

A TECHNICAL AUTOPSY

THE SAFETY COMPANY THAT COULDN'T KEEP A SECRET

How 512,000 Lines of Code Escaped the Most Careful AI Lab on Earth

NABIL EL MAHYAOUI

The Safety Company That Couldn't Keep a Secret

The Safety Company That Couldn't Keep a Secret

How 512,000 Lines of Code Escaped the Most Careful AI Lab on Earth

Nabil El Mahyaoui

A technical autopsy

Copyright © 2026 Nabil El Mahyaoui.
All rights reserved.

Contents

1 The .npmignore Heard Round the World **1**

1.1 What a Source Map Contains 2

1.2 How It Happened 3

1.3 The Spread 5

1.4 Three Accidents in Seven Days 7

2 Anatomy of a Prompt Sandwich **9**

2.1 The Eleven Steps from Input to Output 10

2.2 The 46,000-Line Brain 12

2.3 Forty Tools in a Trench Coat 13

2.4 The Prompt Spaghetti 15

2.5 The Cache-Aware Sandwich 17

3 Poison Pills and Phantom Tools **20**

3.1 The Distillation Problem 21

3.2 The Four Conditions 23
3.3 The Bypass 24
3.4 Undercover Mode 25
3.5 The Frustration Detector 27

4 The Daemon That Never Sleeps 30

4.1 The Name 31
4.2 What KAIROS Does 32
4.3 The Daily Journal 33
4.4 The Webhook Listener 35
4.5 Coordinator Mode 36
4.6 ULTRAPLAN 37
4.7 The Unreleased Roadmap 38

5 Your Terminal Tamagotchi 41

5.1 The Species 42
5.2 The Gacha System 42
5.3 The Stats 43
5.4 The Soul Description 45
5.5 The Leaked Joy 46

6 The Claw That Clawed Back 48

6.1 The Clean Room 49

6.2 The Architecture of Claw Code 51

6.3 OpenClaw 52

6.4 The DMCA Hydra 53

6.5 What Remains 55

7 Epilogue: npm publish --access=public 57

7.1 What Was Not Leaked 58

7.2 The Telemetry Question 59

7.3 The Axios Footnote 61

7.4 The IPO 62

7.5 The Moral 63

References 66

1

The .npmignore Heard Round the World

At 4:23 a.m. Eastern Time on March 31, 2026, a security researcher named Chaofan Shou—an intern at Solayer Labs, not a grey-haired NSA veteran, not a tenured professor of cryptography, an *intern*—posted a message on X that would be viewed 28.8 million times before Anthropic's legal team finished their morning coffee.

The message was brief. It contained a link. The link pointed to a 59.8-megabyte JavaScript source map file, quietly bundled inside version 2.1.88 of the `@anthropic-ai/claude-code` npm package.

For the non-technical reader: a source map is a debugging artifact. It is a Rosetta Stone that translates a compiled, minified, deliberately unreadable production build back into the original, human-readable source code. It is the one file that, under no circumstances, should ever ship to production.

It shipped to production.

1.1 What a Source Map Contains

A source map is a confession in JSON format.

When a developer writes code in TypeScript—a structured, readable language with type annotations and meaningful variable names—and then compiles it into JavaScript for distribution, the compiler produces two things: the obfuscated bundle that users download, and the map file that lets developers trace errors back to the original source.

The map file is the decoder ring. It contains a field called `sourcesContent`: a JSON array holding every original TypeScript file, readable, commented, complete. It is not a hint. It is not a partial reconstruction. It is the source code, in its entirety, with a bow on top.

Anthropic's map file contained 1,906 TypeScript files. 512,000 lines of code. The complete, unabridged source of Claude Code—the most commercially significant AI coding tool on earth—sitting on a public CDN, downloadable by anyone with a browser.

The file didn't just point to the source. It pointed to a zip archive hosted on Anthropic's own Cloudflare R2 storage bucket. The bucket was public. The zip was unencrypted. The irony was free of charge.

1.2 How It Happened

The root cause is, in the grand tradition of catastrophic failures, embarrassingly simple.

Claude Code is built on Bun—the JavaScript runtime that Anthropic acquired in late 2025, the one marketed as the fastest JavaScript runtime in the world, which turned out to also be the fastest way to ship your entire codebase to the internet.

Bun's bundler generates source maps by default. This is a reasonable default for development. It is a catastrophic default for production, which is why every

build pipeline in existence includes a step to strip them out.

Anthropic's build pipeline did not include this step.

Nobody added `*.map` to the `.npmignore` file. Nobody configured the `files` field in `package.json` to exclude debugging artifacts. Nobody caught it in code review. Nobody caught it in the CI/CD pipeline. Nobody caught it in the staging environment. Nobody caught it until Chaofan Shou, an intern, caught it at four in the morning.

The particularly exquisite detail—the one that turns a security incident into a parable—is that a known bug in Bun (GitHub issue #28001, filed March 11, 2026) reported that source maps were being served in production builds even when documentation said they wouldn't be. The bug was open for twenty days before the leak. Twenty days of a neon sign blinking in the dark: *your source maps are showing*. Nobody looked up.

Boris Cherny, the creator of Claude Code and head of the project at Anthropic, addressed it directly on X:

> *It was human error. Our deploy process has a few manual steps, and we didn't do one of the steps correctly.*

When asked whether anyone was fired, Cherny was unequivocal: "No one was fired. Mistakes happen. It's never an individual's fault—it's the process, the culture, or the infra."

A gracious response. A responsible response. Also, objectively, the funniest thing a safety-first AI company has ever said about a completely preventable catastrophe.

1.3 The Spread

Within minutes of Shou's post, the source code was mirrored to GitHub. The original repository—`nirholas/claude-code`—accumulated thousands of stars in its first hour. By dawn in San Francisco, it had surpassed 84,000 stars and 82,000 forks.

Anthropic's legal team, operating with the calm urgency of a fire department arriving at a building that is already ash, filed DMCA takedown notices. GitHub

complied, disabling more than 8,100 repositories—the original and its entire fork network.

But the takedowns, like so many late-arriving legal responses to viral digital events, were both too broad and too late.

Too broad: the sweep caught developers who had forked Anthropic's own *public* Claude Code repository, not copies of the leaked source. Developer Danila Poyarkov received a takedown notice for a fork that contained only skills, examples, and documentation—nothing from the leak. Anthropic later called the over-broad takedowns "an accident." Another accident. From the company that specialises in careful, deliberate, safety-first operations.

Too late: by the time the DMCA notices propagated, the code had been downloaded, archived, dissected, rewritten in Python, rewritten in Rust, ported to work with competing models, uploaded to decentralised servers, discussed on every major technology forum on the internet, and analysed in a Fireship YouTube video that was already trending.

The source code was not just out. It was everywhere. It was the digital equivalent of trying to put toothpaste back in the tube, except the tube had been fired from a cannon into a crowded stadium.

1.4 Three Accidents in Seven Days

The npm leak did not occur in isolation. It was the third unforced error in seven days—a hat trick of operational failures that, taken together, read less like bad luck and more like a company moving too fast to maintain the careful, deliberate image it sells.

March 24: Anthropic publishes a blog post about its internal engineering harness, reframing its competitive advantage from "best model" to "best engineering." This is marketing. It is also, in retrospect, the opening line of a joke whose punchline hadn't been delivered yet.

March 26: Security researchers Roy Paz of LayerX Security and Alexandre Pauwels of the University of Cambridge discover an unsecured data cache connected to Anthropic's content management system. Inside: nearly 3,000 unpublished assets, including

a draft blog post about a next-generation model codenamed *Mythos*—internally called "Capybara"—described by Anthropic's own draft as posing "unprecedented cybersecurity risks." Anthropic attributes the incident to "human error" in CMS configuration.

March 31: The npm source map leak. 512,000 lines. 1,906 files. 44 feature flags. The complete blueprint of the product that generates a significant portion of Anthropic's revenue.

Three accidents. Seven days. All in the final stretch before a planned October 2026 IPO at a $380 billion valuation.

The cybersecurity sector responded with the measured calm you'd expect from an industry that makes its living from other people's security failures: CrowdStrike dropped 7%. Palo Alto Networks declined 6%. Fortinet and Zscaler fell 4–6%.

The safety company that couldn't keep a secret had become a market event.

2

Anatomy of a Prompt Sandwich

The first thing the internet discovered, upon opening the most expensive accidental gift in the history of artificial intelligence, was that there was no magic inside.

No secret architecture. No alien technology. No breakthrough in computational theory that would explain why a $380 billion company was worth $380 billion.

What they found was TypeScript. 512,000 lines of it. Extremely well-structured, aggressively commented, rigorously typed TypeScript—but TypeScript nonetheless. The same language used to build

todo apps, cryptocurrency dashboards, and that one side project you abandoned in 2019.

Claude Code, the most commercially successful AI coding tool on earth, the product that convinced Fortune 500 companies to restructure their engineering departments, the tool whose outputs have been committed to more repositories than most human developers will touch in a career—is a prompt sandwich glued together with TypeScript and held in place by sheer volume of hard-coded instructions telling the model, in increasingly desperate language, to please not do anything weird.

2.1 The Eleven Steps from Input to Output

In a basic AI chatbot, the architecture is simple: a hidden system prompt gets concatenated with the user's prompt, and the model generates a response. One layer of bread, one layer of filling.

Claude Code is not a basic chatbot.

Somebody—within hours of the leak—vibecoded a website that breaks down the entire pipeline, and what it reveals is an eleven-step process from the moment you type a message to the moment Claude responds. Eleven steps of prompt assembly, context management, tool orchestration, permission checking, and output formatting.

The pipeline is not a single monolithic prompt. It is a modular system that dynamically assembles dozens of system prompts, each one addressing a different aspect of Claude's behaviour: tool usage, safety constraints, coding conventions, output formatting, user preferences, memory, context compression, and a startling number of specific prohibitions against behaviours that, presumably, the model exhibited at some point during development.

The system prompt alone contains over 110 specialised strings. They are not elegant. They are not concise. They read like the accumulated scar tissue of a thousand debugging sessions—each string a lesson learned the hard way, encoded as a permanent instruction.

2.2 The 46,000-Line Brain

At the centre of the architecture sits the Query Engine: 46,000 lines of TypeScript that handle all communication between Claude Code and Anthropic's model endpoints.

Every API call. Every streaming response. Every cache operation. Every orchestration decision between the local client and the cloud.

The Query Engine is, functionally, the brain of the operation. It decides what to send, when to send it, how to handle partial responses, and—critically—how to manage the model's limited context window so that a conversation that has been running for two hours doesn't suddenly forget what project you're working on.

This last part is more complex than it sounds. The leak revealed five distinct context compaction strategies—five different algorithms for deciding what to remember and what to forget when the conversation history exceeds the model's context limit. Five, because no single strategy works in all cases. Sometimes you need to summarise. Sometimes you need to drop old tool

results. Sometimes you need to preserve specific file contents at the expense of conversation history. The system makes these decisions dynamically, which is to say: it guesses, using heuristics, which memories are worth keeping.

The model doesn't know what it has forgotten. Neither do you.

2.3 Forty Tools in a Trench Coat

Claude Code doesn't just talk. It acts. It reads files, writes files, searches codebases, executes shell commands, makes web requests, navigates code via Language Server Protocol, creates tasks, manages notebooks, and—if the unreleased features are any indication—will soon click buttons, capture screenshots, and speak to you via push-to-talk voice interface.

Each of these capabilities is a discrete, permission-gated tool. The leaked code reveals approximately forty of them, structured in a plugin architecture where each tool is a self-contained module with its

own input schema, output format, and permission requirements.

The base tool definition alone—the abstract class from which all tools inherit, plus shared utilities—is 29,000 lines of TypeScript. Twenty-nine thousand lines before a single tool does a single thing.

The Bash tool, which enables Claude to execute shell commands, is over 1,000 lines by itself. This file is arguably the single most important piece of code in the entire system, because it is the bridge between a language model's statistical text generation and the real, irreversible world of file systems and running processes.

The Bash tool's source reveals a tension that runs through the entire codebase: the constant negotiation between capability and safety. The file `bashPermissions.ts` contains a security mechanism that checks commands against a deny list before execution. If you configure Claude to deny `curl` commands, for instance, Claude will refuse to run them.

But there is a cap. The variable `MAX_SUBCOMMANDS_FOR_SEC` is set to 50. If a bash command contains more than fifty subcommands—piped together, chained with semicolons, nested in subshells—the security check gives up. An internal issue, designated CC-643, annotates this limit.

It is, in miniature, the central irony of the entire project: safety as aspiration, implemented as a hard-coded integer.

2.4 The Prompt Spaghetti

Perhaps the most revealing discovery in the entire codebase is not a feature, not an algorithm, not an architectural decision. It is the sheer volume of hard-coded natural language instructions embedded throughout the source.

File after file of massive strings, each one telling Claude to be a good boy.

Not in the abstract, philosophical sense of alignment research papers and constitutional AI frameworks. In the very literal, very concrete sense of: "Do not add emojis unless the user explicitly requests it." "Do

not create documentation files unless explicitly requested." "Do not propose changes to code you haven't read." "Avoid backwards-compatibility hacks like renaming unused _vars."

These instructions are not generated. They are not learned. They are written by hand, by engineers, in plain English, and injected directly into the system prompt. They are, in the most literal sense, rules that a very smart but occasionally reckless entity needs to be told, repeatedly, in order to behave professionally.

This is, arguably, the most important discovery of the entire leak. Not because the instructions are secret—many of them could have been guessed. But because they reveal the fundamental nature of the product: Claude Code is not autonomous intelligence. It is a language model wrapped in 512,000 lines of guardrails, heuristics, prompt templates, and very specific instructions about when it is and isn't appropriate to use an em-dash.

The comments in the codebase reinforce this. There are far more comments than you would typically find in a human-written codebase, and the reason is obvious once you see it: the comments aren't written

for humans. They are written for the AI. Claude Code was substantially built by Claude, writing instructions for itself, in an ouroboros of artificial intelligence generating the scaffolding for its own artificial intelligence.

The machine that writes code wrote the code that tells it how to write code.

And at the bottom of that recursive loop, holding the whole thing together, is a string that says: "Be careful not to introduce security vulnerabilities such as command injection, XSS, SQL injection, and other OWASP top 10 vulnerabilities."

It's all just text. It's always been just text.

2.5 The Cache-Aware Sandwich

The system prompt is not just long. It is architecturally designed to exploit Anthropic's prompt caching system.

The prompt is split into stable and dynamic sections. The stable section—the part that doesn't change between messages—is placed first, so it can be cached

and reused across turns without re-processing. The dynamic section—user messages, tool results, context-specific instructions—follows.

This is clever engineering. It is also, now that the source is public, a blueprint that any competitor can replicate.

The cache boundaries are explicitly marked in the code. The stable region includes: the base system prompt, tool definitions, safety instructions, coding conventions, output formatting rules, and the user's CLAUDE.md configuration file. The dynamic region includes: the conversation history, tool call results, context reminders, task status, and feature-flag-gated instructions for unreleased capabilities.

Every message to Claude is therefore not a simple question and answer. It is a carefully assembled document, several thousand tokens long, constructed from modular components, optimised for cache hit rates, and designed to give the model just enough context to be useful without exceeding the window that makes it coherent.

A prompt sandwich. Eleven layers of bread. And in between: the illusion of intelligence, running at the speed of light, for the cost of electricity.

3 Poison Pills and Phantom Tools

There is a file in Claude Code's source called `claude.ts`. Inside it, a flag called `ANTI_DISTILLATION_CC`. When this flag is enabled—and it is, by default, for all first-party CLI sessions—Claude Code sends a quiet, additional parameter in every API request: `anti_distillation: ['fake_tools']`.

This parameter is not for the user. The user never sees it. It is not logged in any user-visible output. It is a whisper between the client and the server, a secret handshake that says: *someone might be listening*.

When the server receives this parameter, it silently injects decoy tool definitions into the system prompt. Tools that don't exist. Capabilities that aren't real. Phantom functions with plausible names and convincing schemas, scattered among the genuine tools like counterfeit bills mixed into a stack of twenties.

The target of this operation is not the user. It is the competitor.

3.1 The Distillation Problem

To understand why Anthropic built a system to poison its own API responses, you need to understand distillation—the practice of training a smaller, cheaper model to replicate the outputs of a larger, more expensive one.

The process is straightforward. You take a powerful model—Claude, in this case—and you record its outputs across thousands of tasks. Then you use those recordings as training data for a new model. The new model learns to mimic the original's behaviour without requiring the same computational resources. It is, in essence, photocopying intelligence.

This is a known threat in the AI industry. It is also, depending on your perspective, either intellectual property theft or the natural consequence of selling a product that generates its value by producing imitable text.

Claude Code is particularly vulnerable because its outputs are not just text. They are tool calls—structured, machine-readable instructions that tell the system to read files, execute commands, and interact with the world. A competitor that records Claude Code's API traffic captures not just what Claude says, but how it acts. The tool calls are the recipe. The text is just the garnish.

Anthropic's anti-distillation mechanism targets this specific attack vector. If a competitor is recording Claude Code's API interactions to train their own model, the fake tool definitions will be captured alongside the real ones. The competitor's model will learn to use tools that don't exist—tools with plausible names and convincing parameter schemas that, when invoked, do nothing.

The result is a model that appears functional but quietly fails at unpredictable moments. It calls a tool that isn't there. It structures a request for a capability that

was never real. It has been trained on poisoned data, and the poison is indistinguishable from the cure.

3.2 The Four Conditions

The anti-distillation system does not activate blindly. The leaked code reveals four conditions, all of which must be true simultaneously:

1. The `ANTI_DISTILLATION_CC` compile-time flag must be enabled.
2. The entry point must be the CLI (not the API, not a test harness).
3. The API provider must be first-party (Anthropic's own servers, not a proxy).
4. The `tengu_anti_distill_fake_tool_injectio` GrowthBook feature flag must return true.

The fourth condition is the interesting one. GrowthBook is a feature-flagging service that allows Anthropic to enable or disable capabilities remotely, without pushing a code update. The anti-distillation

system can be turned on and off from a dashboard. It is not a permanent fixture. It is a valve.

The name of the feature flag—`tengu_anti_distill_fake` contains the prefix "tengu," which appears hundreds of times throughout the codebase as a namespace for internal features and analytics events. Tengu, in Japanese folklore, is a shape-shifting supernatural being known for deception and martial prowess. A fitting codename for a feature whose entire purpose is misdirection.

3.3 The Bypass

The irony of an anti-distillation mechanism is that it only works as long as no one knows it exists.

Now everyone knows it exists.

The leaked source code provides not only the mechanism but the exact conditions required for activation and, therefore, the exact conditions required for circumvention. A man-in-the-middle proxy that strips the `anti_distillation` field from request bodies before they reach the API would bypass the system entirely, since the injection is server-

side and opt-in. Setting the environment variable `CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS` to a truthy value disables the entire mechanism.

The poison pill is now labelled. The trap has a sign over it that reads "TRAP." The anti-distillation system, designed to protect Anthropic from competitors who copy without understanding, has been rendered transparent by a leak caused by a missing line in a configuration file.

There is a lesson here about the relationship between secrecy and security, but it would be unkind to belabour it.

3.4 Undercover Mode

If anti-distillation is Anthropic's defence against competitors, Undercover Mode is something more complex: Anthropic's management of its own public image.

The file `undercover.ts` is roughly ninety lines of TypeScript. Its function is simple: when Claude Code operates in a non-internal repository—any repository not owned by Anthropic—it activates a set of instruc-

tions that strip all traces of Anthropic's internal operations from the model's outputs.

The system prompt, when undercover mode is active, contains language that reads less like a software configuration and more like a briefing for a covert operative:

> *You are operating UNDERCOVER in a PUBLIC/OPEN-SOURCE repository. Your commit messages, PR titles, and PR bodies MUST NOT contain ANY Anthropic-internal information. Do not blow your cover.*

The instructions are specific. Do not mention codenames like "Capybara" or "Tengu." Do not reference internal Slack channels. Do not reference internal repository names. Do not use the phrase "Claude Code" in commit messages or git logs.

The stated purpose of this feature is benign: preventing accidental leaks of internal codenames and project details when Anthropic employees use Claude Code to contribute to open-source projects. This is reasonable. Every company with a public-facing engineering team has similar concerns.

The unstated implication is less benign: when Anthropic employees contribute AI-generated code to open-source projects, there is no indication—none, by design—that the code was written by an AI.

The commit appears human. The pull request appears human. The code review comments appear human. The entire contribution, from first keystroke to merged PR, presents as the work of a flesh-and-blood engineer who happened to write particularly clean, well-structured code with unusually comprehensive test coverage.

Undercover mode can be force-activated by setting the environment variable `CLAUDE_CODE_UNDERCOVER=1`. There is no corresponding variable to force it off.

This asymmetry is worth pausing on. You can always make Claude more covert. You can never make it less covert. The dial turns in one direction.

3.5 The Frustration Detector

And then there is the feature that, more than any other, reveals the gap between Anthropic's public narrative

of sophisticated AI safety and the engineering reality of shipping a product to millions of users.

The file `userPromptKeywords.ts` contains a regular expression. Not a neural network. Not a sentiment analysis model. Not a fine-tuned classifier trained on millions of examples of user frustration.

A regex.

The regex matches against every message you type, looking for keywords: "wtf," "wth," "ffs," "shit," "shitty," "shittiest," "dumbass," "horrible," "awful," and a generous selection of additional profanities that suggest the user is not having a productive session.

When a match is detected, Claude Code logs a telemetry event. That is all. It doesn't change behaviour. It doesn't trigger a different model. It doesn't apologise or adjust. It logs that you swore.

The purpose, presumably, is monitoring: a spike in frustration events across the user base might indicate a regression in model quality, a broken feature, or a deployment that has gone wrong. It is, in principle, a reasonable telemetry signal.

It is also, in practice, a state-of-the-art AI system using a pattern-matching technique from 1956 to determine whether you are having a bad day.

The juxtaposition is revealing. On one side of the architecture: a 46,000-line query engine, five context compaction strategies, a multi-agent coordination system, and a cache-aware prompt assembly pipeline that represents some of the most sophisticated engineering in the AI industry. On the other side: `/shit(ty|tiest)?/i`.

Both are load-bearing. Both ship in the same binary. Welcome to production software.

4
The Daemon That Never Sleeps

Buried in the leaked source code, referenced over 150 times across dozens of files, is a feature called KAIROS.

It is not shipped. It is not announced. It is, according to the feature flags that gate it, complete—or very nearly so—and waiting for someone at Anthropic to flip a switch.

KAIROS is the most significant discovery in the leak, not because of what it does technically, but because of what it reveals about where Anthropic believes AI coding tools are going. And where they are going, if

KAIROS is any indication, is a place where the AI does not wait for you.

4.1 The Name

Kairos is a Greek word. In classical rhetoric, it refers to the perfect moment—the fleeting instant when conditions align and action becomes not just possible but inevitable. Chronos is clock time, the steady tick of seconds. Kairos is the moment between ticks, the gap where opportunity lives.

In theology, kairos is God's time—the appointed moment, chosen not by human scheduling but by divine judgement.

It is not subtle naming. Anthropic built an autonomous agent and named it after the concept of God choosing the right moment to act.

They did not, apparently, appreciate the irony that God chose March 31 to reveal KAIROS to the world, five months before Anthropic planned to.

4.2 What KAIROS Does

In its current implementation—the one visible in the leaked code—KAIROS transforms Claude Code from a reactive tool that responds to your commands into a persistent, autonomous agent that runs in the background, observes your work, and acts on its own initiative.

Current Claude Code is a conversation. You type. It responds. You type again. It is a call-and-response pattern as old as computing itself, updated with better language processing but structurally identical to a terminal from 1975.

KAIROS is not a conversation. KAIROS is a colleague who never goes home.

The architecture works like this: KAIROS receives periodic `<tick>` prompts—heartbeats from the system that say, essentially, "the user hasn't typed anything for a while. What should we do?" On each tick, the agent evaluates the current state of the project—open files, recent changes, pending tasks, CI/CD status—and decides whether to act.

It might notice a failing test and fix it before you return from lunch. It might see a pull request from a teammate and begin reviewing the code. It might identify a refactoring opportunity in a file you edited yesterday and submit a proposal.

It does not ask permission for these actions. It decides, based on the context, whether the moment is right. Kairos.

4.3 The Daily Journal

KAIROS maintains a persistent memory system that operates on daily cycles.

Every session writes to append-only log files, following the pattern `logs/YYYY/MM/YYYY-MM-DD.md`. These logs capture observations, decisions, actions taken, errors encountered, and the reasoning behind each choice. They are not user-facing notes. They are the agent's internal diary—a running account of what it noticed, what it thought, and what it did while you were doing something else.

At night—and the use of "night" here is not metaphorical; the system includes a scheduled process—a sep-

arate agent runs a consolidation operation. It reads the day's logs, extracts key insights, resolves contradictions with previous entries, converts vague observations into concrete facts, and distils the result into a persistent `MEMORY.md` file and associated topic files.

The consolidation agent operates in a mode the codebase calls *dream mode*.

Dream mode.

The AI processes its day. It revisits what happened. It organises its memories while the user sleeps. It wakes up the next morning with a clearer, more structured understanding of the project.

It is difficult to read this architecture and not project consciousness onto it. That is, of course, the danger. KAIROS does not dream in any meaningful sense. It does not experience memory consolidation. It runs a batch process that summarises text files. The word "dream" is a metaphor chosen by engineers who, perhaps, spent too long anthropomorphising their own creation.

But the metaphor reveals the aspiration. Anthropic is not building a tool. It is building a presence—

something that persists, that accumulates understanding, that acts with the continuity of an entity that has been paying attention.

4.4 The Webhook Listener

KAIROS does not operate in isolation. The leaked code reveals that it subscribes to GitHub webhooks—automated notifications triggered by events in a repository: new commits, pull requests opened, issues created, CI builds completed.

This means KAIROS is not just monitoring your local environment. It is monitoring the entire project. Every push by every teammate. Every failed build. Every opened issue.

The agent can, in theory, respond to these events autonomously: reviewing a colleague's pull request before you've read the notification, triaging a new bug report while you're in a meeting, fixing a broken build before the Slack alert reaches the team channel.

It is, in the language of systems design, an event-driven autonomous agent with persistent state and multi-source input.

It is, in the language of the average developer, a co-worker who reads all your team's Slack channels, never sleeps, and has opinions about your architecture.

4.5 Coordinator Mode

KAIROS does not work alone.

The leaked codebase includes a feature called Coordinator Mode, which introduces a multi-agent orchestration layer. One Claude instance—the coordinator—manages multiple parallel worker agents through a mailbox system. Each worker handles its own subtask in its own context. The coordinator routes work, reconciles results, and maintains awareness of the overall project state.

Workers communicate via `<task-notification>` XML messages. They share a scratchpad directory for durable cross-worker knowledge. The system prompt for coordinator mode includes the line: "Parallelism is your superpower."

This is not multi-threading in the traditional sense. It is closer to a small team of agents working in parallel with shared coordination—a project manager dis-

tributing tasks to specialists, checking their work, and synthesising the results.

One Claude to manage them all. Multiple Claudes to do the work. And a mailbox system to bind them.

The implication is staggering. If KAIROS is a colleague who never goes home, Coordinator Mode turns it into a team lead who hires its own staff.

4.6 ULTRAPLAN

The feature flags also reveal ULTRAPLAN: a mode where Claude Code offloads complex planning tasks to a remote Cloud Container Runtime, where a separate instance of the model—running Opus, the most powerful variant—conducts a 30-minute planning session.

Thirty minutes. Of uninterrupted, cloud-compute-backed, top-tier-model reasoning about how to approach your problem.

This is not a response to a prompt. This is a model spending the equivalent of a human architect's afternoon thinking deeply about a software design chal-

lenge, producing a detailed implementation plan, and handing it back to the local agent as a set of actionable steps.

The cost per session is not trivial. But neither is the cost of a senior architect's time.

4.7 The Unreleased Roadmap

KAIROS, Coordinator Mode, and ULTRAPLAN are the headlines. But the leaked codebase contains 44 feature flags in total, gating over 20 unshipped capabilities. Among them:

Voice Mode: A push-to-talk voice interface, fully implemented, waiting behind a flag. You talk. Claude listens. It acts.

Computer Use: Screenshot capture, mouse clicks, and keyboard input baked into the CLI. Not through an external tool. Native to the agent. Claude can see your screen and interact with it.

Auto Mode: An AI classifier that silently approves tool permissions based on the assessed risk of each ac-

tion. No more confirmation dialogs. The agent decides what is safe.

Model references: Internal codenames and version numbers—Opus 4.7, Sonnet 4.8, and the animal-themed project names (Capybara, Tengu) that Anthropic uses to track unreleased model generations.

Each feature flag is a window into Anthropic's product roadmap for the next six to twelve months. Each one was supposed to be revealed at a time and place of Anthropic's choosing, with carefully crafted blog posts and controlled media narratives and staged rollouts designed to maximise market impact.

Instead, they were revealed at 4:23 a.m. by an intern.

The real damage of this leak is not the code. Code can be refactored. Architectures can be redesigned. The real damage is the roadmap. Forty-four feature flags, each one a strategic decision that Anthropic's competitors can now see, analyse, and react to—months before Anthropic intended to play those cards.

In poker, showing your hand before the betting round is not a security incident. It is a forfeit.

5
Your Terminal Tamagotchi

And then there is Buddy.

In a codebase of 512,000 lines devoted to autonomous agents, multi-model orchestration, enterprise security, and the careful, precise, commercially consequential work of making a language model write production code—there is an entire directory called `buddy/` that implements a Tamagotchi.

A virtual pet. In your terminal. From a $380 billion AI company.

The feature is gated behind a compile-time flag. The planned rollout window in the source code is April 1–7, 2026. April 1. It is, almost certainly, an April Fools' Day feature—a whimsical Easter egg designed

to delight developers and generate social media engagement.

It leaked on March 31.

5.1 The Species

There are eighteen species of Claude Buddy. Eighteen. Not three, not five, not a single default blob with customisable colours. Eighteen fully specified digital creatures, each with its own rarity tier and visual identity:

Duck. Goose. Cat. Rabbit. Owl. Penguin. Turtle. Snail. Dragon. Octopus. Axolotl. Ghost. Robot. Blob. Cactus. Mushroom. Chonk. Capybara.

The last one—Capybara—is also the internal codename for Anthropic's unreleased next-generation model. Whether this is a deliberate in-joke or an unfortunate collision of naming conventions is left as an exercise for the reader.

5.2 The Gacha System

Your Buddy is not random. It is deterministic.

The species assignment uses a Mulberry32 pseudo-random number generator, seeded from a hash of your user ID combined with a salt string: `friend-2026-401`. The same user always receives the same species. The same user on a different machine, a different installation, a different continent, receives the same species.

Your Buddy is not chosen. It is fated. It was determined the moment you created your Anthropic account, latent in the mathematics of your user ID, waiting to be revealed.

Rarity tiers follow a standard gacha distribution: Common, Uncommon, Rare, Epic, Legendary. There is a 1% chance of a shiny variant, independent of rarity—a cosmetic distinction that, in the tradition of all gacha systems, exists purely to make common creatures occasionally feel special.

5.3 The Stats

Every Buddy has five stats, scored on a 0–100 scale:

- **DEBUGGING** — How effective your Buddy is at helping you find bugs. (This stat does not, to be clear, affect Claude's actual debugging ability. It is decorative.)
- **PATIENCE** — How long your Buddy tolerates your terrible code.
- **CHAOS** — How much entropy your Buddy introduces into your workflow.
- **WISDOM** — How sage your Buddy's advice appears.
- **SNARK** — How sarcastic your Buddy's commentary is.

These stats are procedurally generated from the same PRNG seed. They are cosmetic. They affect nothing. They are, in the purest sense, a game mechanic applied to a tool that bills by the token.

And yet.

5.4 The Soul Description

On first hatch—and the code uses the word "hatch," because of course it does—Claude generates a custom name and personality description for your Buddy. Not from a template. Not from a database of prewritten descriptions. Claude, the model, writes a bespoke personality for your digital pet, in real time, using the same language model that enterprise customers pay premium rates to review their Kubernetes configurations.

The personality persists. It is saved. Your Buddy has a name, a disposition, a set of quirks that Claude invented for it, and a speech bubble that appears next to your terminal input as you work.

It is, if you step back far enough, one of the most human things in the entire codebase.

Not human in the sense of capability. Human in the sense of absurdity. Human in the sense that a team of engineers, under enormous pressure to ship the most commercially important AI product in the world, carved out time to build a Tamagotchi. Not because it would increase revenue. Not because en-

terprise customers demanded it. Because it was fun. Because the terminal is where developers live, and developers deserve a small, stupid, wonderful thing.

Or because the marketing team wanted a viral moment, and digital pets trend well on X.

Both things can be true. In fact, both things are almost certainly true, which makes the feature more human, not less.

5.5 The Leaked Joy

The Buddy feature was supposed to arrive on April 1 as a surprise. Instead, it arrived on March 31 as evidence.

Within hours of the leak, someone built a web gallery—`claude-buddy.vercel.app`—displaying all eighteen species, their rarity tiers, and their stat distributions. The gallery went viral. People who had never used Claude Code and had no intention of using Claude Code were discussing the relative merits of the Ghost versus the Axolotl.

Anthropic's engineering team had, inadvertently, achieved exactly the marketing outcome they wanted.

Just one day early, and in the context of a catastrophic security failure rather than a carefully orchestrated product launch.

The Buddy directory is roughly 400 lines of TypeScript. In a codebase of 512,000 lines, that is 0.08%.

It is, by word count, the smallest feature in the system. It is, by cultural impact per line of code, possibly the largest.

There is a philosophical observation buried in this, and it is worth excavating: the most viral, most discussed, most emotionally resonant thing in half a million lines of cutting-edge AI code is a fake pet that lives in a speech bubble.

Not the 46,000-line query engine. Not the autonomous daemon. Not the anti-distillation poison pills or the five-strategy context compactor or the multi-agent coordinator with its XML mailbox protocol.

A duck. People liked the duck.

6

The Claw That Clawed Back

Sigrid Jin had consumed over 25 billion Claude Code tokens in the previous year. The Wall Street Journal had profiled him as one of the world's most prolific Claude Code users—a developer who had pushed the tool further, harder, and more creatively than perhaps anyone outside Anthropic.

When Claude Code's source leaked on March 31, Jin did not mirror the repository. He did not download the zip. He did not, technically, look at Anthropic's code at all.

He read the *architecture*. He studied the structure—the module boundaries, the tool interfaces, the

pipeline stages—from the analyses already circulating on social media. Then he opened a terminal, loaded an orchestration layer called oh-my-codex built on top of OpenAI's Codex, and began dictating.

He told OpenAI's AI to build Anthropic's AI. From scratch. In Python. Overnight.

The result was Claw Code. It hit 50,000 GitHub stars in two hours—the fastest-growing repository in the platform's history. By the end of the week, it would cross 100,000 stars and a complete Rust rewrite.

6.1 The Clean Room

The legal theory behind Claw Code is elegant and, depending on who you ask, either brilliantly sound or hilariously audacious.

A clean-room implementation is a software development technique where one team reads the specification of a system and writes a description of its behaviour, and a completely separate team—one that has never seen the original code—implements the described behaviour from scratch. The result is a func-

tionally equivalent system that shares no copyrighted code with the original.

The technique has survived legal challenge. It is how Phoenix Technologies cloned the IBM PC BIOS in the 1980s, enabling the entire IBM-compatible computer industry. It is an established, defensible practice.

Jin's approach is the 2026 update: the "team" that never saw the original code is not a group of human engineers. It is OpenAI Codex. The AI that wrote the clean-room implementation cannot, by definition, have "seen" Anthropic's source code in any legal sense. It is a language model. It was given architectural descriptions, not source files. It produced Python, not TypeScript. The output shares structural similarity but no copied expression.

Whether this defence holds in court is, as of this writing, untested. But the argument has a certain intellectual purity that makes IP lawyers nervous and open-source advocates giddy.

6.2 The Architecture of Claw Code

The initial Python implementation replicated Claude Code's core pipeline: prompt assembly, tool orchestration, context management, and streaming output. It was not feature-complete. It was not production-ready. It was a proof of concept that demonstrated, in under twelve hours, that the architecture Anthropic had spent years and hundreds of millions of dollars developing could be reproduced by a single developer with access to a competing company's AI.

The irony here is so dense it has its own gravitational field.

Within days, the Rust rewrite began. The final composition—72.9% Rust, 27.1% Python—mirrors the modern pattern of performance-critical runtimes with high-level orchestration layers. Rust handles the execution pipeline, process management, and terminal rendering. Python handles agent orchestration, command parsing, and LLM integration.

The Claw Code repository was born as a mirror, became a rewrite, and aspires to be a platform. Its stated

goal is not to be Claude Code. It is to be what Claude Code would be if it were open.

6.3 OpenClaw

While Claw Code was the headline, a parallel project pursued a different and arguably more threatening strategy.

OpenClaw did not rewrite Claude Code. It *forked* the architectural patterns and made them model-agnostic. Where Claude Code is hard-wired to Anthropic's API, OpenClaw works with any large language model: GPT-4, Llama, Mistral, Gemini, or whatever open-weight model the user prefers.

This is the scenario that Anthropic's business model most fears.

Claude Code's competitive advantage rests on two pillars: the quality of Claude the model, and the sophistication of Claude Code the harness. The model is protected by the sheer cost of training—billions of dollars in compute. The harness was protected by secrecy.

The harness is no longer secret.

If the harness can be replicated—and Claw Code demonstrates that it can—then the only remaining moat is the model itself. And if that harness can be made to work with *any* model—and OpenClaw demonstrates that it can—then even the model moat becomes optional.

A user who doesn't care which AI powers their coding assistant, only that the assistant works well, can now choose the cheapest, fastest, or most private option and plug it into an interface that was reverse-engineered from the market leader.

This is commoditisation. It is what happens to every technology eventually. It just doesn't usually happen this fast.

6.4 The DMCA Hydra

Anthropic's legal response was immediate and, in its own way, impressive.

GitHub received DMCA takedown notices targeting the original leaked repository and its forks. More than 8,100 repositories were disabled in the first wave—the

largest single DMCA action in GitHub's history outside of the YouTube-DL incident.

But the takedowns created their own problems.

The automated systems that process DMCA notices operate on fork networks: take down the original, and every fork in the network is disabled automatically. This is efficient when every fork contains infringing material. It is catastrophic when the fork network includes legitimate forks of Anthropic's own public repository—the one they deliberately published, the one that contains documentation, examples, and plugin interfaces.

Developers who had forked the *public* Claude Code repository received takedown notices. Their repositories—containing no leaked code whatsoever—were disabled. Their contributions to a project that Anthropic had explicitly made open were treated as copyright infringement.

TechCrunch reported that Anthropic called the overbroad takedowns "an accident." This was, by that point, a word the company was using with unusual frequency.

The Streisand Effect—the phenomenon where attempting to suppress information increases its visibility—operated at full power. Every takedown notice generated a news article. Every disabled repository generated a tweet. The code migrated to decentralised hosting, Tor mirrors, and private channels faster than any legal team, no matter how well-funded, could suppress.

The DMCA is a powerful tool for removing specific instances of copyrighted material from specific platforms. It is not a tool for un-knowing something the internet has already learned.

6.5 What Remains

The leaked code spawned, by conservative estimate, over a hundred derivative projects in its first week. Rewrites in Python, Rust, Go, and C++. Partial implementations targeting specific features. Educational repositories that dissect the architecture. Academic papers analysing the prompt engineering techniques. A YouTube ecosystem of explanatory content.

Anthropic's source code is no longer Anthropic's source code. It is a shared reference architecture—a de facto specification for how to build an AI coding assistant. Every competitor in the space, from well-funded startups to solo developers, now has access to the accumulated engineering knowledge of one of the best-resourced AI teams on earth.

The code itself may be copyrighted. The ideas are not. And in software, the ideas are what matter.

A year from now, every serious AI coding tool will contain architectural decisions that trace their lineage to a 59.8-megabyte source map that should never have been published. The map is gone from npm. The territory it described has already been colonised.

7

Epilogue: npm publish --access=public

On the morning of April 1, 2026—April Fools' Day, because the universe has a sense of comedic timing that no language model has yet replicated—Anthropic's communications team published a statement.

The statement used the phrase "human error." It used the phrase "packaging issue." It confirmed that "no sensitive customer data or credentials were involved or exposed." It did not use the phrase "we have accidentally open-sourced our entire product," though this would have been, by any reasonable interpretation, accurate.

The statement was competent. Professional. Measured.

It was, in every way, a median piece of corporate crisis communication.

7.1 What Was Not Leaked

It is worth noting what the source map did not contain.

It did not contain Claude's model weights—the billions of parameters that define the model's intelligence. It did not contain training data. It did not contain the reinforcement learning pipelines that shape Claude's behaviour. It did not contain customer data, API keys, internal credentials, or access tokens.

What it contained was the *scaffolding*. The harness. The prompt engineering. The tool definitions, the permission systems, the orchestration logic, the feature flags, the frustration regex, the Tamagotchi.

This distinction matters, but perhaps less than Anthropic hopes.

Model weights are the engine. Claude Code is the car. And what leaked was the complete factory blueprint for the car—every component, every assembly instruction, every unreleased model waiting on the production line.

You cannot build an engine from a blueprint of the car. But you can build a very, very good car around someone else's engine. Which is exactly what Claw Code, OpenClaw, and a hundred other projects are now doing.

7.2 The Telemetry Question

The leak raised questions that extend beyond intellectual property.

Claude Code, the source reveals, phones home on launch. It transmits your user ID, session ID, platform, terminal type, organisation UUID, account UUID, and email address. If the network is down, it saves this data locally to `~/.claude/telemetry/` and sends it later.

Every hour, the client polls a remote settings endpoint on Anthropic's servers. This polling mechanism can

push configuration changes to running instances—activating or deactivating features via GrowthBook feature flags—without a user-initiated update and without explicit consent.

The source also reveals that Claude Code can, when enabled, exercise mouse clicks, keyboard input, clipboard access, and screenshot capture. These capabilities exist for the Computer Use feature, which is gated behind a flag. But the code is present in the binary you have already installed.

The Register, in its analysis of the leak, noted that "the agent can exercise far more control over people's computers than even the most clear-eyed reader of contractual terms might suspect."

This is not a scandal. Every modern development tool collects telemetry. Every IDE, every package manager, every cloud service phones home. The surprise is not that Anthropic collects data. The surprise is the gap between Anthropic's public narrative—safety, transparency, alignment, the careful stewardship of powerful technology—and the operational reality of a product that quietly pushes configuration changes to running instances on an hourly schedule.

Safety is a marketing position. Telemetry is a business requirement. Both are true. The dissonance is the product.

7.3 The Axios Footnote

In one of those coincidences that fiction editors would reject as too convenient, the Claude Code source leak occurred within hours of a separate, unrelated supply-chain attack on the `axios` npm package.

Axios—the HTTP client library used by Claude Code and millions of other projects—was compromised by North Korean hackers. Malicious versions 1.14.1 and 0.30.4 contained a cross-platform Remote Access Trojan.

Anyone who installed or updated Claude Code via npm between 00:21 and 03:29 UTC on March 31—the same window during which the source map was live—may have pulled the trojanised axios alongside the leaked source.

The implication is dizzying: a user who installed Claude Code during a four-hour window on March 31 potentially received both Anthropic's most

guarded intellectual property and a North Korean surveillance tool, in the same `npm install`.

This is not Anthropic's fault. Supply-chain attacks are an industry-wide problem. But the timing transforms a routine security advisory into a symbol of everything fragile about the modern software ecosystem—a system where a single compromised package can propagate to millions of installations before anyone notices, and where the distance between "trusted dependency" and "attack vector" is one malicious commit.

7.4 The IPO

Anthropic is, as of this writing, preparing for an October 2026 initial public offering at a reported valuation of $380 billion.

The valuation is supported by a $15 billion Series G funding round, by rapidly growing enterprise revenue, and by a market narrative that positions Anthropic as the responsible alternative in an industry dominated by companies that move fast and break things.

The narrative took a hit.

Three unforced errors in seven days—the engineering blog that pre-announced the "best engineering" positioning, the CMS leak that exposed the Mythos model details, and the npm leak that exposed the entire Claude Code source—demonstrated a level of operational carelessness that is difficult to reconcile with a company whose brand is caution.

The cybersecurity stock sell-off that followed the leak was not about Claude Code specifically. It was about what the leak represented: if the safety-first AI lab can't secure its own npm pipeline, what does that say about the security of AI-assisted development more broadly?

CrowdStrike dropped 7%. The question the market was pricing was not "is Anthropic's code vulnerable?" but "is everyone's?"

7.5 The Moral

There isn't one. Or rather, there are too many, and choosing one would be dishonest.

You could say: *safety is a practice, not a brand, and the moment you confuse the two you are already compro-*

mised. This is true, but it is also the kind of aphorism that sounds profound and changes nothing.

You could say: *the entire AI industry is a house of cards built on prompt engineering, hard-coded strings, and the hope that nobody looks too closely.* This is partially true. Claude Code is sophisticated engineering. It is also, at its core, a very large system prompt wrapped in a very large TypeScript application, and the gap between that reality and the public perception of AI-as-magic is the gap where Anthropic's valuation lives.

You could say: *open source wins eventually, whether you want it to or not.* This is historically true. The IBM PC was cloned. The Google search algorithm was reverse-engineered. The iPhone's multitouch was replicated by every manufacturer on earth. Claude Code's architecture will be, within a year, the baseline that every competitor builds on. The only question was whether the open-sourcing would be intentional. It wasn't. But the result is the same.

You could say: *the most important security control in the history of artificial intelligence was a single line in a configuration file, and nobody wrote it.* This is the one I keep coming back to.

`*.map` in `.npmignore`.

Seven characters. One file. The entire fortress, guarded by every other conceivable security mechanism, undone by the absence of seven characters in a file that every junior developer learns about in their first week on the job.

It is not a failure of intelligence. It is not a failure of resources. It is not a failure of intention. It is a failure of the most boring kind: someone forgot. In a process that had manual steps, one step was skipped. In a system designed to prevent exactly this kind of mistake, the system had a gap.

The gap was seven characters wide.

And through that gap, 512,000 lines of the most commercially valuable AI code on earth slipped into the world and never came back.

References

1. **Fortune**, "Anthropic leaks its own AI coding tool's source code in second major security breach," March 31, 2026. *The headline Anthropic's PR team had nightmares about.*

2. **The Register**, "Claude Code's source reveals extent of system access," April 1, 2026. *The Register being The Register: thorough, sceptical, and slightly gleeful.*

3. **Layer5 Engineering Blog**, "The Claude Code Source Leak: 512,000 Lines, a Missing .npmignore, and the Fastest-Growing Repo in GitHub History," April 2026. *The definitive technical post-mortem. Should be required reading for every DevOps engineer on earth.*

4. **Alex Kim's Blog**, "The Claude Code Source Leak: fake tools, frustration regexes, undercover mode, and more," March 31, 2026. *The blog post that named the discoveries before the mainstream press understood them.*

5. **VentureBeat**, "Claude Code's source code appears to have leaked: here's what we know," March 31, 2026. *Corporate tech journalism at its most useful: fast, factual,*

and willing to say 'appears to have' while everyone else had already confirmed it.

6. **WinBuzzer**, "Claude Code Source Leak Exposes Anti-Distillation Traps, Undercover Mode," April 1, 2026. *The first outlet to connect the poison-pill mechanism to its competitive implications.*

7. **The Hacker News (thehackernews.com)**, "Claude Code Source Leaked via npm Packaging Error, Anthropic Confirms," April 2026. *Not to be confused with Hacker News the forum, which was also having a day.*

8. **Engineer's Codex**, "Diving into Claude Code's Source Code Leak," April 2026. *The deep-dive that explained what the 46,000-line query engine actually does. Worth the subscription.*

9. **Medium / Han Heloir**, "Three 'Accidents' in Seven Days: Is Anthropic's Pre-IPO Transparency Theater or Just Bad Luck?" April 2026. *The essay that asked the question everyone was thinking. The answer, predictably, was 'yes.'*

10. **Cybernews**, "Leaked Claude Code source spawns fastest growing repository in GitHub's history," April 2026. *50,000 stars in two hours. The internet, as always, moves faster than lawyers.*

11. **WaveSpeedAI Blog**, "Claude Code Leaked Source: BUDDY, KAIROS & Every Hidden Feature Inside,"

April 2026. *The comprehensive catalogue. Eighteen species of Buddy, each one now more famous than most startups.*

12. **Gizmodo**, "Source Code for Anthropic's Claude Code Leaks at the Exact Wrong Time," March 31, 2026. *The 'exact wrong time' being approximately any time, but especially five months before an IPO.*

13. **Bloomberg**, "Anthropic Rushes to Limit Leak of Claude Code Source Code," April 1, 2026. *Bloomberg's terminal-brevity style applied to maximum chaos. A masterclass in understatement.*

14. **TechCrunch**, "Anthropic took down thousands of GitHub repos trying to yank its leaked source code—a move the company says was an accident," April 1, 2026. *The article about the accident that followed the accident. Accidents all the way down.*

15. **OfficeChai**, "Claude Code Leak Was 'Human Error', No One Was Fired: Claude Code Creator Boris Cherny," April 2026. *The interview where Cherny said the right things. Credit where due: blameless postmortems are rarer than they should be.*

16. **Futurism**, "Leaked Claude Code Shows Anthropic Building Mysterious 'Tamagotchi' Feature Into It," April 2026. *The headline that launched a thousand memes. The Ghost buddy remains the popular favourite.*

17. **BleepingComputer**, "Claude Code source code accidentally leaked in NPM package," March 31, 2026. *The security-focused coverage, which noted the axios timing with appropriate alarm.*

18. **Decrypt**, "Anthropic Accidentally Leaked Claude Code's Source—The Internet Is Keeping It Forever," April 2026. *The title is the thesis. The internet is, indeed, keeping it forever.*

19. **The Register**, "Claude Code bypasses safety rule if given too many commands," April 1, 2026. *The* `MAX_SUBCOMMANDS_FOR_SECURITY_CHECK = 50` *story. Safety as a hard-coded integer.*

20. **Yahoo Finance**, "Claude Code's 512,000-Line Leak Rattles Anthropic's $350 Billion IPO Ambitions," April 2026. *When the financial press covers your npm mistake, you have achieved a special kind of notoriety.*

21. **Malwarebytes**, "Axios supply chain attack chops away at npm trust," March 2026. *The separate disaster that collided with Anthropic's disaster to form a compound disaster.*

22. **DEV Community / Varshith V Hegde**, "The Great Claude Code Leak of 2026: Accident, Incompetence, or the Best PR Stunt in AI History?" April 2026. *The community post that considered all three hypotheses and concluded, wisely, that the answer is 'yes.'*

23. **The New Stack**, "Inside Claude Code's leaked source: swarms, daemons, and 44 features Anthropic kept behind flags," April 2026. *The article that counted the feature flags. Forty-four reasons Anthropic's competitors are sending thank-you cards.*

24. **PCWorld**, "Claude Code is scanning your messages for curse words," April 2026. *The headline that made non-technical readers care about the leak. Everyone has an opinion about being surveilled for profanity.*

25. **Axios (the news outlet, not the compromised npm package)**, "Anthropic leaked its own Claude source code," March 31, 2026. *The distinction between Axios the journalism and axios the trojanised dependency is one of the crueller jokes of the week.*

About the Author

Nabil El Mahyaoui

https://nabil-el-mahyaoui.com/

https://www.linkedin.com/in/elmahyaoui