redaccess

RED ACCESS RESEARCH / CATEGORY REPORT

The Shadow Builders Inside Your Organization.

A category report on Shadow AI — the people building it, the applications they've already deployed, and the layer most security stacks still miss. Plus: what to do about it, starting Monday morning.

May 2026
Vol. 01

redaccess.io
Agentless · Session-Layer · SSE

CONTENTS

Inside this report.

EXECUTIVE SUMMARY

A category-level exposure pattern.

Across the platforms we investigated most rigorously, Red Access identified a pattern that crosses every industry, every continent, and every company size we examined.

380K+

Publicly accessible web assets identified across leading vibe-coding platforms.

5,000

Appeared to be built for corporate purposes — internal tools, dashboards, workflows.

2,000

Of those (40%) contained sensitive corporate, operational, or personal data deployed without basic security controls.

THE THRESHOLD THAT MATTERS

None of this required exploitation. Everything documented in this report was reachable by anyone with a browser.

The shift.

In the past two years, a new category of software has quietly become a standard part of how enterprise work gets done. *Vibe coding* — the broader space of Generative AI development platforms where anyone can build a working application by describing what they want — has compressed what used to take engineering teams months into something a non-developer can do before lunch. The marketing manager building a campaign tracker, the operations lead automating a vendor workflow, the finance team prototyping a board-prep dashboard. None of this is unusual anymore. All of it is genuinely useful. None of it is going away.

This report is about the category's growing pains, and what security teams need to know — and do — while it grows.

The findings. The exposures spanned six continents and every industry examined. They included a live financial dashboard belonging to one of the largest banks in Latin America; medical personnel information from systems associated with a major national health organization; strategic documents of a $200 billion company exposed through a third-party vendor; and phishing infrastructure impersonating Bank of America, FedEx, and Costco, hosted on the legitimate domains of vibe-coding platforms.

The shift. This is the new shape of Shadow AI — and it is not the same Shadow AI the industry has been discussing for the past couple of years. The earlier conversation was about employees pasting sensitive data into AI-chat on personal accounts. The problem in this report is different in kind: employees building applications, connecting them to production systems, and deploying them publicly. While their IT is completely unaware. We call the people doing this *Shadow Builders*. They are not malicious. They are competent employees solving real problems faster than their organization could.

What this means for security teams. Three things, all of them present-tense.

01

The exposure is systemic, not incidental.

A category-level pattern across every industry, on every continent. Organizations are passing audits while these exposures are live.

02

The exposure is invisible to most existing security stacks.

EDR, DLP, CASB, and network firewalls each see fragments of the activity, but none is positioned to recognize, catalog, attribute, or govern the category. The visibility gap is structural, not a tooling failure.

03

The category is maturing — that is not the same as airtight.

Defaults are improving; some platforms are shipping private-by-default modes and clearer access controls. This is the right direction. It is also not, on its own, a remediation. Applications already deployed under earlier defaults remain live. The discovery, governance, and visibility work belongs to the security organization regardless of what any platform ships next.

The point of this report is to help that work. What follows describes what is happening, why traditional controls miss it, and — most importantly — what a CISO can do this week to find out whether their organization is exposed, and to start governing a category that is not waiting for permission to grow.

A READING NOTE

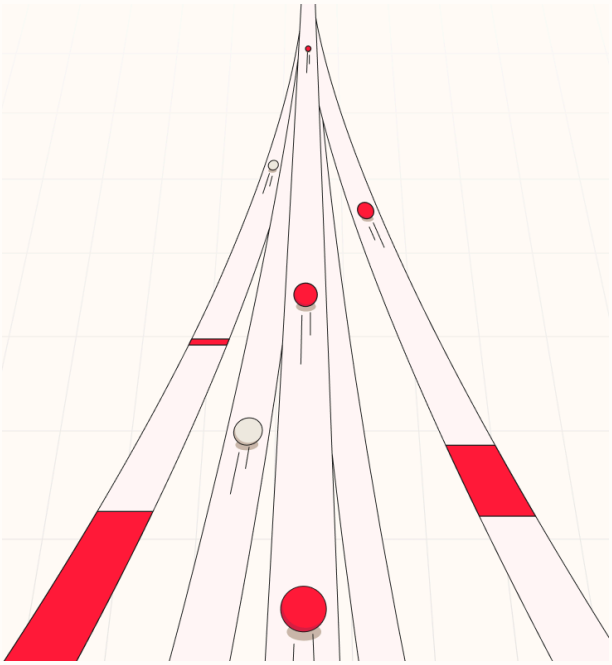
> **"The honest answer to who is creating this risk? is 'your colleagues, mostly.'**
>
> **The honest answer to what is this risk? is what the rest of this report is about."**

The five chapters that follow describe the category, document the findings, walk the build path that produces them, and explain why most existing controls miss it. The final three chapters are practical: a diagnostic, a checklist, and an architectural recommendation.

Chapter 01

Defining the Category.

The conversation moved from prompt to product. The risk surface moved with it.



FROM SHADOW IT → SHADOW AI

The conversation moved from prompt to product.

The term *Shadow AI* has been in circulation for roughly two years. Until recently, it meant something narrower than what this report describes. It meant employees using AI tools their employer hadn't sanctioned — pasting customer data into AI-chat, running confidential documents through unmanaged summarizers, asking Claude to draft a contract on a personal account. That problem is real, and it isn't going away. But it is not the problem documented here.

The Shadow AI surfaced in this research is a different category. It isn't about employees *talking to* AI. It's about employees *building with* AI — spinning up full applications, connecting them to production systems, and deploying them (often on the open internet), almost always without involving security, IT, or the rest of the organization. The conversation moved from prompt to product. The risk surface moved with it.

TERM 01

Shadow AI

The category of risk: the rapidly expanding population of AI-built applications operating outside enterprise governance, frequently connected to sanctioned systems, and rarely visible to the security team.

TERM 02

Shadow Builders

The people producing them: the marketing manager prototyping a campaign tracker, the operations lead automating a vendor workflow, the HR coordinator building a candidate intake form. They are not rogue actors. They are competent employees doing exactly what the platforms invited them to do.

The platforms, for their part, are doing exactly what their original audience asked for: making it possible for anyone to build an application by describing what they want, in minutes, with no engineering ticket and no procurement cycle. The fact that this is a genuinely useful capability is part of what makes the resulting risk category serious. **People are using vibe coding because vibe coding works.**

Shadow AI inverts Shadow IT.

What hasn't fully caught up is the guardrails — both technical and behavioral — governing what happens after the build. A Shadow Builder is rarely thinking from a security point of view. They're making productivity decisions. In the cases documented in this report, that frequently meant publicly accessible URLs, no authentication, and admin access granted to whoever opened the page. A finance manager building a board-prep dashboard on a vibe-coding platform is focused on shipping something by Friday. Security choices are secondary, at best — and often invisible inside the flow.

This is what separates Shadow AI from Shadow IT. **Shadow IT was about employees adopting unsanctioned SaaS** — a marketing team buying a Trello account on a corporate card without telling IT. The risk was real but bounded; the data stayed inside the SaaS vendor, and identity, governance, and audit logs were at least available, even if the security team didn't have access to them.

Shadow AI inverts that. The application is custom-built, the data is custom-loaded, the integrations are direct connections to production CRMs, ERPs, ticketing systems, and BI tools — and the artifact is very often published on the open internet. The platform underneath may be audited; the application built on it isn't. There is the builder, the platform, and the URL. IT? Mostly not in the room.

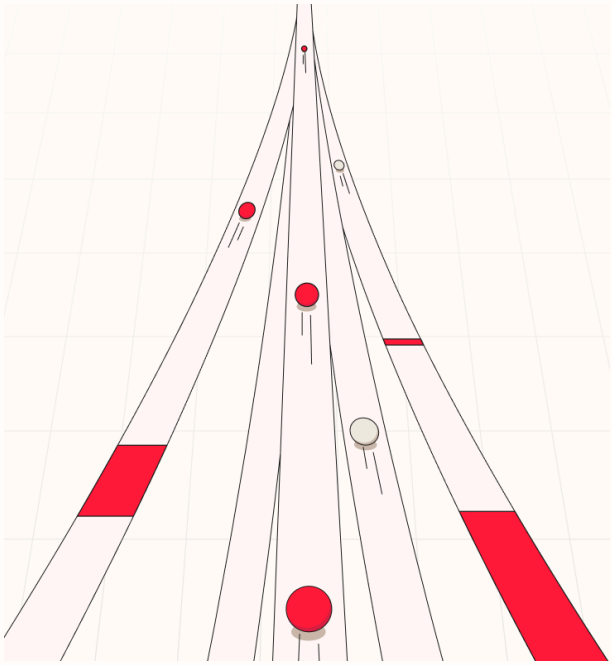
The result is a category of enterprise risk that didn't exist three years ago and now exists in global, cross-industry volume. These “homegrown” applications are generated faster than security teams can inventory them, deployed in places traditional discovery tools don't look, and connected to systems that traditional access controls assumed were reachable only through sanctioned channels. The catching up is happening on every side — platforms, enterprises, security tooling — and none of it is finished.

> **Shadow AI and Shadow Builders are the working vocabulary of a problem that is already in your environment, whether or not you've named it yet.**

Chapter 02

What We Found, and How.

No exploits. No credentials. No vulnerabilities. The threshold that matters is that the exposures existed at the level of a public URL.



METHODOLOGY · SCOPE · DISCLOSURE

An important thing to say about this research is what it isn't.

It isn't a hack. No exploits were used. No credentials were obtained. No authentication was bypassed. No vulnerability in any platform's infrastructure was discovered or weaponized. Every exposure documented in this report was reachable through ordinary access to the public internet — by anyone with a browser, no special tools required.

We have intentionally omitted the specific discovery techniques from this report; the threshold that matters is that the exposures existed at the level of a public URL. The mechanism didn't need to be sophisticated.

How the research began. We were working on finding a solution to help our clients with this exact problem: their employees were building applications on vibe-coding platforms, connecting them to internal systems, and the security team had no visibility into what existed, what data was inside, or who could reach it. As we worked on the solution, a broader pattern became visible. A meaningful percentage of those applications weren't just unsanctioned — they were actively reachable from the open internet, with sensitive data inside. The scale of what we found prompted us to look further.

The platforms. The investigation focused on the leading platforms in the vibe-coding category — the AI-driven app builders, no-code and low-code tools, and adjacent platforms (especially those that host new applications on their own subdomains) that have expanded dramatically in the past two years. They were selected because they are widely adopted, they make app deployment fast and frictionless, and they are representative of the category as a whole. The 380,000+ web assets and the headline numbers in the Executive Summary come from many different such platforms. But other platforms likely exhibit similar exposure patterns. We did not exhaust the category.

Every industry. Every continent.

The industries. Every industry examined contained Shadow AI applications with real exposure. Financial services, healthcare, government at every level, education, retail and wholesale distribution, manufacturing, logistics, consulting, real estate, enterprise software — including major vendors whose strategic documents were exposed not by their own employees but by third-party partners building tools that consumed sensitive partner data. We did not encounter an industry that was clean.

The geographies. Exposure was identified across six continents (Antarctica didn't make the cut). The pattern is not regional. It correlates with the presence of Shadow Builders — which is to say, with the presence of modern web-based work. Wherever employees use vibe-coding platforms to solve problems, exposure follows.

A note on disclosure. The number of exposures identified is in the thousands; notifying every affected organization individually is not operationally possible at that scale. Our approach has been to notify the most urgent and severe cases directly and to disclose the broader pattern to the vibe-coding platforms themselves. Since we started this work, many sensitive sites have been taken down — either by the platforms (usually making them “private”) or by the users themselves. It also illustrates that the pattern this report describes is not one any single notification could resolve. Remediation has to happen on all three fronts — the platforms, the enterprises adopting them, and the security industry — which is one of the reasons this report exists.

> **If your organization operates in any of the industries listed, in any of the regions listed, with any meaningful number of employees who use the modern web — the pattern in this report is statistically likely to exist somewhere in your environment.**

Chapter 03

Findings: A Closer Look.

The exposures share a single structural property: they were not protected by anything. What varies is the data inside, and the kind of system the data was inside of.

FOUR CATEGORIES OF EXPOSURE

Grouped by type, not by source.

The cases below are representative, not exhaustive. They are grouped by type of exposure rather than by industry, region, or platform — because the pattern that matters is the type, not the source. And maybe one of them will sound familiar in a way that will help you mitigate a risk in your environment, today.

CATEGORY 01 / MOST CONSEQUENTIAL

Live operational systems.

Not static documents or copied datasets — running enterprise systems, dashboards, internal portals, workflow tools, connected to live integrations like Power BI, Jira, and Notion, reflecting current operational state. A reader who reaches the URL is not seeing a snapshot of the data. They are seeing what the organization's own employees see when they log in.

Representative: Live financial dashboard, large LATAM bank.

Also: Government field-operations dashboard.

CATEGORY 02 / REGULATED

Regulated personal data.

Subject to specific regulatory regimes — HIPAA, GDPR, the Data Protection Act, and equivalent frameworks elsewhere. The applications documented here met none of those requirements. The data was reachable on the open internet, with no audit trail, no access control, and no organizational accountability for who viewed it.

Representative: National health-service contractor — named clinicians, direct phones, hospital affiliations.

CATEGORY 03 / NDA-GRADE

Strategic and confidential business documents.

Where operational systems leak ongoing business state, these applications leak deliberately sensitive documents: commercial plans, partner playbooks, client-facing deliverables, internal competitive strategy. Material that, in any other context, would carry an NDA, a confidentiality clause, or both.

Representative: Vendor-prepared business-optimization analysis for a named enterprise client. Transaction volumes, fees, effective rates, quarter-over-quarter commentary.

CATEGORY 04 / ACCESS PRIMITIVES

Credentials, API keys, and default administrative access.

The categories above expose data. The cases here expose *access* — the foundational primitives an attacker would otherwise have to compromise an account to obtain. Some applications stored API keys and integration secrets in places reachable through the public URL. Others granted administrative privileges by default to anyone who reached the page.

Representative: Internal customer-conversation tool labeled “Admin Panel” — live chat history, contact details, full configuration.

The exposures were not concentrated. They were distributed.

What this inventory adds up to.

The cases above are drawn from one sector each — finance, government, healthcare, real estate, B2B services. Eight more from different sectors would have read identically. The pattern does not vary by industry, by region, or by company size. Fortune 500 enterprises were exposed through the homegrown tools their third-party partners built. Mid-market organizations were exposed through tools their own employees built. Public institutions were exposed through tools built by the staff who run them.

What unifies them is the architecture, not the actor. Every case began the same way: an employee with a productivity goal opened a vibe-coding platform, built something useful, connected it to the data they needed, and published it — without ever considering the security primitives expected from enterprise software.

For a CISO reading this: the safe assumption is that something resembling at least one of the categories above exists in your environment. Not because you have careless employees, but because your employees are doing exactly what employees at every other organization in this report did.

WHAT COMES NEXT

The next chapter examines how these applications get built — how the path from "I need a quick tool" to "publicly reachable production system with admin access by default" completes itself in fewer steps, and with fewer guardrails, than most security teams would believe.

Chapter 04

Anatomy of a Shadow Build.

The path is short. Most Shadow Builds complete in under a day, often in under an hour. There are roughly five steps.

FIVE REASONABLE STEPS. ONE SYSTEMIC OUTCOME.

04 / THE FIVE-STEP DIAGRAM

From productivity intent to publicly reachable production system.

WHERE GOVERNANCE HAS TO LIVE

Steps 1 and 2 are productivity decisions you don't want to block. Steps 3, 4, and 5 are where visibility and policy have to live.

WHY IT'S SYSTEMIC

None of the five steps requires the builder to do anything wrong. The exposure is the cumulative output of five reasonable ones.

The five steps, in detail.

01

The intent.

A non-developer employee — a marketing manager, a finance lead, an operations coordinator — encounters a problem they want solved. *I need a way to track these proposals. I need a dashboard for the team. I need a form for vendors to submit invoices.* Three years ago, it would have ended with a complicated spreadsheet, an email thread, or a Notion page. The choice is no longer "settle for the spreadsheet" versus "file a ticket and wait three months." It is "settle for the spreadsheet" versus "have it working before lunch." The decision tilts predictably.

02

The build.

The employee opens a vibe-coding platform and types a description. Sometimes on the weekend. The platform generates a working application. The employee iterates by describing changes in plain language. Within minutes, the application looks and behaves close enough to what was wanted. In most of this process, the employee doesn't spend much time thinking about access control, authentication, identity, encryption, or the regulatory regime governing the data.

03

The connection. — the pivot.

This is the step that converts a personal project into an enterprise risk surface. The marketing manager wants the dashboard to show actual campaign data, so she connects it to her organization's BI tool. The finance lead wants the proposal tracker to read from existing project records, so he connects it to a CRM. From the security team's perspective, the result is a sanctioned enterprise system now reading from or writing to an application **no one in IT provisioned**, hosted on a platform **no one in IT vetted**, governed by access controls **no one in IT specified**. Many simply upload the data — pasting customer records into a chat field. From an architecture standpoint, this variant is in some ways worse: there is just data, somewhere it shouldn't be, in an application no one knows exists.

04 **The deployment.**

When the application is ready, the employee publishes it. On most vibe-coding platforms, this is a single-click action. The application is assigned a URL on the platform's subdomain and goes live immediately. Usually no staging workflow, no review gate, no security check, no organizational approval. The deployment is the build's natural endpoint. Several platforms have moved to private-by-default modes — but the inventory of applications already in public-default state is substantial.

05 **The access controls (or absence of them).**

Once the application is live, who can use it is governed by whatever the builder set up. In the cases documented in the previous chapter, the answer was overwhelmingly nothing. **The URL was the only credential** — sometimes paired with a decorative login page — and the URL was indexed by public search engines. In a meaningful number of cases, the application went further: it granted administrative access by default. Anyone who reached the URL was treated as the system's owner.

What this path produces.

An enterprise application with none of the markers traditional security architecture relies on to find and govern it. It does not appear in any inventory. No procurement record exists. No identity provider issued credentials for it. No DLP policy was authored for it. No firewall rule references it. No CASB connector indexes it.

> The security team's existing controls are not failing to protect this application — they are, in a very specific sense, not aware that it exists.

Chapter 05

Why Your Stack Missed This.

The problem is not a lack of telemetry. It is a lack of correlation, attribution, and governance.

THE CORRELATION GAP

Three structural realities.

Most security teams have a mature stack: EDR on endpoints, DLP for data, CASB for SaaS, firewalls and SSE for the perimeter, and in some cases an enterprise browser layered on top. The relevant fragments of signal for vibe-coded applications exist across all of these systems. They remain disconnected.

Existing tools produce signal at their own layer, but none are positioned to recognize that a generic vibe-coding-platform domain in a firewall log is actually a custom finance app, connected to corporate data, and published to the public internet.

EDR & ENTERPRISE BROWSERS

The visibility boundary.

To an EDR agent, building a vibe-coded application looks like a standard, non-malicious browser session — a *Context Gap*. Even where modern suites do see inside the build flow, they only do so on devices the organization owns and operates inside the browsers it manages — an *Ownership Gap* that grows wider as Shadow Builders work from personal laptops, contractor machines, and BYOD devices.

DLP & CASB

The channel limitation.

CASB was built for Shadow IT, where the unsanctioned thing was an unsanctioned SaaS vendor with a discoverable identity; it cannot recognize an unbounded population of custom applications hosted on a vibe-coding platform's subdomains. DLP catches user-mediated leakage but cannot see system-mediated leakage, where a vibe-coded application connects programmatically and moves data cloud-to-cloud, bypassing endpoint and email channels entirely.

FIREWALL & SSE

The contextual blind spot.

A firewall sees traffic to a vibe-coding platform's primary domain. It does not natively know that a specific subdomain is a discrete application with its own owner, its own data permissions, and its own public-exposure status. SSE platforms were designed to extend governance up the stack — but most deployments are partial. The gap remains.

A structural gap, not a tooling failure.

This is not a failure of investment. Each tool is doing its specific job competently. The Shadow AI category simply sits across the gaps between them, generating fragments of signal — a firewall log here, an OAuth grant there, an upload event somewhere else — that never assemble into a single, governable picture.

For a CISO, this is the operational reality the rest of the report assumes. Until something is positioned to correlate the fragments into a unified application inventory, the question of **"which vibe-coded applications exist in our environment, who built them, and what are they connected to?"** won't be answered by any single control already in your stack.

> **The chapters that follow describe how to start answering it anyway — through discovery, governance, and the layer where these applications actually live.**

Chapter 06

The Other Side.

Attackers are vibe coding too. What happens when the same platforms are used for the opposite purpose.

PATTERN OBSERVED · PATTERN EMERGING

Two patterns. They are not equivalent in evidentiary status.

The first is an observed finding from this research, the second is a category of threat the conditions documented in this report make plausible enough that security teams should put policies in place against it now, whether or not it has yet appeared in their environment. We treat them separately because the difference matters.

PATTERN 01 / OBSERVED

FOUND IN RESEARCH

Brand-impersonation phishing infrastructure.

Live phishing pages on vibe-coding-platform subdomains impersonating Bank of America, FedEx, Trader Joe's, McDonald's, and Costco — among others. The pages were polished. They used real brand assets, ran urgency timers, structured their funnels through familiar reward-claim flows, and at first glance were indistinguishable from legitimate consumer pages.

Sites built on vibe-coding platforms inherit the platform's primary domain — and from a reputation standpoint, the parent domain is legitimate, widely used, and trusted by every email and web security tool that filters by sender or destination reputation. **Domain-based filtering, one of the more reliable layers of email and web security for the past decade, performs less well against this pattern.**

PATTERN 02 / CATEGORY-OF-THREAT

CONDITIONS IN PLACE NOW

Internal impersonation.

A spoofed email from someone with plausibly-real internal authority, claiming to share “the new AI tool I built for the team — please log in to start using it,” with a link to a vibe-coded credential-capture page, exploits four conditions simultaneously: the email's domain reputation looks ordinary; the user's pattern-match doesn't fire because internal AI tools really do get shared this way now; the security team's discovery infrastructure cannot distinguish a malicious vibe-coded page from the dozens of legitimate ones already in the environment; and the existence of legitimate vibe-coded tools provides ambient cover.

Why both patterns are hard to detect from outside the session.

Vibe-coding platforms host attacker-built and employee-built applications on the same parent domains, with identical-looking URL structures, behind the same TLS certificates. From outside the browser session, an attacker's subdomain and a legitimate employee's subdomain are not meaningfully distinguishable on the signals email and web security typically rely on. **Distinguishing them requires inspecting what the applications actually do** — what they ask the user for, what data they capture, where they send it. That inspection has to happen inside the browser session.

What this means for security teams.

The first pattern is a domain-reputation problem that is structurally hard to solve through traditional reputation-based filtering, because the parent domain is legitimate. New layers of inspection and new categories of indicators are still being developed across the industry.

The second pattern is fundamentally a human-and-cultural problem before it is a technical one. It depends on the absence of a verification channel between “a colleague claims they built an internal AI tool” and “this is actually a colleague who built an internal AI tool.” That channel does not exist in most enterprises today.

> **Phishing that doesn't look like phishing because it looks like work.**

It is not a problem any single product solves. It is a defensive practice the security organization should build now — ideally before the first incident, rather than after.

Chapter 07

The Shared Responsibility Gap.

Three parties, each with work to do that the others can't do for them.

PLATFORMS · ENTERPRISES · THE SECURITY INDUSTRY

07 / WHO OWES WHAT

No single party can close the gap alone.

The exposures documented in this report are not the fault of any single party. They are the cumulative output of a category that has expanded faster than the people, the solutions, and the practices around it have been able to keep up.

PARTY 01

The platforms.

The most upstream point of leverage. A simple change to a default setting, applied across the user base, propagates immediately to every new application built on the platform from that day forward. No other party has that kind of reach. Several have moved to private-by-default, surfaced access-level prompts more prominently, restructured build flows.

PARTY 02

The enterprises.

The platforms' work, even when done well, does not reach inside an enterprise. Discovery, governance, and visibility live with the security organization — and remain its responsibility regardless of how mature any individual platform's defaults become. The four-piece job: discovery, policy, education, ongoing visibility.

PARTY 03

The security industry.

Closing the gap inside any single existing tool is incremental progress; closing it across the category requires correlating multiple signals into a single picture. The work is to keep building toward correlation and discovery purpose-built for this category, rather than retrofitting Shadow AI into mental models designed for Shadow IT.

What's still unfinished, on the platform side: the population of applications already deployed under earlier defaults remains live. New defaults usually secure new applications. The platforms' next move — and a very consequential one for enterprise customers — is some form of revisiting that existing inventory.

The platforms' work is the platforms'. The security industry's work is the security industry's.

An enterprise with a thousand employees has, almost certainly, some number of vibe-coded applications already in its environment — built by employees, by vendors, by partners — connected to internal systems, deployed at unknown levels of public exposure, governed by unknown access models. **Knowing what exists, who built it, what it touches, and who can reach it is the enterprise's work.** No platform, however well-recalibrated, can do it for them.

The work itself is not exotic. It is the same work enterprises have done before for other categories that emerged faster than governance could keep up: a discovery effort to inventory what already exists, a policy effort to define what should and shouldn't happen going forward, an education effort so employees understand what's acceptable and where the reporting paths are, and an ongoing visibility practice so new applications surface as they're built rather than after they cause an incident.

> **Neither one is finished. Neither one is going to finish soon enough to make the enterprise side of the responsibility wait.**

The next two chapters are about that work in practice — first the diagnostic, then the action plan. The platforms recalibrating is good news. It is not, on its own, your remediation.

Chapter 08

Could This Be Happening in Your Org?

Five questions. The point is not to test your stack — it is to surface where the answers, if you tried to produce them today, would be guesses.

A DIAGNOSTIC

If a question feels uncomfortable to answer with confidence, that is the chapter doing its job.

01 **Do you know which vibe-coding platforms are in active use across your workforce?**

Not just licensed — in *active use*. By everyone. Including employees on personal accounts, contractors on their own machines, vendors building tools that touch your data, and partners running apps that integrate with your systems. “We don't have a corporate license” is not the same as “no one is using one.”

02 **Of the applications already built, do you know which are connected to corporate systems — and through what?**

OAuth grants against your CRM. API keys pulling from your BI tool. Service accounts wired into your ticketing system. Manual uploads of data from your data warehouse into a chat field. Each of these is a connection. The question is not whether such connections exist. It is whether you can list them.

03 **Do you know which of those applications are publicly reachable on the internet?**

How many require *organizational* authentication — verifying not just that the visitor has an email account, but that they belong to your organization? The answer set you're looking for is concrete: a list of applications, with a status next to each.

04 **If a Shadow Builder shared a vibe-coding-platform link in Slack or email tomorrow, would your security team be able to tell whether it was real?**

If an attacker spoofed the same pattern — an email from a plausibly-real internal sender with a link to a vibe-coded credential-capture page — what would let your team distinguish it from a legitimate share? If the answer is “the URL looks weird,” the answer is no.

05 **If an exposure already exists in your environment, what would have to be true for you to find out about it before it becomes an incident?**

Today, in most environments, the answer involves luck — a journalist's call, a customer's note, a researcher's disclosure. The mature answer is a discovery and monitoring practice that surfaces vibe-coded applications continuously, not reactively.

Chapter 09 / The Action Plan

What to Do Monday Morning.

Five moves, in order. Each is a category of work, not a single task. Together they form the foundation of a practice you should expect to maintain ongoing.

Five moves. Start this week.

- [ ]

MOVE 01 — THIS WEEK

Discover what already exists.

Run a workforce-wide inventory. The first pass will be incomplete, because you are asking employees to volunteer information about tools they may not have understood needed disclosing. That's fine. Useful first prompt to circulate: *"If you have built a tool — an app, a dashboard, a form, a tracker — using an AI development platform, please tell us. We are not auditing. We are inventorying."*

- [ ]

MOVE 02 — WEEKS 1–2

Map connections and exposure.

For each application surfaced, capture three things: what corporate systems it is connected to, how, and whether it is publicly reachable. The connections are the load-bearing question. Public reachability is the most actionable signal in the short term — a candidate for immediate review or hardening regardless of what it contains.

- [ ]

MOVE 03 — WEEKS 2–4

Establish policy and an accepted-use path.

Name the platforms that are sanctioned, define what kinds of data are acceptable to load into them, define what authentication levels are required for any deployed application, and — most importantly — establish a registration path. Employees building tools should have somewhere to tell you. The path should be lower-friction than the alternative of them not telling you.

- [ ]

MOVE 04 — WEEKS 4–8

Brief vendors and partners.

A meaningful share of the exposures documented in this report came not from the affected organization's own employees but from third parties. Add vibe-coding platforms to your third-party risk program. Update vendor questionnaires. Notify partners formally. This will not catch everything. It catches more than the current default of catching nothing.

- [ ]

MOVE 05 — ONGOING

Build ongoing visibility.

Steps one through four are setup. This is the practice. The inventory you build this month will be incomplete next month. The mature posture is not a one-time discovery effort but an ongoing visibility practice — continuous discovery, continuous mapping, continuous review of what's connected and what's exposed. The point is that it exists, and that it does not depend on luck.

09 / VISIBILITY WITHOUT ENFORCEMENT IS OBSERVATION

Once you know what exists, the stack needs rules to match.

Some of this maps to tools you already own. Data-flow rules from sanctioned systems into vibe-coding platforms are a DLP and CASB exercise. Authentication requirements for connected applications are an identity exercise. Scrutiny on inbound vibe-coding links sits inside email and messaging controls. These are categories existing controls can be pointed at.

The harder categories don't map cleanly. Discovering applications deployed in publicly reachable configurations, surfacing ownership and exposure at the application level, correlating platform usage with what was built and where it ended up — and then enforcing policy on any of it — requires two things the existing stack rarely delivers together:

REQUIREMENT 01

Consolidated visibility at the layer where vibe-coding activity actually happens — instead of fragments scattered across DLP, CASB, identity, and network logs that no one is reliably correlating.

REQUIREMENT 02

The ability to enforce policy on what that visibility surfaces — in the same place, at the same layer, without bouncing the decision through another tool.

> **The question stops being which rules to write and starts being where visibility and enforcement have to live, together, to reach this category at all.**

Chapter 10 / The Architectural Answer

Where Visibility Has to Live.

Vibe coding is, end to end, a web-session event.

THE SESSION LAYER

Every step of the anatomy happens at the session layer. Not adjacent to it. Inside it.

This is not a product question. It is a question about layers — about which vantage point in the modern enterprise is positioned to see the activity that produces a Shadow AI application from the moment an employee opens a vibe-coding platform to the moment that application becomes reachable on the public internet.

The build is a browser event. The user opens a tab, describes what they want, watches code generate in the same tab, and iterates from there. The connection — the OAuth grant that ties the new application to a sanctioned enterprise system — is a browser event. The data the application is built around — pasted, dragged, copied, pulled — moves through the session. The deployment is a browser event. The publish action that turns the build into a live application at a public URL is a click inside the same tab where everything else happened.

> **A control positioned at the session layer therefore sees the whole build path — not a fragment of it.**

It sees the platform being opened, the prompts being typed, the OAuth grant being granted, the data being moved, the application being published, the user doing all of it. It sees the activity in the form it actually occurs, attributable to a specific person and a specific application instance, regardless of which platform was used or which network path the traffic took.

The structural difference between the session layer and the rest.

Device layer	Sees the operating system, not what is happening inside the session. EDR can tell you a browser process is running. It cannot tell you that the browser process is publishing an application to a vibe-coding platform's free-tier domain.
Network layer	Sees traffic, not application-as-business-object. A firewall or SSE can tell you a user reached a vibe-coding platform's domain. It cannot tell you the user just deployed a customer-data dashboard there.
SaaS catalog layer	Sees known vendors, not custom-built applications. CASB can govern access to the vibe-coding platform itself. It has no concept of the thousands of distinct applications built on top of it.
Channel layer	Sees watched endpoints, not the OAuth flows that bypass them. DLP can inspect a file upload to a sanctioned destination. It is not positioned to see the moment a custom application receives access to a sanctioned system through a browser-mediated grant.
Managed-browser layer	Sees inside the session — but only on the devices and browsers it controls. The closest architectural neighbor to a session-layer answer. Its limitation is reach. The work follows the worker. A control that requires the worker to use a specific browser, on a specific device, follows neither.
Session layer	**The only layer that is present regardless of device, browser, network path, or whether the application being built is sanctioned or not.** It is present because the session is what the work is made of. There is nowhere else the activity can occur.

The category is maturing. The platforms are recalibrating. The session is where vibe coding happens.

This becomes more important, not less, as the category matures. Platforms are recalibrating. New defaults are shipping. Future builds will be safer. None of that addresses the inventory of applications already deployed under the previous defaults — applications that are reachable now, connected to enterprise data now, governed by no one now.

Discovering what already exists is not a configuration task on a forward-looking control. It is an archaeology task that depends on having the raw material to do it: the OAuth grants, the domain patterns, the build-flow signals, the publish events, the user attributions. **Those signals exist at the session layer. They do not exist, in correlatable form, anywhere else.**

The same is true for the BYOD and contractor dimension that defines so much of how this category actually presents. A control that follows the work — across any device the worker uses, any browser they prefer, any network they happen to be on — is the only kind of control that meets vibe coding where it occurs. Anything anchored to a managed device or a managed browser sees only the part of the category that crossed its boundary. The rest happens elsewhere, and stays invisible.

> **The security layer that governs what happens at the session layer, across any browser and any device, is the one that finally sees this category whole.**

redaccess

ABOUT RED ACCESS

The session-layer security platform.

Red Access is the agentless, session-layer security platform delivering Security Service Edge (SSE) capabilities without the deployment and management cycles of traditional architectures. Built for the modern enterprise and operating at the web session — across browsers, SaaS, GenAI, extensions, and desktop apps — Red Access is invisible to end users and integrates with existing security environments, making it fast to deploy across global, hybrid, and contractor workforces.

If the patterns described in this report resemble what you suspect is happening in your own environment — or if you would like to find out — reach out.

Talk to the team:

info@redaccess.io

info.redaccess.io/request-a-demo