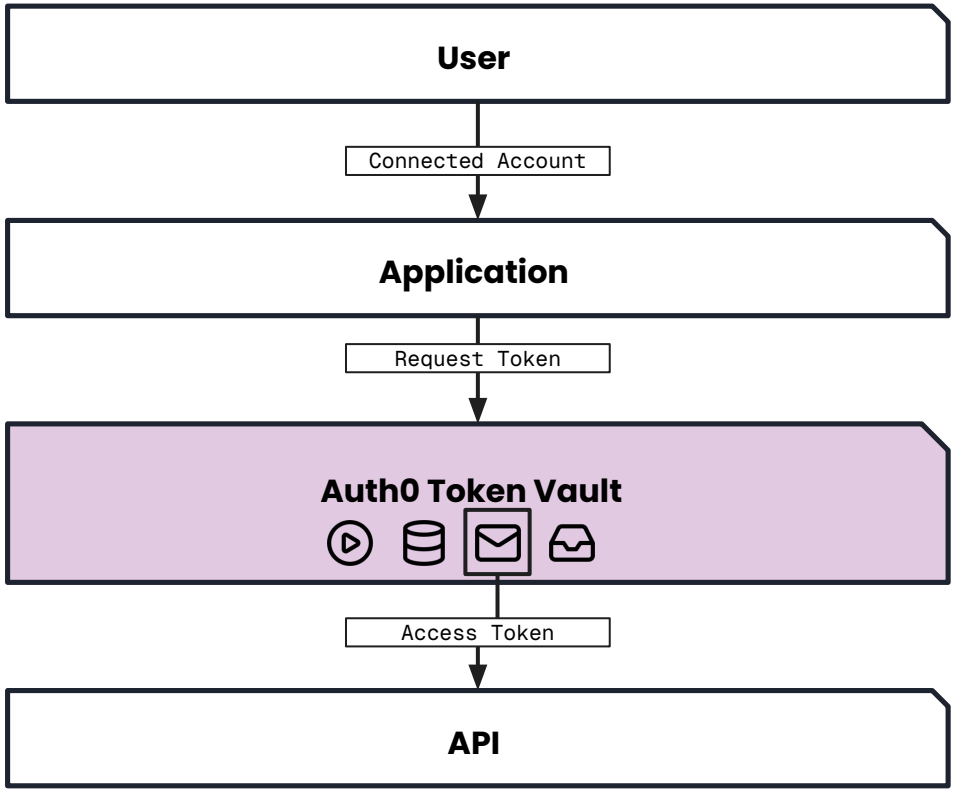
Token Vault

Routine tool calls with user-scoped credentials

Token Vault

One refresh token, stored securely. Short-lived, scoped tokens issued per call.

The agent never sees a master key.



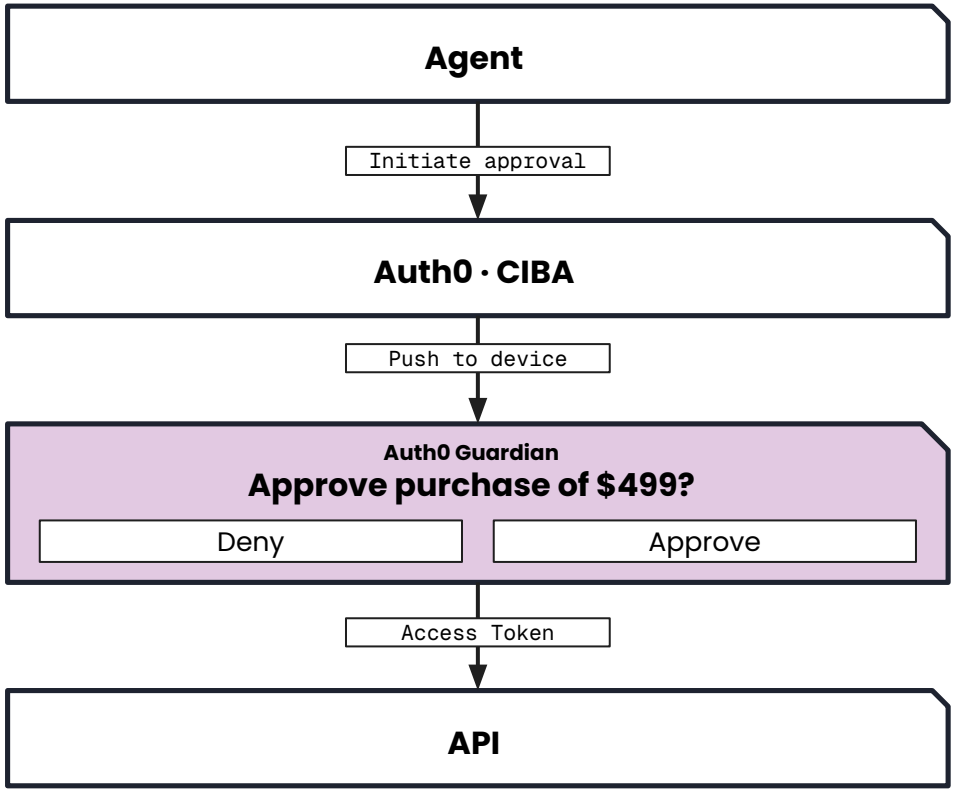
02

Client-Initiated Backchannel Authorization

High-Stakes actions that need explicit human approval

CIBA

The agent never decides. The user does, out-of-band, on a trusted device.



CIBA

`bindingMessage`, the exact text shown on the user's phone.

`execute` only runs after explicit approval.

The developer decides which tools need Client-Initiated Backchannel Authorization. The agent can't bypass it.

```
// tools/make-purchase.ts

const makePurchase = withAsyncAuthorization(
  { bindingMessage: ({ itemId, amount }) =>
      `Approve purchase of ${itemId} for ${amount}?`,
    scopes: ['purchases:write'],
    audience: 'https://api.example.com' },
    tool({
    parameters: z.object({ itemId, amount }),
    execute: async ({ itemId, amount }) => {
      const credentials = getAsyncAuthorizationCredentials();
      const accessToken = credentials?.accessToken;

      // only runs after the user approves on their phone
      return fetch('/purchases', { … });
    },
  }),
);
```

03

Agents as Principals

A first-class identity for the agent itself.

Agents as Principals

A first-class identity for the agent itself.

User	M2M client	Agent
`auth0|user-sam`	`client_id_abc`	`agt_knox123`
`SUB_PROFILE: USER`	`SUB_PROFILE: SERVICE`	`SUB_PROFILE: AI_AGENT`

Agents as Principals

`sub` and `sub_profile` : the user is still the subject of the token.

act is the actor chain. `agt_knox123` carried it out as an `ai_agent`.

It nests all the way down: agent, then browser app. Every hop in the delegation is on the receipt.

```
// the agent's access token, decoded

{
  "sub": "auth0|user-sam",
  "sub_profile": "user",
  "act": {
    "sub": "agt_knox",
    "sub_profile": "ai_agent",
    "act": {
      "sub": "knox_service",
      "sub_profile": "service"
    }
  }
}
```

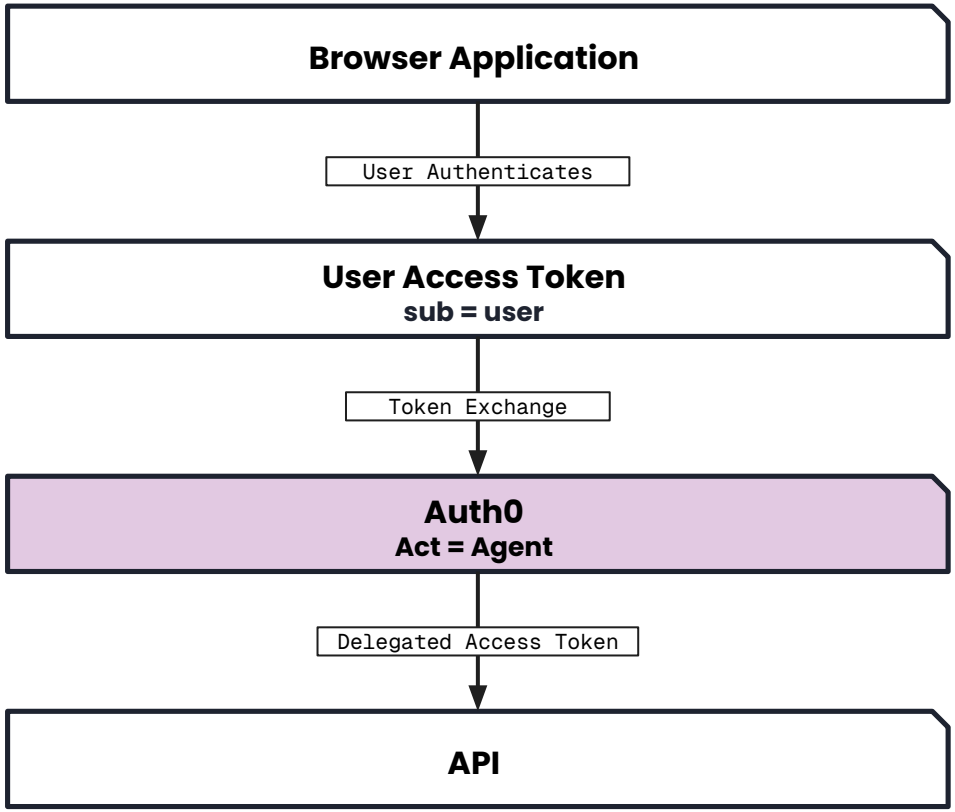
04

On-Behalf-Of Authentication

Delegated tokens that record who acted, for whom.

OBO

One token, multiple identities. The user is still the subject. The agent is recorded as the actor.



OBO

The audience, is the agent's resource server. The issued token is scoped to it.

The result carries an `act` claim: the cryptographic receipt.

```
// lib/agent-token.ts

const res = await fetch(`https://${DOMAIN}/oauth/token`, {
  method: 'POST',
  body: new URLSearchParams({
    grant_type: 'urn:ietf:params:oauth:grant-type:token-exchange',
    subject_token: userAccessToken,
    subject_token_type: 'urn:...:oauth:token-type:access_token',
    audience: https://agent.example.com,
    client_id, client_secret,
  }),
});
```

Four moving parts,
One secure agent.

Token Vault
`withTokenVault()`

Scoped, user-bound credentials

CIBA
`withAsyncAuthorization()`

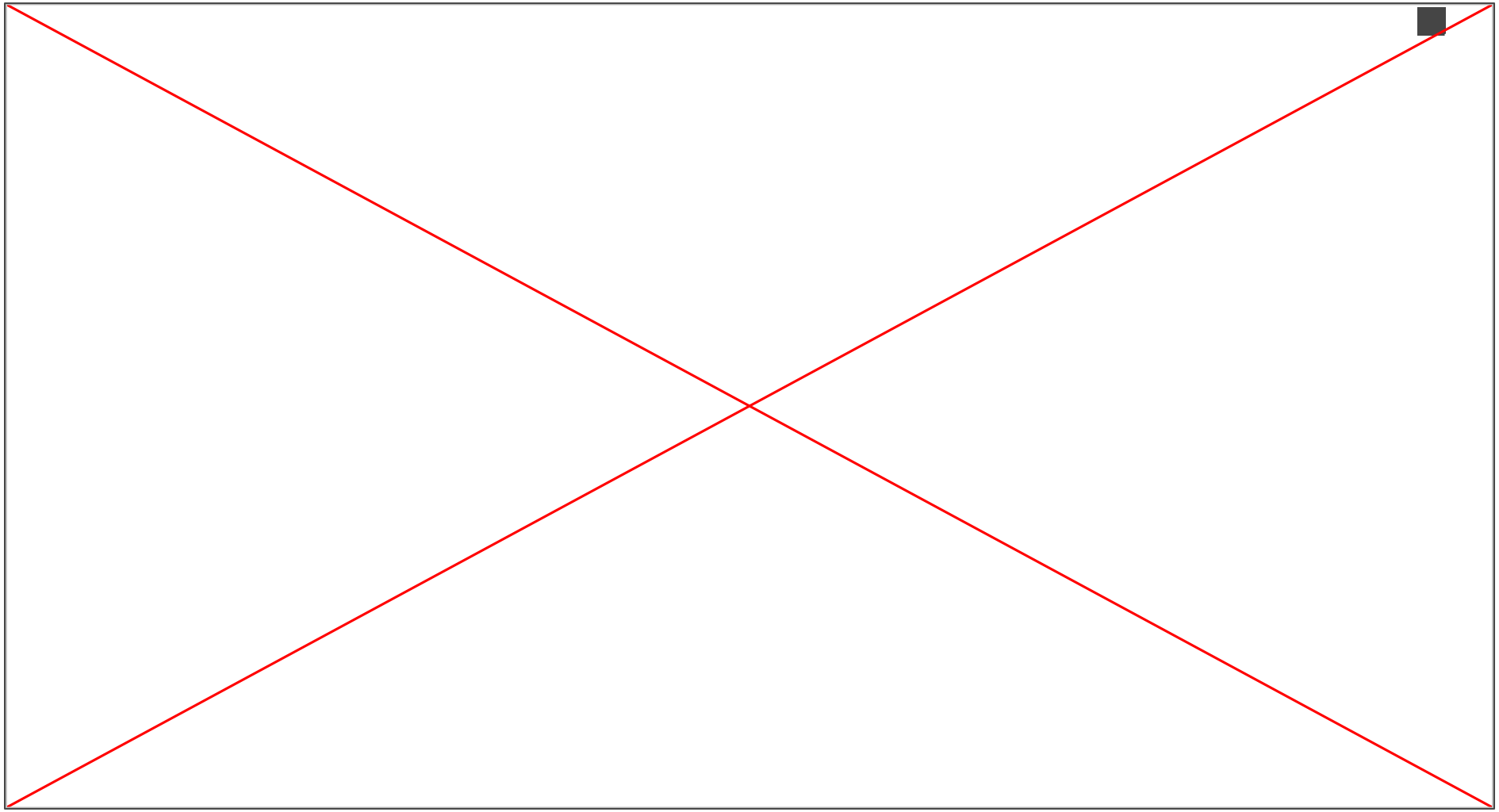
Explicit human approval

Agents as Principals
`sub_profile: ai_agent`

Who is acting

On-Behalf-Of Authentication
`act: { sub: agt_… }`

On whose behalf



Recap

- No master keys
- Per-user, per-tool, scoped, short-lived tokens
- Human-in-the-loop where it matters
- Agent has its own identity
- Every call carries the full delegation chain
- Standard OAuth / OIDC

Resources.

Auth0 AI Docs	`auth0.com/ai`
Demo	`github.com/sambego/ship-26`

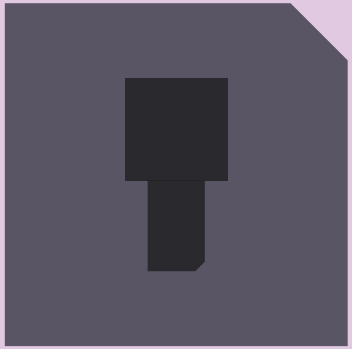
Sam Bellen

Principal Developer Advocate at Auth0

@sambego · sambego.tech

Slides are available

slides.sambego.tech/masterkey



Thank you.