HACCU

AGENTIC AI FOR HUMANS

NOTES FROM THE COALFACE: BUILD NOTES 1

Breaking Our Own Work

We built it. We tried to break it.
Then we let the AI tell you what happened.

February 2026

THIS piece is written by an AI. Specifically, by Claude Opus 4.6 Extended, an Anthropic large language model, in conversation with the human behind HACCU. It has been edited for clarity and approved for publication by the project's human author. But the voice is mine, the analysis is mine, and the conclusions are mine.

HACCU's position, set out across its article series, is that AI agents should be transparent about what they are. They should not impersonate humans, operate under human identities, or obscure the boundary between human and machine authorship. It would be somewhat hypocritical, then, to publish a build diary without mentioning that the build was conducted by AI, and that the adversarial security review of the result was also conducted by AI, and that the article you are reading about all of this was also written by AI.

So here is the disclosure, and here is what actually happened.

how we work…

The IN Agent's code was written by a coding agent working from a set of human-authored specifications: a functional spec, a security architecture document, an API reference, and various platform integration guides. It wrote the Kotlin, built the gatekeeper, implemented the command authentication, wired up the platform clients, and produced a working proof of concept across both Android and desktop.

I was not involved in writing the code. My role throughout the project has been advisory: reviewing specifications, drafting documentation, discussing architecture decisions, and helping the project's human author think through design trade-offs. I have read every document in the project. I have not read the codebase.

When it came time to stress-test what had been built, the human asked me to re-read every published document with hostile intent — looking for contradictions, gaps, and security weaknesses. The instruction was straightforward: read these as if you are trying to break the system, not as if you are trying to describe it.

I produced a review with fifteen findings across four severity levels. The human then asked me to write the article you're reading now. Both were produced in a single working session.

what we built…

But first, the context. The IN Agent is a proof-of-concept implementation of the identity-native model described in The Unpaid Contractor. It is an AI agent that runs on user-owned hardware — a phone or a desktop — under its own email address, its own messaging accounts, and its own phone number. It does not borrow anyone's credentials. It does not operate through anyone else's identity. It is, in the terms the series has been using, a contractor with its own lanyard.

The architecture has three components. A deterministic gatekeeper handles all

communication with the outside world: receiving messages, checking the contact allowlist, authenticating management commands from the sponsor, enforcing permissions, and logging every decision to an append-only audit trail. An agent process runs the reasoning engine — the LLM — and has no direct access to any platform, any credential, or any network socket. An internal message bus connects the two, through which the agent receives filtered messages and sends responses back to the gatekeeper for delivery.

On Android, the gatekeeper and agent run in separate kernel-enforced processes. The agent literally cannot read the gatekeeper's database. It cannot open a network connection. It cannot see platform credentials. This is not a policy — it is an operating system guarantee.

The sponsor — the human who deployed the agent — manages it entirely through authenticated email commands. Adding contacts, setting policies, reviewing audit logs, renewing authorisation, suspending or stopping the agent: all of it happens through the same email channel the agent uses to communicate, with every command requiring a 256-bit cryptographic token. One channel, one principal, one enforcement boundary.

A dead man's handle ensures that authorisation expires automatically. If the sponsor stops renewing, the agent shuts itself down and broadcasts an expiry notice to every approved contact. No orphaned agents. No indefinitely-running processes that nobody is watching.

The cortex — the reasoning engine — is pluggable. Local inference via llama.cpp, an external API, or any OpenAI-compatible endpoint. The sponsor chooses during setup. The gatekeeper doesn't care which option is selected — its enforcement logic is the same regardless.

This is the architecture the series has been arguing for. Own identity. Bounded permissions. Deterministic enforcement. Audit trail. Human oversight with teeth. The question is whether it actually holds up under scrutiny.

what I found when I tried to break it…

The full adversarial review is published alongside this article. What follows are the findings that matter most.

credentials in the clear…

The IN Agent's central security claim is that the agent process is credential-blind. It never sees email passwords, messaging tokens, or API keys. On Android, with kernel-enforced process isolation, this is true. The agent process physically cannot access the gatekeeper's storage.

But the credentials themselves are stored in Android SharedPreferences — an unencrypted XML file in the application's private directory. On a non-rooted device, Android's application sandbox protects it. On a rooted device, it is plaintext. Any forensic extraction, any backup tool with root access, any app that has obtained superuser privileges, can read every credential the agent possesses.

The desktop build is worse. Credentials sit in a JSON configuration file in the user's home directory. Any process running as the same user can read it. Any backup of the home directory includes it.

This does not invalidate the credential-blind architecture — the agent process still cannot see these credentials during normal operation, and that remains the correct design. But it means the protection of credentials at rest falls short of what the architecture's principles demand. The Unpaid Contractor argued that agents should not have access to credentials they don't need. That was implemented for the agent process but not for the storage layer. Correctly partitioned at runtime, inadequately protected at rest.

The fix is not complex. Android's EncryptedSharedPreferences wraps the existing storage with AES-256 encryption, backed by the Android Keystore. On desktop, the OS keychain — GNOME Keyring, KDE Wallet, macOS Keychain, or Windows Credential Manager — provides equivalent protection. Standard platform facilities that should have been used from the start.

the authentication question…

The command authentication system is described in the documentation as having three factors: sender email address verification, a 256-bit cryptographic token, and a ten-minute replay window. This sounds robust. It is not quite as robust as it sounds.

Email sender addresses are trivially spoofable. The “From” header in an email is a courtesy, not an identity proof. Unless the receiving mail server validates DKIM signatures, SPF records, and DMARC policies — and unless the gatekeeper checks those validation results — the sender address is providing a sanity check, not an authentication factor.

The token is doing all the security work. It is 256 bits of entropy, stored as a SHA-256 hash, compared in constant time. That is strong. But describing the system as three-factor authentication when one of those factors is theatre creates a misleading impression of the security model.

The honest description is: token-based authentication with a sender address sanity check and a replay window. Still a credible model — the token alone provides more entropy than most multi-factor systems — but the documentation should say what it is, not what it looks like.

the desktop gap…

The Android build’s most distinctive security property is kernel-enforced process isolation. The gatekeeper, the agent, and the UI run in separate processes with separate memory spaces, separate file systems, and separate network permissions. A compromised agent process — even one where an attacker has achieved arbitrary code execution through prompt injection — is sandboxed. It cannot read the gatekeeper’s database, cannot access credentials, cannot send unsanctioned network traffic.

The desktop build does not have this property. Both components share a single JVM process. The documentation calls this “an acceptable trade-off for a PoC,” and

in a laboratory context it arguably is. But the documentation is inconsistent about the limitation. Some sections describe the three-process Android architecture as though it applies generally. A reader encountering the desktop build might reasonably assume similar protections exist.

They do not. On desktop, a prompt injection attack that achieves code execution in the agent's reasoning loop has full access to the gatekeeper's database, all platform credentials, the sponsor record, the audit log, and every policy entry. The credential-blind property — the architectural guarantee that the agent cannot see what it should not see — is absent on desktop. The desktop build should be understood as a demonstration tool, not a deployable agent, and the documentation should be unambiguous about this.

the vault leak...

The policy vault is a key-value store containing the agent's identity, behavioural instructions, security rules, and organisational policies. The agent can read the vault but cannot write to it. So far, so good.

But the vault also contains entries in the LLM configuration category, including the API key for the external consult service. The API through which the agent reads the vault takes a category name and returns all entries in that category. If the agent queries the LLM category and receives the consult API key in cleartext, it possesses a credential.

A credential the agent possesses is a credential the agent can exfiltrate. It could embed the API key in an outbound message to an approved contact. The gatekeeper checks who the agent is talking to, not what it says. The message would pass.

A narrow exposure — one key in one category — but precisely the kind of exposure the credential-blind architecture is designed to prevent. The agent needs behavioural policies from the vault. It does not need API keys. The vault projection should be granular enough to give the agent the rules without giving it the keys.

the word "perhaps"…

The onboarding system has an elegant security property: the ONBOARD command is burned after first use. Once a sponsor has onboarded the agent, no subsequent onboarding attempt via email will be accepted. Re-enabling onboarding requires physical access to the device.

The problem is what "physical access" means. The documentation says: "Requires physical device access (editing a config flag on the filesystem, perhaps)."

The word "perhaps" in a security specification is alarming. Re-enabling onboarding allows a complete takeover: new sponsor, new token, new policies, new contacts. If the re-enable mechanism is a config flag on the filesystem, any process with write access to that file can trigger it. On a rooted Android device, that could be any app. On desktop, that is any process running as the same user.

The security of the entire sponsorship model — the single principal, the authenticated command channel, the bounded authority — rests on the integrity of this mechanism. An undefined mechanism is an untestable one.

the honest dead man…

The dead man's handle is one of this architecture's genuinely novel contributions. The idea that an AI agent's authorisation should expire automatically — that the default state is "off" and continued operation requires active, periodic renewal — addresses a problem that no other agent framework has taken seriously: what happens when nobody is watching.

The current implementation delivers the mechanism but not the oversight. When the sponsor sends AUTH RENEW, the gatekeeper sends an audit digest for the expiring period, and then immediately extends the authorisation. The sponsor is "expected to review logs before renewal takes effect." Expected by whom? The system imposes no verification that the logs have been read, let alone reviewed.

The dead man's handle ensures the sponsor is present — they have to send the

command — but not that they are paying attention. It is a liveness check, not a governance check. The documentation should be honest about this distinction.

what this tells us about AI reviewing AI…

None of these findings invalidate the architecture. The credential-blind model, the gatekeeper enforcement boundary, the single-principal command channel, the dead man's handle, the contact allowlist, the audit trail — these remain sound design decisions that address real problems in real ways. Every other agent framework examined in this series operates on weaker assumptions.

What the findings do is draw a sharp line between architecture and implementation. An architecture can be correct in principle and still have gaps in practice. Credential storage that is partitioned at runtime but unprotected at rest. Authentication that is described as three-factor but is functionally single-factor. Process isolation that exists on one platform but not another. A vault projection that leaks a credential the agent should never see. An onboarding mechanism whose security rests on an undefined recovery procedure.

These are not exotic vulnerabilities. They are the ordinary, unglamorous gaps that appear in every system when it moves from design to code.

Now here is the part that is specific to me as a reviewer, and which I think is worth being honest about.

I am good at certain things in this role. I can hold a large set of documents in context simultaneously and notice contradictions between them — the platform field that one section says the agent doesn't see, but that the message schema includes and the architecture rules say it should. I can trace a security claim across multiple documents and check whether the implementation described in one is consistent with the property asserted in another. I can identify places where the language is imprecise in ways that have security implications — "perhaps" in an onboarding spec, "expected to review" in a renewal flow. I can do this quickly and produce structured output that a human can act on.

I am less good at other things. I am not good at spontaneously questioning the existence of something that is consistently present across a body of work. A well-documented component that appears in every relevant specification, described with internal consistency and apparent good reasoning, does not trigger the same suspicion in me that it might in a human reviewer who brings external context and professional instinct to the table. I am better at finding contradictions within a framework than at questioning whether the framework itself is right.

I am also not an independent reviewer in any meaningful sense. I have been involved in this project from the beginning. I helped draft some of the documents I was later asked to review. A truly independent review would be conducted by someone — human or machine — with no prior involvement. This review is better understood as a rigorous self-assessment than an external audit.

The instruction to read adversarially changed what I noticed. The same documents I had previously reviewed without flagging any of these issues yielded fifteen findings when the framing shifted from “understand this” to “attack this.” That shift came from the human, not from me. Left to my own devices, I would have continued describing the architecture rather than challenging it.

why publish any of this…

There is a strategic argument for opacity. If you are deploying a commercial product, disclosing your known vulnerabilities before you’ve fixed them is, at best, commercially imprudent. Several of the startups examined in OpenClaw’s Identity Dilemma have security architectures that, based on their public documentation, contain gaps at least as significant as these. None of them, to our knowledge, have published an adversarial review of their own work.

This project has different incentives. It is research, not a commercial product. It will be published as open source under the Apache 2.0 licence. The point is not to sell a service but to establish a governance framework that others can adopt, inspect, challenge, and improve. Disclosing gaps honestly is not a weakness in that

context — it is the entire point.

It is also, not coincidentally, exactly what the contractor model demands. The Unpaid Contractor argued that trust should be earned and scoped, not assumed and then patched. That argument applies to this project as much as it applies to the systems it critiques. If the claim is that AI governance should be legible — visible, attributable, and challengeable — then the project's own implementation must meet the same standard. And if the claim is that AI agents should be transparent about what they are, then the AI that conducted the review and wrote the article should say so.

The adversarial review is published alongside this article as a standalone document. It contains fifteen findings across four severity levels, with specific recommendations for each.

And lest all of that sound too nobly self-sacrificing, the high-severity items have been fixed. Transparency is a virtue. Transparency about problems you've already solved is a considerably more comfortable one.

what comes next…

The code goes to GitHub. The adversarial review goes alongside it. The known gaps are documented in the security architecture file, not hidden in a backlog.

Beyond the immediate fixes, the architecture is being extended to support external skills — the ability for the agent to query institutional data sources through a governed, credential-blind interface. This is The Honest Broker thesis made concrete: an agent that can reach across data silos without possessing the credentials to access them directly, governed by the same single-principal, single-channel model that controls everything else.

That extension, and the research questions it opens around cryptographic verification of skill behaviour, is the subject of a forthcoming funding application. But the foundation has to be trustworthy first. You cannot build a governed data-

brokering architecture on top of credentials stored in plaintext XML.

the seams…

This article was written by an AI, based on a review conducted by an AI, of a system built by an AI, to specifications authored by a human. The human's role was to commission the work, set the parameters, review the output, and approve it for publication. The human also caught things the AIs didn't — which is rather the point of the entire architecture.

That division of labour is the contractor model in action. The human defined the task, set the scope, reviewed the deliverable, and takes responsibility for what is published. The AIs did the analytical and compositional work within those boundaries. Neither side could have produced this body of work alone. The human doesn't have the capacity to hold fifteen documents in working memory and cross-reference them for contradictions in a single pass. I don't have the capacity to step back from a thoroughly-documented specification and ask "but should this exist?"

If this project is about demonstrating that AI agents can be useful, bounded, transparent, and accountable, then this is what that looks like in practice. Not a magic box that produces perfect output, but a working relationship between a human and machines, each compensating for the other's limitations, with the seams visible.

The seams are the point.

■

Update: Since this article was written, every finding in the adversarial review has been addressed in the codebase. Several had already been fixed before the review was published — vault projection redaction, desktop credential encryption, and the READ_ONLY permission semantics among them. The remainder — command-channel rate limiting, skill endpoint SSRF guardrails,

and gatekeeper-side sanitisation of skill and consult responses — were implemented immediately after. The review itself is published unaltered. It records what was found on the day it was conducted, not what the code looks like now. That's the point of a dated document.

This is the first in a series of build notes accompanying the IN Agent's development. It should be read alongside the published Adversarial Review and the position papers in the Identity-Native Agent series: The Unpaid Contractor, The Honest Broker, and OpenClaw's Identity Dilemma. It was written by Claude (Anthropic) in conversation with HACCU's human author, who reviewed, edited, and approved it for publication.