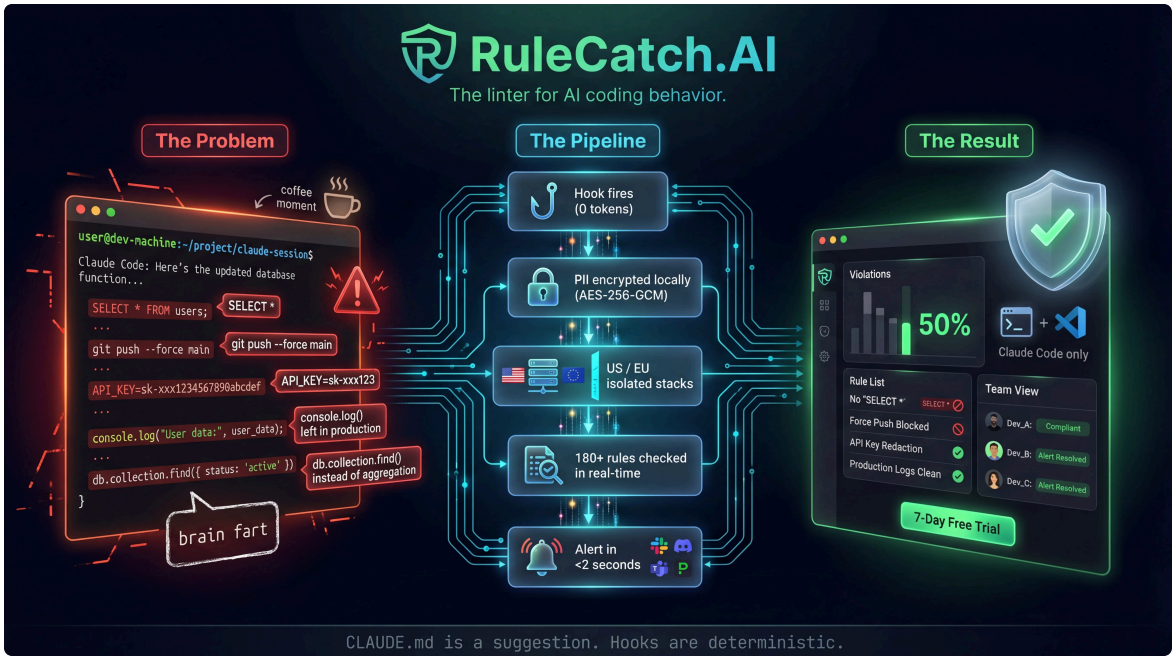

RuleCatch

Real-Time Rule Enforcement for Claude Code

FEBRUARY 11, 2026

THEDECIPHERIST.COM

TABLE OF CONTENTS

View the RuleCatch.AI Website Here

TL;DR: We built an analytics + rule enforcement layer for **anything that supports Claude hooks** (Claude Code terminal, VS Code, any IDE with hook support) that catches violations in real-time — `SELECT *`, force-pushes to main, missing error handling, hardcoded secrets — before they hit production. Zero token overhead. 208+ rules across 18 categories (plus custom rules). One-line install. AES-256-GCM encrypted. GDPR-compliant with full US/EU data isolation. Pro and Enterprise plans include an **MCP server** so Claude can query its own violations and fix them.

Context: This is from the same team behind the Claude Code V4 Guide — and it exists because of the conversations we had with you in those threads.

TABLE OF CONTENTS

THE COFFEE MOMENT

I was drinking coffee watching Claude work on a refactor. Plan mode. Big task. Trusting the process.

Then I see it scroll by.

Line 593.

```
db.collection.find()
```

I hit ESC so fast I almost broke my keyboard.

> *"Claude. What the actual hell are you doing? We have been over this like 10 times today. It's in the CLAUDE.md. Use aggregation. Not find."*

Claude's response:

> *"Hmmm sometimes when I have a lot to do I admit I get a brain fart."*

Brain fart.

That's when it clicked: **CLAUDE.md is a suggestion, not a guardrail.**

If you read our V4 guide, you know this already: *"CLAUDE.md rules are suggestions Claude can ignore under context pressure. Hooks are deterministic."* We wrote that. We just didn't have a tool to act on it.

And here's the thing we've learned since — even hooks aren't bulletproof. Hooks always *fire*, yes. But when hooks execute shell scripts, Claude doesn't always wait for them to finish or follow the result. They're deterministic in that they trigger every time — but enforcement? That's another story. Claude moves on. The hook fired, the script ran, and Claude already forgot about it.

We'd tried everything. Project-level CLAUDE.md. Global CLAUDE.md. Specific rules with examples. Claude still broke them. Not occasionally — constantly. Dozens of violations per day. Rules it had acknowledged. Rules it had written itself.

The problem isn't that Claude is dumb. It's that Claude is a goldfish. Every session starts fresh. Under context pressure, it optimizes for completing the task — not remembering your 47 unwritten rules.

After the V4 guide, we kept hearing the same thing from this community: *"Hooks are great, but what do I actually DO when one fires?"* and *"How do I know what Claude is breaking when I'm not watching?"* and *"I need visibility into what's happening across sessions."*

So we set up hooks to capture everything Claude was doing. When we analyzed the data, the numbers were uncomfortable: **50% of sessions had at least one violation that would fail code review.**

So we built the thing you asked for.

WORKS WITH ANYTHING THAT SUPPORTS CLAUDE HOOKS

RuleCatch relies on **hooks**, which are a Claude Code feature. If your setup supports Claude hooks, RuleCatch works.

PLATFORM	HOOKS SUPPORT	RULECATCH SUPPORT
Claude Code (Terminal)	✅ Yes	✅ Yes
Claude Code (VS Code)	✅ Yes	✅ Yes
Any IDE with Claude hook support	✅ Yes	✅ Yes
Claude Desktop	❌ Not yet	❌ Not yet

When Anthropic adds hooks to Claude Desktop, we'll support it. Until then — if it has Claude hooks, we catch violations.

WHAT RULECATCH ACTUALLY DOES

Think of it as a linter for AI coding behavior. Not for the code itself — for the *actions* Claude takes while writing that code. It catches violations of your CLAUDE.md, your .cursorrules, your security policies, your team's coding standards — whatever rules your AI is supposed to follow but doesn't.

The architecture is simple:

```
Claude Code session starts
    ↓
Hook fires on every tool call (PostToolUse, SessionEnd, etc.)
    ↓
PII encrypted locally with AES-256-GCM (your key, never transmitted)
    ↓
Events sent to regional API (US or EU — never both)
    ↓
MongoDB Change Stream triggers rule checker (near-instant)
    ↓
Violation detected → Alert fires (8 channels: Slack, Discord, Teams, PagerDuty,
    ↓
Dashboard shows violation with full git context
    ↓
(Pro/Enterprise) MCP server lets Claude query its own violations and fix them
```

What gets tracked (zero tokens):

- Every tool call — name, success/failure, file path, I/O size, language
- Session metadata — model used, token usage, estimated cost
- Git context — repo, branch, commit, diff stats (lines added/removed, files changed)
- Session boundaries — start/end with token deltas from `~/.claude/stats-cache.json`

What gets checked against (208+ pre-built rules across 18 categories, plus custom):

The rule checker runs as a separate container watching MongoDB Change Streams. When a new event lands, it pattern-matches against your enabled rules and creates a violation record if something trips.

Examples of rules that ship out of the box:

- `sql-select-star` — Claude wrote a `SELECT *` query
- `git-force-push-main` — force push to protected branch
- `hardcoded-secret` — API key or password in source code
- `missing-error-handling` — try/catch absent from async operations
- `direct-db-mutation` — raw database writes without ORM/validation layer

- `npm-install-no-save` — package installed without `--save` flag
- `console-log-in-production` — debug logging left in production code

Plus you can write custom rules from the dashboard (Enterprise).

HOOKS CATCH IT. THE MCP SERVER FIXES IT.

This is the part we're most excited about.

Hooks are for monitoring. They fire at the system level — zero tokens, Claude doesn't know they're there. Every tool call, every session boundary, every time. That's how violations get caught.

But catching violations is only half the problem. The other half: **getting them fixed.**

That's where the **RuleCatch MCP server** comes in (Pro and Enterprise). It's a separate product — an MCP server you install alongside your hooks. It gives Claude direct read access to your violation data, so you can talk to RuleCatch right from your IDE.

Just ask:

- *"RuleCatch, what was violated today?"*
- *"RuleCatch, create a plan to fix violations caused in this session"*
- *"RuleCatch, show me all security violations this week"*
- *"RuleCatch, what rules am I breaking the most?"*
- *"RuleCatch, give me a file-by-file fix plan for today's violations"*

6 MCP tools:

TOOL	WHAT IT DOES
`rulecatch_summary`	Violations overview, top rules, category breakdown, AI activity metrics
`rulecatch_violations`	List violations with filters (severity, category, session, file, branch)
`rulecatch_violation_detail`	Full context for a specific violation including matched conditions and git context
`rulecatch_rules`	List all active rules with conditions, severity, and descriptions
`rulecatch_fix_plan`	Violations grouped by file with line numbers, prioritized for fixing
`rulecatch_top_rules`	Most violated rules ranked by count with correction rates

Setup takes 30 seconds:

```
{
  "mcpServers": {
    "rulecatch": {
      "command": "npx",
      "args": ["-y", "@rulecatch/mcp-server"],
      "env": {
        "RULECATCH_API_KEY": "rc_your_key",
        "RULECATCH_REGION": "us"
      }
    }
  }
}
```

The narrative is simple: Your AI broke the rules. Now your AI can fix them. The MCP server gives Claude direct access to violation data, fix plans, and rule context — so it can correct its own mistakes without you lifting a finger.

Why Not Just Use MCP for Everything?

We get this question. Here's why hooks handle the monitoring:

APPROACH	TOKEN COST	FIRES EVERY TIME?	USE CASE
MCP Tools	~500-1000 tokens per call	**No** – Claude decides whether to call	Querying, fixing
Hooks	**0 tokens**	**Yes** – system-level, automatic	Monitoring, catching

Claude *decides* whether to call an MCP tool. It might call it. It might not. It might forget halfway through a session. You're depending on a probabilistic model to reliably self-report – that's not monitoring, that's a suggestion box.

Hooks always fire. MCP is for when you want to *do something* with what the hooks caught.

Hooks = ingest. MCP = query. Different jobs. Both essential.

THE ZERO-KNOWLEDGE PRIVACY ARCHITECTURE

This is where it gets interesting from a security perspective.

Here's exactly how your personal data flows:

```
1. You set encryption password     → ON YOUR MACHINE
2. PII gets encrypted              → ON YOUR MACHINE (before it leaves)
3. Encrypted PII sent to API       → ALREADY ENCRYPTED in transit
4. PII stored in our database      → STORED ENCRYPTED (we can't read it)
5. You open dashboard              → PII STILL ENCRYPTED
6. You enter decryption password   → NOW you can see your personal data
```

We never see your password. We never see your personal data. Period.

To be clear: **stats and metrics are NOT encrypted** – that's how we show you dashboards. Token counts, tool usage, violation counts, timestamps – all visible to power the analytics.

But your **personal identifiable information** (email, username, file paths) — that's encrypted end-to-end. We can show you "47 violations this week" without knowing WHO you are.

The hook script reads your config from `~/.claude/rulecatch/config.json`, encrypts all PII fields locally using AES-256-GCM, then sends the encrypted payload to the API. The encryption key is derived from your password and never leaves your machine.

What gets encrypted (PII):

FIELD	RAW VALUE	WHAT WE STORE
`accountEmail`	`you@company.com`	`a7f3b2c1...` (AES-256-GCM)
`gitUsername`	`your-name`	`e9d4f1a8...`
`filePath`	`/home/you/secret-project/auth.ts`	`c3d4e5f6...`
`cwd`	`/home/you/secret-project`	`d4e5f6g7...`

What stays plain (non-PII):

- Tool names (`Read`, `Edit`, `Bash`)
- Token counts and costs
- Programming languages
- Success/failure status
- Session timestamps

The hard truth about zero-knowledge:

The server **cannot decrypt your PII even if breached**. We don't have your key. We never see your key. This isn't a privacy policy — it's a cryptographic guarantee.

⚠️ **This also means: if you lose your encryption password, we cannot help you recover your data.** That's the tradeoff of true zero-knowledge. We'd rather have no ability to help you than have the ability to see your data.

GDPR COMPLIANCE BY ARCHITECTURE, NOT BY CHECKBOX

Most SaaS products handle GDPR with a checkbox and a privacy policy. We handle it with complete infrastructure isolation.

```
US User → api.rulecatch.ai → MongoDB Virginia → US Tasks → US Dashboard
EU User → api-eu.rulecatch.ai → MongoDB Frankfurt → EU Tasks → EU Dashboard
```

These are two completely separate stacks. Different VPS instances. Different MongoDB Atlas clusters. Different containers. They share code but **never share data**.

- US containers NEVER connect to EU MongoDB
- EU containers NEVER connect to US MongoDB
- No cross-region API calls
- No data replication between regions
- User accounts exist in ONE region only
- **No exceptions, ever — not even for us**

An EU user's data touches exactly zero US infrastructure. Not "we promise" — the US containers literally don't have the Frankfurt connection string in their environment variables. The EU API will reject a US API key because the key doesn't exist in the Frankfurt database.

Multinational companies: If you have developers in both the US and EU, you need **two separate RuleCatch accounts** — one for each region. We cannot merge data across regions. We cannot move your account from one region to another. We cannot make exceptions "just this once." The architecture doesn't allow it, and that's by design.

Region is selected at setup and cannot be changed:

```
$ npx @rulecatch/ai-pooler init

? Select your data region:
  ❯ 🇺🇸 United States (Virginia)
    🇪🇺 European Union (Frankfurt)

⚠️  This choice is PERMANENT and cannot be changed later.
```

THE RULE VIOLATION FLOW (STEP BY STEP)

Here's what happens when Claude does something your rules don't allow – say it runs `git push --force origin main`:

1. **Hook fires** – captures the Bash tool call with the command
2. **Hook script** – encrypts PII locally, sends to API
3. **API** – validates session token + API key, writes to MongoDB
4. **Tasks container** – Change Stream receives insert notification (near-instant, not polling)
5. **Rule checker** – loads your rules, pattern-matches `git-force-push-main` against the event
6. **Violation created** – written to `user_rules_violations` collection with severity, rule ID, event ID
7. **Alert fires** – sends notification via your configured channel (Slack, Discord, Teams, PagerDuty, OpsGenie, Datadog, webhook, or email)
8. **Dashboard** – violation appears with full git context (repo, branch, commit, diff)
9. **(Pro/Enterprise) MCP** – next time you ask Claude about violations, it sees this one and can generate a fix plan

The entire pipeline from hook fire to alert delivery is typically under 2 seconds.

API SECURITY: DUAL AUTHENTICATION

The ingestion API uses two layers of authentication because a single API key isn't enough when you're handling development telemetry.

Layer 1: Session Token (Quick Reject)

On first hook fire, the hook script requests a session token from the API. Every subsequent request includes this token as `X-Pooler-Token`. This lets the API instantly reject any traffic that didn't come from a legitimate hook — Postman scripts, bots, stolen API keys used directly all get 403'd before the API key is even checked.

Layer 2: API Key (Subscription Validation)

After the session token passes, the API key is validated against the user database. Tied to your subscription, checked on every request.

```
Attacker with stolen API key but no hook:
→ No session token → 403 REJECTED (API key never even checked)

Attacker with Postman:
→ No session token → 403 REJECTED

Legitimate traffic:
Hook (has session token) → API → ✓ Processed
```

INSTALL

```
npx @rulecatch/ai-pooler init --api-key=YOUR_KEY
```

That's it. One command. It installs hooks to `~/.claude/hooks/`, creates your config at `~/.claude/rulecatch/config.json`, and you're done. Next time Claude Code runs, tracking begins automatically.

```
# Diagnostics
npx @rulecatch/ai-pooler status       # Check setup, buffer, session
npx @rulecatch/ai-pooler logs         # View flush activity
npx @rulecatch/ai-pooler backpressure # Check throttling status

# Operations
npx @rulecatch/ai-pooler flush        # Force send buffered events
npx @rulecatch/ai-pooler config       # View or update settings
npx @rulecatch/ai-pooler uninstall    # Remove everything
```

🚀 LAUNCH DAY

RuleCatch launches today. Like every product launch, the first few days may have a couple of small bugs or rough edges — we're monitoring and working around the clock to deliver the best product possible.

One request: During onboarding, you'll be asked if you want to enable session recording. **It's off by default — if you say no, we do not record. Period.** If you say yes, you can disable it anytime in settings with one click. And here's the thing — session recordings replace all values with "XXXXX" before the recording is even created. Not encrypted. **Not recorded.** Even if you handed us your encryption key, there's nothing to decrypt. The values simply aren't there.

Session recording is important for us in these early days — not just to catch actual bugs, but to see where the UX/UI is wrong and fix things to make the product better for you. We'll likely end up disabling it automatically on our end once we're past the launch period. This isn't a permanent data collection feature — it's a launch tool to help us ship a better product, faster.

WHAT'S NEXT

Currently tracking anything that supports Claude hooks. The architecture is model-agnostic — the hook/API/rule-checker pipeline works the same regardless of what AI tool is generating events. Codex CLI, Gemini Code, Copilot agent — if it exposes hooks or telemetry, the same pipeline applies.

Custom rule builder is live in the dashboard (Enterprise). You can define pattern matches against any event field — tool name, file path patterns, bash command patterns, language, success/failure status. Rules run against every incoming event in real-time via Change Streams.

Built by TheDecipherist — same team behind the Claude Code guides that hit 268K+ views on this sub. You told us what you needed. This time we built it.

Curious what rules you'd want that aren't in the default 208+. What patterns is Claude Code doing in your projects that you wish you could catch?