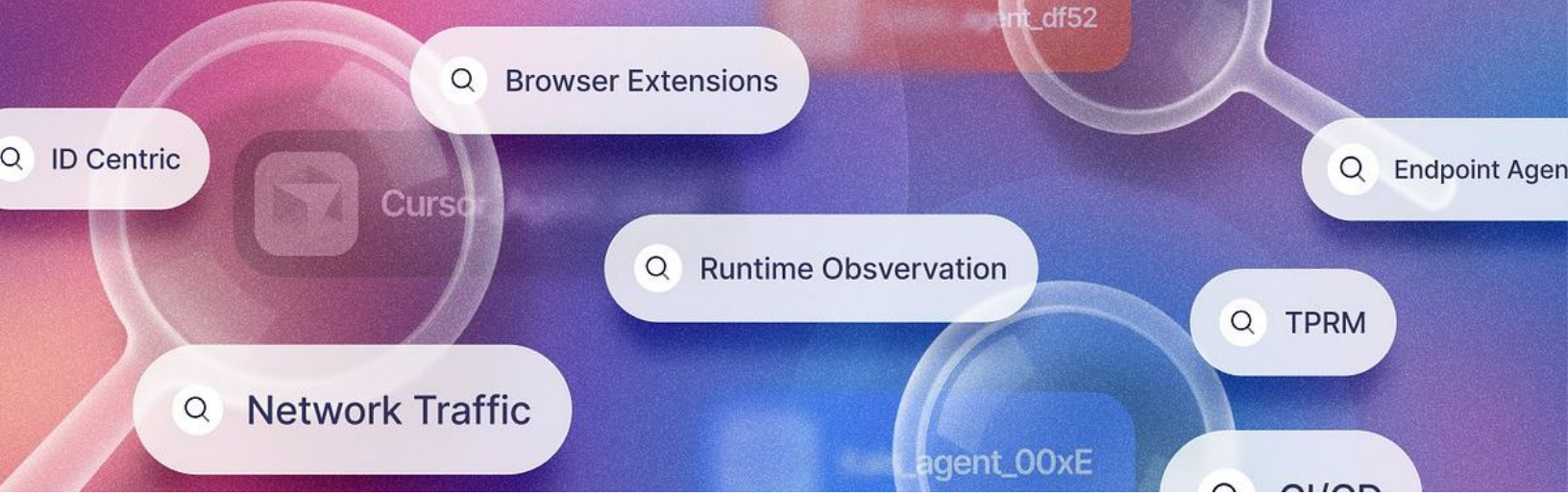

Practitioner's Guide to Shadow AI Agent Discovery

Your guide to finding, securing, and governing the agentic workforce.

Agentic AI is driving experimentation across every industry. Every employee can now build agents inside familiar tools like Microsoft Copilot Studio, Salesforce Agentforce, ServiceNow, and automation platforms like N8N and Workato; and grant those agents broad, persistent access to business data and actions.

In this guide, we'll compare various approaches to shadow AI agent discovery, how they work, and where blindspots remain. You'll learn:

- How to think about AI agent types and discovery surfaces
- How to compare common agent discovery methods including:
  - Network traffic analysis
  - Browser extensions
  - Endpoint agents
  - CI/CD and source code scanning
  - Runtime and observability-based discovery
  - TPRM solutions or processes
  - Identity-centric discovery
  - SaaS API/control plane discovery
- Critical questions to ask of AI security vendors

What is an AI agent, anyway?

This sounds like a trick question, but it's fundamental. Agents can describe a wide range of systems with varying levels of autonomy, from generative AI assistants that simply recommend actions a human user can take, to systems that execute complex, multi-step workflows. Agent-washing makes it even harder to distinguish traditional automation workflows from true agentic capabilities.

A working definition

An AI agent is a software system that uses an AI model to determine its next action toward a goal and takes autonomous or semi-autonomous actions through tools or APIs to execute multi-step tasks, adjusting its approach based on results and new context. Agents can be embedded in SaaS and agentic AI platforms, run locally on machines, or operate entirely through APIs.

The business appeal is obvious. AI agents promise to automate repetitive work, accelerate development cycles, and extend what small teams can accomplish. But unlike a SaaS app that sits behind an SSO login, agents actively *do* things independently: they make API calls, store and retrieve data, take actions in connected systems, and sometimes spawn other agents to get subtasks done.

Why it matters for security: The security implications of an unknown, unmanaged agent are considerably higher than those of an unknown SaaS app. Agents don't just sit there. They act, and they act with whatever permissions they've been granted.

The result: a fast-growing blind spot

The result is a fast-growing blind spot: shadow agents that IT and security often do not know exist, but that can read, move, and act on sensitive data. Agentic AI security solutions have emerged recently to discover shadow AI agents, yet it can be difficult to compare offerings in such a frenetic marketplace.

A seemingly simple question becomes difficult to answer: **how do you actually discover shadow AI agents in your environment?** The short answer is that no single discovery method or tool is perfect. Each technical approach reveals a fundamentally different picture of what agents exist and where, what they can access, who controls them, and what they're doing.

Tip: Look for discovery solutions that surface not just agent existence, but ownership, permissions, and lifecycle status. Detection without governance context is just an alert.

What should agent discovery include?

AI agent discovery should be scoped to include agents that have been formally procured and deployed by IT as well as shadow agents—the growing wave of agents that employees and developers spin up without going through any formal review or approval process.

Much like shadow AI discovery and SaaS discovery, the purpose of AI agent discovery is to build a comprehensive and continuously updated inventory of every agent in your environment: what it is, who owns it, what systems it connects to, what permissions it holds, and what data it can access or act on.

Tip: Effective agent discovery must be ongoing, not a one-time audit. Shadow agents appear continuously as employees experiment with platforms like Copilot Studio, Agentforce, and N8N.

Effective agent discovery helps you answer:

- What agents exist in our environment?
- Who created them, and who is accountable for them?
- What systems and data can they access?
- Which agents are risky (public exposure, hardcoded secrets, excessive permissions)?
- Which agents belong to creators who have left the company?

These answers are fundamental to agentic AI security and governance.

Tip: Discovery without attribution is limited. Every discovered agent should be mapped to a human owner—that's what makes governance actionable rather than informational.

Why is agent discovery important?

Agents introduce a category of security risk that is meaningfully different from standard SaaS risk. Instead of passive tools your employees log into, they're active participants in your environment that take actions, make decisions, and interact with sensitive systems on an ongoing basis. Here's why getting a handle on them matters:

Data exposure and exfiltration risk

Agents with broad access to files, email, or databases can exfiltrate sensitive data intentionally (by a malicious actor) or accidentally (by a poorly configured prompt). Without discovery, you won't know which agents hold this access or whether it's appropriate.

Non-human identity proliferation

Every agent needs credentials to operate: API keys, OAuth tokens, service accounts. These non-human identities are notoriously difficult to track and are a common vector for credential abuse and privilege escalation.

Compliance requirements

Frameworks like SOC 2, HIPAA, GDPR, and the EU AI Act increasingly require organizations to document automated decision-making systems, data processors, and AI tools that handle regulated data. You can't document what you haven't found.

Prompt injection and model manipulation

Agents that consume external content are vulnerable to prompt injection attacks, where malicious instructions are embedded in that content to hijack agent behavior. Discovery is a prerequisite for assessing and mitigating this exposure.

Orphaned agent access

Agents deployed by employees who leave the organization can continue to operate with active credentials long after departure—especially if they were never registered anywhere to begin with.

Supply chain risk

Many agents depend on third-party LLM providers, cloud-hosted models, or external APIs. A compromise upstream can have downstream consequences for your data and workflows.

Compliance requirements:

Frameworks like SOC 2, HIPAA, GDPR, and the EU AI Act increasinly require organizations to document automated decision-making systems, data processors, and AI tools that handle regulated data. You can't document what you haven't found.

The Bottom Line

Agents need the same rigor that IT and security teams have (slowly, painfully) built around human users. Discovery is where that work begins. An unknown agent isn't just an ungoverned tool—it's an ungoverned actor with persistent access to your most sensitive systems.

Understanding different types of AI agents

Discovery method selection is driven by where an agent lives and how it was built. Different agent types inhabit fundamentally different surfaces, and the method that finds one type is often completely blind to another.

At a high level, agents can run across a wide range of environments:

Type of agent	Where it can run
Browser-native agents	Any desktop or mobile browser
Browser extension agents	Desktop browsers with the extension installed
IDE / CLI agents	Local IDE (VS Code, Cursor, JetBrains); local command line/terminal; cloud-based dev environments (GitHub Codespaces)
Embedded agents	A vendor's SaaS platform, hosted in their cloud and accessed through the web interface; locally installed desktop apps; mobile apps
Workflow agents	Third-party automation platform clouds (Zapier, Make.com); self-hosted servers (n8n); triggered on schedule or by event
Custom-built agents	Local machine; on-premises server; org's own cloud (AWS Bedrock, Google Vertex AI, Azure AI Foundry); containerized or server-less environments
Hybrid agents	Multiple environments by design—the same agent codebase can run in an IDE, a CLI, or a cloud workflow

Heavily invested in Microsoft 365, Salesforce, or ServiceNow?

Embedded agent features are almost certainly already active whether IT provisioned them intentionally or not.

Active software development teams building on AI APIs?

Custom-built agents are likely already being created, often outside any formal IT process.

Empowered business users who adopt cloud tools without IT?

Shadow agents are likely in use across a variety of SaaS apps and automation tools, many of which don't expose agents through their APIs.

All of the above?

Most organizations need to run several discovery methods in parallel since agents exist across all of these surfaces at once.

AI Agent Discovery Methods Explained

1 Method 1: Network traffic analysis (including SASE/SSE)

Network-based methods detect agents by inspecting traffic between your environment and external AI services. SASE/SSE vendors have a natural adjacency here because they're already inline for access decisions. The agent signal is traffic flowing to an AI service endpoint—destination domains, DNS patterns, volume, and timing. Network analysis can detect anomalous data flows and flag potential exfiltration paths. It can enforce policy at the network boundary through blocking or filtering.

While network analysis can tell you that *something* in your environment called an AI service, it can't distinguish an autonomous agent from a human using a chatbot. It can't tell you who owns the agent, what permissions it holds, what data it accesses, or what tools it invokes. Additionally, most agent traffic runs over encrypted HTTPS, so payloads are opaque without decryption keys. It can't tell you the semantic meaning of the data in transit, the internal application context for why data was accessed, or the full permission model of the agent that generated the traffic.

Keep in mind that agents that operate locally, server-side, or offline generate no network traffic at all.

Agent signal:

Traffic flowed to an AI service endpoint. Destination domains, DNS patterns, volume, and timing.

Strengths:

- Can surface active agents making external calls even if never formally registered or reported to IT
- No per-app integrations required; works across agent types making observable external calls
- Can catch rogue or unauthorized agent behavior in transit

Limitations:

- Can't see dormant agents or agents operating entirely within internal systems
- Can't distinguish an agent from a human using a chatbot
- Doesn't reveal who owns an agent, what it's authorized to do, or why it accesses what it accesses

Where it fits

Catching previously unknown connections to AI service providers. Best suited as an early warning signal ("something is talking to Anthropic's API") rather than a source of agent-level governance context.

2 Method 2: Browser extensions

Browser extensions observe web-based activity: URLs visited, pages loaded, session timing. In the context of agents, they can catch employees interacting with web-based agent builders or running browser-based AI assistants. More capable implementations go further: the browser extension can capture agent-level data at the moment an employee creates or lists an agent on a supported platform, including creator, components, permissions, and risk signals, without requiring an API integration into that platform.

Browser-based discovery supports user-level governance, including acceptable use policies, data loss prevention rules, and detection of risky behaviors like pasting sensitive data into AI tools. It catches early-stage shadow AI exposure. But it has no visibility into backend or system-to-system data flows, data accessed by autonomous agents without user interaction, or full permission and infrastructure context.

This is the most practical way to discover agents on platforms that don't expose agent details via public APIs, which is where most shadow agent activity actually lives.

Agent signal:

A user visited or interacted with a web-based AI agent tool.

Strengths:

- Fast path to detecting shadow agents before they become formal deployments
- Works without requiring API integrations into the platforms being used
- Some extensions can capture prompt-level detail

Limitations:

- Only covers managed browsers on managed devices
- Blind to desktop agents, mobile agents, API-based agents, and server-side SaaS agents
- Can't reveal what an agent is authorized to do, what data it accesses, or how it connects to other systems

Where it fits:

Detecting employees using web-based agent builders like Copilot Studio, Agentforce, or similar no-code platforms through a browser. Particularly valuable for surfacing shadow AI agents on platforms that don't expose public APIs, which is the category of agent risk that most discovery tools miss entirely.

3 Method 3: Endpoint agents

Endpoint agents run on managed workstations and monitor local application behavior, including AI-related desktop apps, IDE extensions, and local model execution. This method shows you which local AI tools are installed and running, and on whose machine and catches agent development activity (ex, an engineer building agents via Cursor, VS Code extensions, or local frameworks like LangGraph) that browser extensions and network tools miss. But endpoint agents can't see server-side agents, SaaS-embedded agents, or agents in cloud environments. They also can't tell you what permissions the agent holds in downstream systems.

Endpoint discovery gives you visibility into the supply side of shadow agent adoption: who is building what, on which devices. But it's limited to managed devices and provides no insight into what those agents do once they leave the developer's machine.

Agent signal:

AI-related software or agent tooling is running on a managed device.

Strengths:

- Catches agent development and adoption activity that network tools and browser extensions miss
- Surfaces the supply side of shadow AI: who is building what, on which devices
- Integrates with existing endpoint management infrastructure

Limitations:

- Only covers managed devices; bypassed entirely on personal or unmanaged hardware
- Can't see server-side agents, SaaS-embedded agents, or anything running in the cloud
- No visibility into permissions or what agents do once deployed

Where it fits:

Catching local agent development and experimentation on managed devices. Important for understanding the origin of agents, but not their operational footprint.

4 Method 4: CI/CD and source code scanning

These tools scan code repositories and build pipelines for agent framework references, SDK imports, orchestration code, and API calls to AI services. They can identify which agent frameworks developers are using (LangGraph, CrewAI, AutoGen), which AI services they're calling, and where agent logic lives in the code. When extended to cloud platform scanning, these tools can also identify agent workloads deployed on platforms like Bedrock, Azure AI, or Vertex AI.

But code scanning tells you about agent *design*, not agent *behavior*. It can't reveal what permissions an agent has been granted in production, what data it actually accesses at runtime, or who authorized its deployment. Agents built outside the pipeline through no-code tools or SaaS-native builders are invisible.

Code scanning supports AI Bill of Materials (AIBOM) efforts and supply chain governance. It answers "what did developers intend to build?" But it can't confirm whether production behavior matches design, whether permissions have drifted, or who is accountable for the running agent.

Agent signal:

Agent-related code, frameworks, or model references exist in the codebase.

Strengths:

- Finds agents before they reach production, while they're still being built
- Supports AI Bill of Materials (AIBOM) and supply chain governance efforts
- Identifies which AI services and frameworks your development teams depend on

Limitations:

- Shows design intent, not production behavior; what's in the code may not match what's deployed
- Completely blind to no-code and low-code agents built in SaaS platforms
- Can't reveal runtime permissions, actual data access, or who authorized deployment

Where it fits:

Inventorying developer-built agents during development, before and during deployment. Pairs well with AIBOM efforts but doesn't replace runtime or identity-based discovery.

5 Method 5: Runtime and observability-based discovery

If you want to understand what agents are actually doing, you have to move closer to execution.

Runtime-based discovery captures activity at the point where agents interact with models, call tools, and execute workflows. This is typically done through inline proxies, SDK instrumentation, or integrations with observability pipelines.

This is the most direct way to see agent behavior. You can observe what data an agent retrieves, processes, or generates. You can see prompts and outputs involving sensitive data. You can trace tool-level data access ("queried CRM," "wrote to database," etc) and detect anomalies, misuse, or policy violations in real time. This is where discovery shifts from inference to reality.

Runtime observability shows whether policies are actually being enforced during execution, meaning PII redaction, prompt filtering, access controls. It detects misuse like data exfiltration or prompt injection impacts. It answers the question that most other methods can't: is the agent doing what it's supposed to?

But, it can't provide a full inventory of all possible data access outside instrumented environments, or context on ownership and permissions without correlation to identity data. And it only covers what's been instrumented, which means anything running outside observed surfaces is invisible.

Agent signal:

Prompts, responses, tool calls, execution chains, and decision flows captured at the point of execution.

Strengths:

- Deepest visibility into what agents actually do, not just that they exist
- Best method for verifying whether an agent is behaving as intended
- Can surface misuse and policy violations that no inventory-based method would catch

Limitations:

- Only covers what's been instrumented; agents outside instrumented environments are invisible
- Requires upfront setup: SDK integration, proxy deployment, or observability pipeline configuration
- Comes with performance and privacy tradeoffs that require organizational buy-in

Where it fits:

Essential when you have real agent runtime risk concerns and need step-level accountability. Best paired with identity and control plane data to close inventory and attribution gaps.

Method 6: Third-party risk management (TPRM)

TPRM systems rely on vendors to self-report their AI agent capabilities through questionnaires and contract disclosures. The agent signal is simply: a vendor claims their product includes agent functionality and discloses details like agent capabilities, data handling terms, and contractual commitments.

TPRM creates contractual accountability for third-party agents and supports supply chain governance at the policy and documentation layer. But it can't verify those claims, can't detect agents embedded without disclosure, and can't identify agents that vendors have added between questionnaire cycles.

Agent signal:

A vendor claims their product includes agent functionality.

Strengths:

- Creates contractual accountability for third-party agents
- Supports supply chain governance at the policy and documentation layer
- Establishes a paper trail for compliance and audit purposes

Limitations:

- Entirely self-reported; there's no way to verify vendor claims through this method
- Point-in-time by nature; agents added between assessment cycles go undetected
- No visibility into how third-party agents actually behave in your environment

Where it fits:

Contractual accountability for third-party agents. Not a substitute for technical discovery, but an important complement for supply chain governance.

7 Method 7: Identity-centric discovery

Identity-centric discovery finds agents by examining the credentials they use to operate: OAuth grants, API keys, service accounts, delegated tokens, and MCP connections. Rather than observing agents indirectly through traffic patterns or code references, this approach discovers evidence of agents through the access infrastructure that makes their activity possible.

This is the method that most directly answers the questions agent governance requires. Specifically: who authorized the agent (human owner), what systems and data it can access (permission scope), whether those permissions are current, excessive, or orphaned, the delegation chains it participates in (where an agent inherits or passes credentials to other agents), and whether the agent has been properly deactivated when no longer needed.

Identity-centric discovery directly supports the lifecycle management that governance programs require: who created this, who approved it, does it still need this access, and is anyone still accountable for it?

Agent signal:

An agent-associated identity, such as an OAuth token, API key, service account, or MCP connection, exists with active or dormant access to enterprise systems.

Strengths:

- Finds hidden agents through the access infrastructure they depend on
- Maps blast radius: what could this agent access if compromised?
- Most directly answers governance questions: who authorized it, what can it reach, is it still needed

Limitations:

- Assumes agents use identifiable, managed credentials; misses agents running under shared or human identities
- Many agents embedded inside SaaS platforms never create their own credentials and are invisible here
- Can't confirm actual agent behavior or whether access is used appropriately

Where it fits:

The richest source of agent governance context for building an agent catalog, including owner, permissions, access scope, and lifecycle status.

8 Method 8: SaaS API / Control plane discovery

This method bridges it bridges two things that are often discussed independently: agent platform integrations and identity-centric discovery.

There's an important distinction between two types of platforms:

Dedicated agent development platforms such as Copilot Studio, Agentforce, Bedrock Agents, Vertex AI Agent Builder, and ServiceNow Now Assist Studio are purpose-built for creating and deploying agents. Integrating with their APIs gives you agent configuration, tool access, and platform-specific metadata

SaaS applications with agent capabilities are a different problem. Platforms like Salesforce, ServiceNow, and Microsoft 365 are business applications that now support agent creation and operation within them. Agents embedded in these platforms can directly change systems and data, so clearly mapping their permissions and operational scope matters.

SaaS platforms expose OAuth grants, third-party app integrations, API connector configurations, and service account activity through their APIs. That means querying SaaS APIs can reveal things like: an OAuth grant giving a third-party AI tool access to your SharePoint sites, a sync connector copying M365 data to an external AI service's vector database, an Agentforce agent with CRM access created by a business user without security review, or a Copilot Studio agent with broad permissions that was never offboarded after its creator left.

This is the method that ties together "what exists" and "who authorized it and what can it do." Through SaaS APIs, you can identify not just that an agent-related integration exists, but the human user who authorized the OAuth grant, the specific data scopes the integration can access, and whether the connection is actively syncing data to an external AI service. For agents built natively within SaaS platforms, the APIs expose agent configuration, tool access, and operational scope directly.

However, this method requires an API connection into every platform where you want to discover agents. It will miss agents that bypass standard OAuth or API-key patterns and shadow agents built entirely outside SaaS platforms: in custom code, local environments, or on unmanaged devices.

Agent signal:

OAuth grants, third-party app registrations, API connectors, service account configurations, and registered agents within SaaS platforms.

Strengths:

- Fast, broad reach, low friction
- Works well for enterprise-managed agents
- Provides rich metadata (permissions, configs)

Limitations:

- Misses shadow / local / custom agents
- Only sees what platforms expose
- Weak visibility into runtime behavior

Where it fits:

The connective tissue between identity-centric discovery and platform-specific agent discovery. SaaS APIs are where you find both the agents that platforms know about (registered agents) and the agent-associated identity signals that platforms expose whether or not anyone thought of them as "agents."

How to evaluate AI agent discovery solutions

When evaluating vendors, don't just ask whether or not a provider can discover AI agents. Instead, get more specific. As you've probably concluded, there's no magic bullet—each discovery approach has its strengths and blind spots, and the best vendors are explicit about both.

Method	Best for	Key blind spots
Network traffic analysis (SASE/SSE)	• Early warning of external AI service connections • Catching unauthorized agent traffic in transit	• Can't distinguish agent from human • No governance context • Blind to internal/offline agents
Browser extensions	• Shadow agents on web-based no-code platforms • Platforms without public APIs	• Managed browsers only • Blind to desktop, mobile, and server-side agents
Endpoint agents	• Local agent development on managed devices • Understanding who is building agents	• Managed devices only • No cloud/SaaS visibility • No runtime permissions
CI/CD and source code scanning	• Developer-built agents pre-production • AIBOM and supply chain governance	• No-code/low-code agents are invisible • Design intent only, not runtime behavior
Runtime observability	• Verifying agent behavior vs. intent • Detecting misuse and policy violations	• Only instrumented surfaces • Requires significant setup
Identity-centric discovery	• Governance: owner, permissions, lifecycle • Orphaned agents and excessive access	• Misses agents under shared or human identities • Embedded SaaS agents may be invisible
SaaS API / control plane discovery	• Enterprise-managed agents on known platforms • Connecting identity to agent config	• Shadow/local/custom agents are missed • Only sees what platforms expose
TPRM solutions or processes	• Contractual accountability for third-party agents • Compliance documentation	• Entirely self-reported • Point-in-time only • No technical verification

Critical questions to ask of all agent discovery vendors

To help with your evaluation, here's a condensed list of questions to ask any vendor about their shadow AI agent discovery capabilities:

1

Discovery Methods

1. What is your primary discovery mechanism—API, identity, runtime, network, endpoint, or browser?
2. What do you actually enumerate: agents, identities, tools, data flows, sessions?
3. What environments do you cover: SaaS, cloud, local, browser, code?

2

Governance Context

- What governance context do you provide for each discovered agent?
- Can you help answer these questions?
  - Who owns it and who created it?
  - What systems and data can it access?
  - Has it been approved? Is it still needed?
  - What happens when the creator leaves the organization?

3

Risk Insights

- What risk signals do you surface specifically for agents?
- Which of these risks can you discover?
  - Publicly accessible agents
  - Hardcoded credentials
  - Unauthenticated MCP connections
  - Agents still running after creator departure
  - Integrations to high-risk applications
- Do you provide risk remediation guidance or automation?
- Do you align your agentic risk findings with any compliance frameworks or industry standards? (OWASP Top 10 for Agentic AI for example)

4

Coverage Assessment

1. What do you miss? (Every vendor has blind spots, will they be explicit about them?)
2. Do you require new infrastructure or agents to deploy your solution?
3. How does coverage grow as new agentic platforms emerge?
   a. Will the platform require new detection rules? If so, how often do you add them?
   b. Will coverage for new agentic platforms require configuration updates?
   c. How often do you add support for new agentic platforms?

AI agent discovery with Nudge Security

Nudge Security's approach to **AI agent discovery** combines multiple methods, giving you the coverage and context to answer the questions security teams face first: what agents exist in your environment, who created them, what can they access, and is anyone still accountable for them?

Nudge Security's agent discovery capabilities serve as the foundation for a complete agentic AI security solution that helps organizations find, govern, and secure their agentic workforce without becoming a bottleneck for AI adoption.

Shadow agent discovery from multiple vantage points

API-based discovery continuously pulls agent data from platforms with public APIs. The browser extension covers platforms that don't expose APIs at all. Identity-centric signals surface evidence of agents through the credentials they use: OAuth grants, API keys, service accounts, and MCP connections.

Creator-level accountability

Every discovered agent is mapped to the human who built it. When a creator leaves your organization, you know. When something needs remediation, you know exactly who to nudge—making governance actionable rather than informational.

Risk insights built for agentic AI

Nudge Security surfaces the signals that matter for agents specifically: publicly accessible agents, hardcoded credentials, unauthenticated MCP connections, integrations to high-risk apps, and agents still running after their creators have left.

Discovery that connects directly to governance

Finding agents is only the starting point. From the same inventory, security teams can set approval status, assign a technical contact, and trigger automated nudges to creators to confirm intent and request remediation, closing the loop without becoming a bottleneck for AI adoption.

Coverage that grows without new infrastructure

Browser-based discovery uses the Nudge Security extension most customers already deploy for SaaS and AI governance. API-based discovery connects through existing app integrations. No new tools, endpoint agents, or network monitoring required.

Start a free 14-day trial today

> "Having Nudge has significantly brought peace of mind because I don't have to go looking for a needle in a haystack anymore."
>
> Leo C, IT Team member at GLAAD