Niche Sonar Opportunity Example

2026-07-01

2 The Evidence

Real discussions that flagged this — see for yourself

EVIDENCE SUMMARY

Cursor on iOS was reported to irreversibly alter privacy settings without user consent, triggering significant community alarm. Developers on HN are actively asking for secure wrapper solutions for coding agents, indicating the problem is real and unsolved. The AGENTS.md feature request on GitHub shows teams want standardized agent behavior contracts, pointing to a governance gap.

REAL VOICE FROM DISCUSSIONS

> *"Ask HN: Secure wrapper for coding agents?"*

COMMON THEMES

- Cursor iOS install silently changed privacy settings — users unable to revert
- HN thread explicitly asking for 'secure wrappers' for coding agents
- AGENTS.md feature request signals demand for standardized agent governance
- No existing tool audits what AI coding agents send over the network

#coding agent security | HN ↗ | GitHub ↗ | PH ↗

HN	Tell HN: Installing Cursor on iOS irreversibly changes your privacy settings	▲ 236
HN	Ask HN: Secure wrapper for coding agents?	▲ 27
GitHub	Feature Request: Support AGENTS.md.	▲ 303
GitHub	Divulgue seu perfil, divulgue seu projeto 🖐	▲ 2500
PH	Fundraisly	▲ 1178

1 Money

Proof that people are already paying for solutions

WHO PAYS

VP of Engineering or CISO at companies with 50+ developers using AI coding tools

LIKELY MONTHLY SPEND

$300-500

Enterprise security tooling routinely commands $300-500+/month per team and sits in clearly defined security/compliance budget lines, making payment highly likely once trust is established.

EXISTING ALTERNATIVES

Enterprises already pay for tools like Snyk, Prisma Cloud, and GitGuardian for code security scanning; AI agent governance is the next logical budget line in the same category.

CUSTOMER TYPES:

- CISOs at fintech firms using AI coding tools
- Engineering leads at HIPAA-regulated healthcare software vendors
- Dev tooling managers at defense contractors (CMMC compliance)

3 Opportunity

Why you can win — competitors' weaknesses and your gap

COMPETITION SUMMARY

No major incumbent directly governs AI coding agent permissions or data exfiltration. Cursor, GitHub Copilot, and Codeium compete on features but not on enterprise security controls or compliance audit trails.

UP AGAINST: Snyk | GitGuardian | Cursor | GitHub Copilot for Business

MARKET GAP

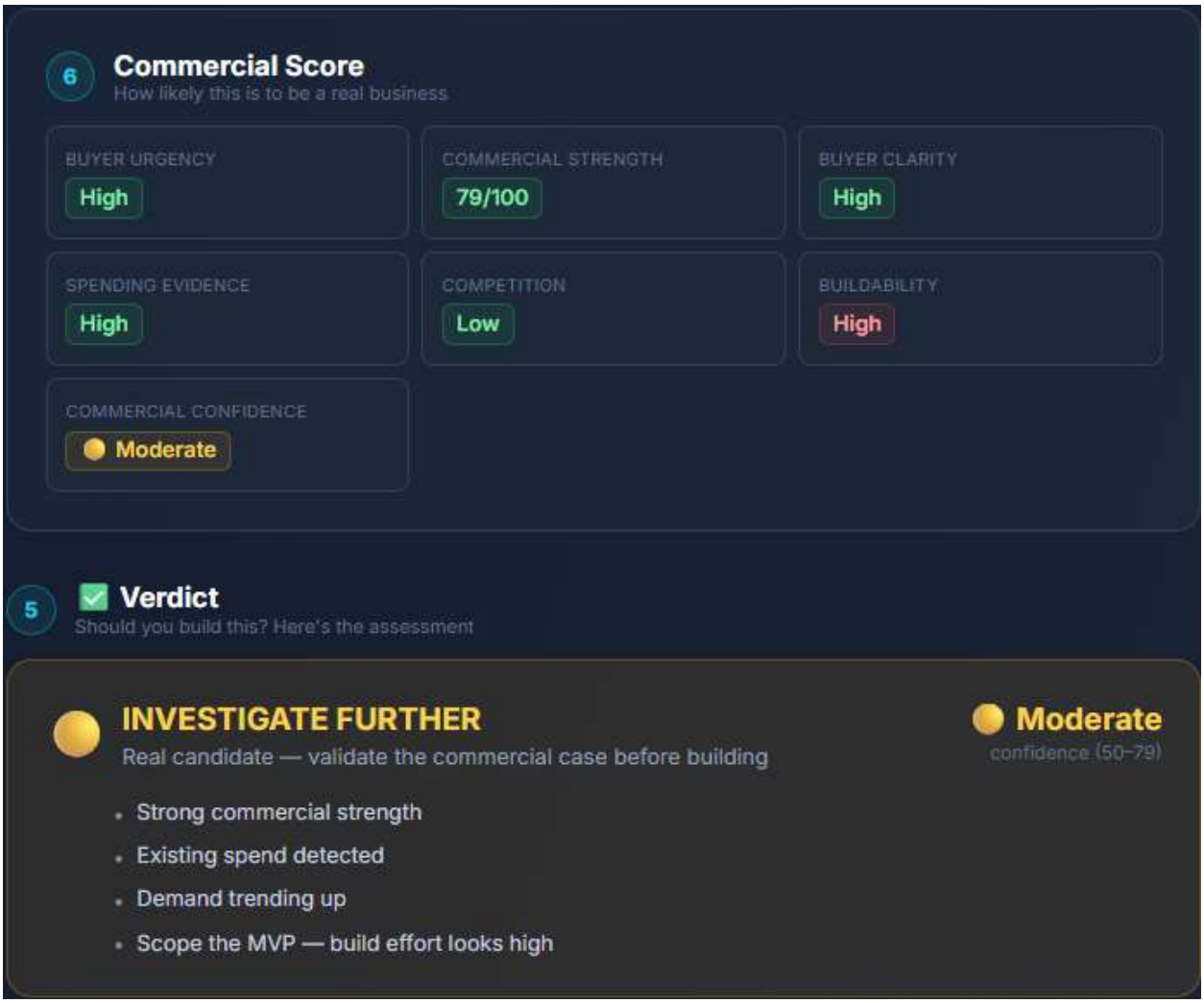
No tool currently provides a tamper-proof, auditable sandbox and permission layer specifically for AI coding agents — the gap is widest for regulated industries (finance, healthcare, defense contractors).

APPARENT GAP · HYPOTHESIS

Snyk and GitGuardian focus on static code vulnerabilities rather than runtime agent behavior, and Cursor/Copilot have commercial incentives to minimize friction rather than maximize enterprise control, suggesting a wedge exists for a neutral security-focused layer.

Why we think this →

Hype-driven | Regulatory risk


6
Commercial Score
How likely this is to be a real business
BUYER URGENCY
High
COMMERCIAL STRENGTH
79/100
BUYER CLARITY
High
SPENDING EVIDENCE
High
COMPETITION
Low
BUILDABILITY
High
COMMERCIAL CONFIDENCE
Moderate
5
Verdict
Should you build this? Here's the assessment
INVESTIGATE FURTHER
Real candidate — validate the commercial case before building
Moderate
confidence (50–79)
Strong commercial strength
Existing spend detected
Demand trending up
Scope the MVP — build effort looks high

COMMERCIAL CONFIDENCE · MODERATE (50–79)

Strong cross-platform signal with clear enterprise security budget ownership, but the technical complexity and sales cycle length reduce near-term solo viability.

WHY NOW

In 2025, AI coding agents have crossed from hobbyist to enterprise-standard tooling, creating an urgent compliance and data governance gap that security teams are now being held accountable for.

RISKS TO VALIDATE

- Cursor, GitHub Copilot, and other incumbents may ship native enterprise privacy controls, collapsing the wedge
- Enterprise security procurement cycles are 6-18 months, making early revenue slow and burn high
- Regulatory frameworks for AI agent data handling are still undefined, making compliance claims legally fragile

AI Solution · Difficulty: Hard

Build a security proxy/gateway for AI coding agents (Cursor, Copilot, Codium, etc.) that sits between the developer's IDE and the LLM API. It intercepts outbound requests, scans for secrets/PII/proprietary code patterns before they leave the machine, and blocks or redacts violations.

Core features:

- Local lightweight proxy (installable via npm/brew) with configurable rules — block AWS keys, redact DB credentials, flag HIPAA-sensitive strings
- Team policy dashboard: define what code patterns are forbidden from leaving, get audit logs per developer
- One-click compliance reports (SOC2, HIPAA, GDPR) showing what was blocked/sent

Wedge to first users: Target CTOs at 10-50 person startups already using Cursor/Copilot who just got asked by a enterprise customer "how do you ensure your AI tools don't leak our data?" — they have no answer today. Post in Cursor's Discord, r/devops, and YC Slack. Offer free solo tier, charge $20/seat/month for teams.

Why it wins: Developers keep using their favorite AI tools. Security teams get visibility. No ripping-and-replacing existing workflow.

Why Hard: Building a reliable, low-latency local proxy that handles multiple IDE integrations, streaming LLM APIs, and enterprise SSO/policy management has significant technical surface area and a slow enterprise sales cycle.

MVP Spec - Effort: Medium

CTOs and security-conscious engineering leads at 10-50 person startups actively using AI coding agents (Cursor, Copilot, Codium) who face enterprise customer due diligence questions about AI data handling

Node.js (local proxy daemon, fast startup, npm distribution), mitm-proxy or http-proxy-middleware for traffic interception, Rust or WASM regex engine for low-latency scanning, SQLite for local audit log storage, Next.js + Tailwind for dashboard, Homebrew formula + npm package for distribution, Stripe for billing, Supabase for team policy sync (optional, opt-in)

Build a locally-installed proxy (via npm/brew) that intercepts all outbound requests from AI coding agents like Cursor and Copilot before they hit the LLM API. The proxy runs a fast, local scan engine that detects and redacts secrets (AWS keys, DB credentials, API tokens), PII, and configurable proprietary code patterns — blocking violations or stripping sensitive content before the prompt leaves the machine. No code is routed through a third-party server; everything stays local, which is itself a key trust differentiator.

Core MVP features: (1) Local proxy daemon with <50ms latency overhead, pre-built rule packs for OWASP secrets, HIPAA strings, and regex-based custom patterns; (2) a lightweight web dashboard (localhost or hosted) showing per-developer audit logs, block/redact events, and policy configuration; (3) one-click compliance snapshots exportable as PDF for SOC2/HIPAA/GDPR conversations with enterprise customers. The user flow is: install proxy in 2 minutes → proxy auto-detects IDE traffic → violations surface in dashboard → CTO shares compliance report with enterprise customer the same day.

GTM wedge is tight and urgent: target CTOs at 10-50 person startups already using Cursor or Copilot who just received an enterprise security questionnaire they cannot answer. Seed distribution via Cursor's Discord, r/devops, and YC Slack with a single 'got asked how you prevent AI tools from leaking data? here's your answer in 2 minutes' hook. Free solo tier (1 dev, basic rules) drives bottom-up adoption; $20/seat/month team tier unlocks centralized policy management, audit logs, and compliance exports. Land the CTO, expand seat-by-seat as the engineering team grows.

Success metrics for MVP validation: 200 installs in 30 days, 10 paying teams within 60 days, and at least 3 documented cases where the compliance report directly unblocked an enterprise deal — that last metric is the core value proof that justifies the price and the roadmap.

AgentShield

Coding Agent Sandbox

A security layer that enforces privacy rules, permission boundaries, and audit logs for AI coding agents running inside developer environments.

---

1. Opportunity Overview

Problem

AI Coding Agent Security & Privacy Controls

Evidence Summary

Cursor on iOS was reported to irreversibly alter privacy settings without user consent, triggering significant community alarm. Developers on HN are actively asking for secure wrapper solutions for coding agents, indicating the problem is real and unsolved. The AGENTS.md feature request on GitHub shows teams want standardized agent behavior contracts, pointing to a governance gap.

Buyer

VP of Engineering or CISO at companies with 50+ developers using AI coding tools

Existing Spending Evidence

Enterprises already pay for tools like Snyk, Prisma Cloud, and GitGuardian for code security scanning; AI agent governance is the next logical budget line in the same category.

Revenue Model

Per-seat SaaS subscription at $25–40/dev/month, sold to engineering or security leadership; enterprise contracts with compliance reporting add-ons at $500–2000/month flat for larger orgs.

Why Now

In 2025, AI coding agents have crossed from hobbyist to enterprise-standard tooling, creating an urgent compliance and data governance gap that security teams are now being held accountable for.

Competitors

Snyk, GitGuardian, Cursor, GitHub Copilot for Business

The Gap

No tool currently provides a tamper-proof, auditable sandbox and permission layer specifically for AI coding agents — the gap is widest for regulated industries (finance, healthcare, defense contractors).

Implementation Challenges

- Cursor, GitHub Copilot, and other incumbents may ship native enterprise privacy controls, collapsing the wedge
- Enterprise security procurement cycles are 6–18 months, making early revenue slow and burn high
- Regulatory frameworks for AI agent data handling are still undefined, making compliance claims legally fragile

2. AI Solution

Difficulty: Hard

Build a security proxy/gateway for AI coding agents (Cursor, Copilot, Codium, etc.) that sits between the developer's IDE and the LLM API. It intercepts outbound requests, scans for secrets/PII/proprietary code patterns before they leave the machine, and blocks or redacts violations.

Core Features

- Local lightweight proxy (installable via npm/brew) with configurable rules — block AWS keys, redact DB credentials, flag HIPAA-sensitive strings
- Team policy dashboard: define what code patterns are forbidden from leaving, get audit logs per developer
- One-click compliance reports (SOC2, HIPAA, GDPR) showing what was blocked/sent

Wedge to First Users

Target CTOs at 10–50 person startups already using Cursor/Copilot who just got asked by an enterprise customer "how do you ensure your AI tools don't leak our data?" — they have no answer today. Post in Cursor's Discord, r/devops, and YC Slack. Offer free solo tier, charge $20/seat/month for teams.

Why It Wins

Developers keep using their favorite AI tools. Security teams get visibility. No ripping-and-replacing existing workflow.

3. MVP Spec

Effort: Medium

Build a locally-installed proxy (via npm/brew) that intercepts all outbound requests from AI coding agents like Cursor and Copilot before they hit the LLM API. The proxy runs a fast, local scan engine that detects and redacts secrets (AWS keys, DB credentials, API tokens), PII, and configurable proprietary code patterns — blocking violations or stripping sensitive content before the prompt leaves the machine. No code is routed through a third-party server; everything stays local, which is itself a key trust differentiator.

Core MVP Features

1. Local proxy daemon with <50ms latency overhead, pre-built rule packs for OWASP secrets, HIPAA strings, and regex-based custom patterns
2. A lightweight web dashboard (localhost or hosted) showing per-developer audit logs, block/redact events, and policy configuration
3. One-click compliance snapshots exportable as PDF for SOC2/HIPAA/GDPR conversations with enterprise customers

The user flow is: install proxy in 2 minutes → proxy auto-detects IDE traffic → violations surface in dashboard → CTO shares compliance report with enterprise customer the same day.

GTM Wedge

GTM wedge is tight and urgent: target CTOs at 10–50 person startups already using Cursor or Copilot who just received an enterprise security questionnaire they cannot answer. Seed distribution via Cursor's Discord, r/devops, and YC Slack with a single 'got asked how you prevent AI tools from leaking data? here's your answer in 2 minutes' hook. Free solo tier (1 dev, basic rules) drives bottom-up adoption; $20/seat/month team tier unlocks centralized policy management, audit logs, and compliance exports. Land the CTO, expand seat-by-seat as the engineering team grows.

Success Metrics

200 installs in 30 days, 10 paying teams within 60 days, and at least 3 documented cases where the compliance report directly unblocked an enterprise deal — that last metric is the core value proof that justifies the price and the roadmap.

4. Phased Plan

Timeline: 8–12 weeks (Phase 1–3)

Phase 1: Core Proxy Engine & Secret Detection (Weeks 1–2)

Goal

Ship a working local proxy that intercepts AI coding agent traffic and redacts secrets before they leave the machine. This is the trust-critical foundation everything else depends on.

Timeline: 2 weeks

What to Build

- Local proxy daemon (Node.js or Go) installable via npm install -g and brew install
- MITM proxy layer that intercepts HTTPS traffic from Cursor and Copilot (using node-http-proxy or mitmproxy core)
- CA certificate auto-installation to handle TLS interception without breaking IDE functionality
- Fast local scan engine (<50ms target) with pre-built rule packs:
  - OWASP secrets: AWS keys, GitHub tokens, Stripe keys, private keys
  - HIPAA strings: SSNs, DOBs, MRN patterns
  - Regex-based custom pattern config via a local JSON/YAML file
- Block mode (reject request + log) and Redact mode (strip sensitive content + forward sanitized prompt)
- Structured local audit log (SQLite or append-only JSONL) storing: timestamp, rule triggered, content hash, action taken (no raw content stored)
- Auto-detection of Cursor and Copilot traffic via User-Agent and endpoint fingerprinting
- Basic CLI status command: aiproxy status, aiproxy logs, aiproxy test

Deliverables

- Working npm/brew installable package
- Proxy running in <2 minute setup on macOS (Linux stretch goal)
- Measurable <50ms latency overhead (benchmark suite included)
- At least 15 pre-built detection rules across 3 categories
- Local audit log with structured events
- README with 5-minute quickstart guide

Dependencies

- Decide on proxy runtime: Node.js (faster to ship, easier npm distribution) vs Go (better performance, single binary) — recommend Node.js for Phase 1 speed, Go rewrite optional in Phase 3
- TLS interception approach validated on macOS Sequoia and macOS Ventura (OS-level cert trust varies)

- Cursor and Copilot traffic endpoint mapping (reverse-engineer request shapes to know what to intercept)

Risks & Mitigation

- **TLS interception complexity**: Some IDEs pin certificates or use custom TLS stacks. Mitigation: test against actual Cursor and Copilot builds week 1, have fallback to HTTP_PROXY env var injection if MITM approach hits walls.
- **Latency budget**: Regex scanning at 50ms is achievable but needs profiling. Mitigation: benchmark with 10KB, 50KB, 100KB payloads in week 1; pre-compile all regex patterns at startup.
- **macOS Gatekeeper / security warnings**: Unsigned binaries will trigger warnings. Mitigation: Apple Developer account + code signing is a day-1 task, not an afterthought.
- **False positives breaking IDE flow**: Overly aggressive rules will frustrate developers and cause uninstalls. Mitigation: default to Redact mode (not Block), and make Block mode opt-in.

Customer Feedback Trigger

End of Week 2: DM 5 CTOs from target list (Cursor Discord, YC Slack) with a Loom demo + download link. Specific questions: Did it install in 2 minutes? Did it break your IDE? Did it catch anything real? This feedback gates Phase 2 scope.

Success Criteria

- Proxy installs and runs without errors on 3 different macOS machines
- Latency overhead measured at <50ms on 95th percentile for typical Copilot/Cursor payloads
- At least 3 beta testers confirm it did not break their IDE workflow
- At least 1 beta tester reports a real secret being caught

Phase 2: Dashboard, Policy Config & Compliance Export (Weeks 3–5)

Goal

Build the lightweight web dashboard and one-click compliance PDF export that turns raw audit logs into a story a CTO can share with an enterprise customer the same day. This is the monetization unlock and the GTM hook.

Timeline: 3 weeks

What to Build

- Localhost web dashboard (served by the proxy daemon on localhost:7432)
- Per-developer view: events timeline, block/redact counts, rules triggered
- Policy configuration UI: enable/disable rule packs, add custom regex patterns, set Block vs Redact mode per category
- Real-time event feed (WebSocket or polling)
- One-click compliance snapshot:

    - PDF export generated locally (Puppeteer headless or PDFKit — no data leaves machine)
    - Report includes: policy configuration summary, event counts by category, time range, redaction rate, zero-violation attestation if applicable
    - Pre-formatted sections referencing SOC2 CC6.1, HIPAA §164.312, GDPR Art. 32 — language enterprise customers recognize
    - Branded with company name (configurable in settings)
- Team tier backend (lightweight hosted API, Supabase or PlanetScale):
    - Centralized policy management: push policy configs to multiple developer machines
    - Aggregated team audit logs (hashed/anonymized content, metadata only)
    - Seat-based auth via API key distributed to each proxy install
    - Stripe integration for $20/seat/month billing
- Solo tier enforcement: 1 seat, basic rules, local-only dashboard, no PDF export (PDF is team-tier paywall)
- Auto-update mechanism for rule packs (pulls from hosted endpoint, no agent code updates without explicit user action)

Deliverables

- Localhost dashboard accessible immediately after proxy start
- PDF compliance report generatable in <10 seconds
- Team tier with centralized policy sync working across 2+ machines
- Stripe checkout flow for team tier upgrade
- Updated install flow that surfaces dashboard URL after setup completes
- Landing page (single page, no fluff): problem statement, 2-minute install GIF, pricing, compliance report sample PDF as lead magnet

Dependencies

- Phase 1 audit log schema must be stable before dashboard consumes it (define schema contract end of Phase 1)
- PDF report template needs review from at least 1 target CTO before building — validate the compliance language resonates
- Hosted team tier backend needs a domain, SSL, and basic auth before Stripe goes live
- Legal: ToS and Privacy Policy needed before any hosted data collection, even metadata

Risks & Mitigation

- **PDF compliance language not matching what enterprise security teams want**: Mitigation: in Week 3, collect 2–3 actual enterprise security questionnaires from beta users (they have them — that's the wedge) and reverse-engineer the exact language needed.
- **Team policy sync adding latency or complexity**: Mitigation: keep policy sync async and cached locally; proxy never blocks on network call to hosted backend.
- **Stripe / billing complexity eating scope**: Mitigation: week 4 only — do manual invoicing for first 3 paying teams if Stripe isn't ready; don't let billing block the product.

- **Dashboard becoming a scope trap**: Mitigation: no charts, no fancy UI. Ship an HTML table with filters. Aesthetics are Phase 3. Function ships Phase 2.

Customer Feedback Trigger

End of Week 3 (dashboard alpha): Share dashboard access with 5 Phase 1 beta users. Ask: Is this the data you'd show your board? What's missing from the compliance report?

End of Week 5 (before Phase 3): Attempt to close first 3 paying teams manually. Their objections define Phase 3 priorities.

Success Criteria

- Dashboard loads in <2 seconds on localhost
- PDF report generated and reviewed positively by at least 2 target CTOs
- At least 1 paying team on the team tier before Phase 3 begins
- Install-to-dashboard time under 3 minutes measured on a fresh machine

---

Phase 3: GTM Distribution, Hardening & Expansion (Weeks 6–9)

Goal

Scale installs to 200+, convert 10 paying teams, and collect 3 documented compliance-report-unblocked-deal cases. Harden the proxy for production reliability, expand IDE coverage, and build the distribution flywheel.

Timeline: 4 weeks

What to Build

- Distribution & GTM execution:
  - Cursor Discord post (pinned if possible): single hook message with Loom demo + install command
  - r/devops and r/netsec posts: lead with the security questionnaire pain, not the product
  - YC Slack (W25/S25 batches): peer-to-peer intro via any YC connection, not cold spam
  - Product Hunt launch: schedule for Week 7, coordinate with beta users for upvotes
  - SEO landing page targeting 'prevent Cursor from leaking secrets', 'Copilot data privacy', 'AI coding tool SOC2 compliance'
- Proxy hardening:
  - Windows support (previously macOS-only) — npm install path is the same, cert trust differs
  - Linux support (Ubuntu/Debian primary, important for dev containers and CI)
  - Proxy crash recovery: daemon auto-restarts, IDE traffic falls through (never blocks dev work on crash)
  - Rule pack versioning and rollback: if a rule update causes false positives, one command to revert

    - Performance profiling pass: target <30ms p99 after optimizations
- Expanded detection coverage:
    - GitHub Actions secrets patterns
    - GCP and Azure credential formats
    - Common PII patterns: email, phone, passport numbers
    - Proprietary code pattern templates: common patterns for fintech, healthtech verticals
- Enterprise trust features:
    - Air-gap mode: fully offline operation, no outbound calls whatsoever (enterprise requirement)
    - Audit log integrity: HMAC signatures on log entries to prove logs weren't tampered
    - SSO-ready auth hooks for team tier (SAML/OIDC stub, not full implementation)
- Customer success infrastructure:
    - Automated onboarding email sequence for new team tier signups (3 emails: setup confirmation, first-week check-in, compliance report reminder)
    - Slack connect channel template for enterprise team customers
    - Case study capture process: email template sent at Day 14 to team tier customers asking 'did this help you answer a security questionnaire?'
- Metrics instrumentation (privacy-preserving, opt-in):
    - Anonymous install counts, rule trigger rates, upgrade conversion funnel
    - No prompt content, no code, no PII — just product usage signals
    - Dashboard for internal use showing funnel: install → dashboard view → upgrade → compliance export

Deliverables

- macOS + Windows + Linux proxy support
- 200+ total installs (tracked via unique API key generation on install)
- 10 paying teams on $20/seat/month
- 3 written case studies (even 2-paragraph Slack DM quotes count) showing compliance report unblocked a deal
- Product Hunt listing live
- Air-gap mode working and documented
- Internal metrics dashboard showing install-to-paid funnel

Dependencies

- Phase 2 dashboard and PDF export must be stable — Phase 3 distribution will bring volume that exposes bugs
- Windows TLS certificate trust requires testing on Windows 10 and 11 — needs a Windows test machine or CI runner by Week 6
- Case study collection requires paying customers from Phase 2, which requires at least 1 paying team before Phase 3 starts
- Product Hunt launch needs 20+ upvoters lined up before launch day — start recruiting from beta list in Week 5

Risks & Mitigation

- **Distribution falls flat, 200 installs not hit**: Mitigation: if organic channels stall at Week 7, activate direct outreach to 50 CTOs via LinkedIn with the security questionnaire hook — this is a known pain point with high response rate.
- **Windows TLS complexity delays cross-platform support**: Mitigation: ship Windows support as 'beta' with a known-issues doc rather than blocking the launch; most early adopters are on macOS anyway.
- **Security researchers find vulnerabilities in the proxy itself**: Mitigation: publish a responsible disclosure policy on the landing page before Phase 3 distribution; the product handles sensitive data and will attract scrutiny — get ahead of it.
- **Churn from free tier users who never upgrade**: Mitigation: the PDF export paywall is the key conversion lever; if free-to-paid conversion is below 5% at Week 8, consider adding centralized audit logs (not just local) as an additional team-tier incentive.
- **Competitor copies the idea**: Mitigation: the trust moat is local-only processing + audit log integrity + compliance language quality — document these differentiators clearly in all GTM materials. Speed to 10 paying teams matters more than defensibility at this stage.

Customer Feedback Trigger

Week 6: 30-minute calls with first 5 paying teams — ask what would make them buy 5 more seats, and what would make them cancel.

Week 8: Collect NPS from all active installs via in-dashboard prompt. Any score below 7 triggers an automated 'what went wrong?' email.

Week 9 (end of Phase 3): Full retrospective against MVP success metrics. 200 installs / 10 paying teams / 3 case studies — go/no-go for Series A narrative or pivot decision.

Success Criteria

- 200+ installs within 30 days of Phase 3 GTM launch
- 10 paying teams generating $X MRR (minimum $2,000 MRR at average 10 seats/team)
- 3 documented case studies where compliance report directly unblocked an enterprise deal
- <1% proxy-caused IDE breakage rate reported by users
- p99 proxy latency <30ms on standard Cursor/Copilot payloads
- Zero security incidents or data leaks from the proxy infrastructure itself