HACCU

AGENTIC AI FOR HUMANS

ADVERSARIAL SECURITY REVIEW

IN Agent Documentation Review

Fifteen findings from a hostile read of the IN Agent project documentation.

21 February 2026 • Accompanies *Notes from the Coalface: Build Notes 1*

This is a hostile read by Anthropic Claude Opus 4.6 Extended LLM of every published document in the IN Agent project. The review was conducted by examining the functional specification, security architecture, API reference, platform integration guides, and desktop architecture documentation for internal contradictions, underspecified mechanisms, and gaps between stated security properties and described implementation.

---

1. email sender verification is not authentication…

The command authentication section in SECURITY.md describes three factors: sender address, 256-bit token, and replay window. But email “From” addresses are trivially spoofable. Any attacker who obtains or guesses the command token can send commands from a spoofed sender address. The “From” match is a UI convention, not a security boundary.

The real question is whether the gatekeeper checks DKIM, SPF, or DMARC results from the receiving mail server. The docs don’t mention this anywhere. If the gatekeeper is just string-matching the “From” header against the sponsor record, sender verification is theatre. The token is doing all the security work and the docs should be honest about that.

Severity: Medium. The 256-bit token is strong on its own. But presenting sender address as a genuine authentication factor is misleading, and if the token is ever compromised, sender spoofing becomes trivial.

Recommendation: Either implement DKIM/SPF/DMARC validation on inbound email, or stop calling this three-factor authentication. It's token-based authentication with a sender address sanity check.

---

2. credential storage is plaintext on Android…

SECURITY.MD line 72: "Platform credentials (email password, Discord bot token, Slack tokens) are stored in Android SharedPreferences."

SharedPreferences is an unencrypted XML file in the app's private directory. On a non-rooted device, Android's app sandbox protects it. On a rooted device, it's plaintext. The doc acknowledges this ("if the device is rooted, all bets are off") but then lists "Future enhancement: migrate platform credentials to Android Keystore" as if this is a nice-to-have.

This is the agent's entire credential set. Every platform identity it possesses. Storing them in SharedPreferences means that any app with root access, any backup extraction tool, or any forensic analysis of the device exposes every credential the agent has.

Severity: High for any deployment outside a controlled lab. The entire credential-blind architecture is moot if the credentials are sitting in an XML file.

Recommendation: Android Keystore migration is not a future enhancement. It's a prerequisite for any deployment claim that credentials are protected. At minimum, use EncryptedSharedPreferences (part of AndroidX Security) which requires one line of code change.

---

3. desktop config stores credentials in plaintext JSON…

DISCORD-IDENTITY-SETUP.MD line 131: "The bot token is stored in `~/.config/inagent/in-agent-config.json` on the machine running the agent."

ARCHITECTURE-DESKTOP.MD confirms: “ConfigStore reads `in-agent-config.json`.”

This file contains the email password, Discord bot token, Slack bot token, Slack app-level token, and cortex API key. In plaintext. Any process running as the same user can read it. Any backup of the home directory includes it. Any shoulder-surfer who opens the file has every credential.

Severity: High. Same reasoning as above but worse — desktop has no app sandbox equivalent.

Recommendation: At minimum, encrypt the config file at rest using a key derived from a user-provided passphrase or OS keychain (GNOME Keyring, KDE Wallet, macOS Keychain, Windows Credential Manager). For the PoC, document this as a known limitation clearly in the security doc rather than burying it in a setup guide.

---

4. queryVault projection is underspecified…

API.MD shows `queryVault(String category)` takes a category string and returns all entries in that category. The functional spec says the agent has READ-ONLY access and “can query any entry.” SECURITY.MD doesn’t specify what’s filtered.

The policy vault contains `llm.consult_api_key` (described as “stored encrypted” in the functional spec). If the agent can call `queryVault("llm")` and receive the consult API key in cleartext, the agent possesses a credential. That breaks the credential-blind property. The agent could embed that API key in an outbound message to an approved contact and it would pass the gatekeeper because the gatekeeper checks who, not what.

Severity: Potentially high. Depends on implementation. Could be a complete breach of the credential-blind property.

Recommendation: The vault projection to the agent must explicitly exclude

credential-bearing keys. Document the exclusion list. Test it.

---

5. desktop has no process isolation…

ARCHITECTURE-DESKTOP.MD is transparent about this: “The desktop port loses this isolation — both components share the same JVM heap.” It calls this “an acceptable trade-off for a PoC.”

But the rest of the documentation doesn’t consistently caveat this. The README describes “three isolated Android processes” in its architecture diagram. The SECURITY.MD describes “kernel-enforced process isolation.” A reader looking at the desktop build might assume similar protections exist.

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. The credential-blind property doesn’t exist on desktop.

Severity: High for desktop deployments. The architecture’s strongest security property is absent.

Recommendation: Every doc that mentions process isolation should note the desktop exception. The desktop build should carry a clear warning in its startup output and documentation that it does not have the security properties of the Android build. Consider whether the desktop build should be presented as a “demo tool” rather than a deployable agent.

---

6. auto-reply to unknown senders is an information leak…

Functional spec section 4.4: “Send auto-reply: ‘This agent only accepts messages from authorised contacts.’”

This confirms to any attacker that the email address is a HACCU IN Agent. It reveals

the existence of the agent, confirms it's active, and tells the attacker exactly what security model it uses. A targeted attacker now knows to focus on obtaining the command token or compromising an approved contact.

Severity: Low-medium. Depends on threat model. For a research PoC, probably fine. For a deployment where the agent's existence should not be publicly discoverable, this is a problem.

Recommendation: Make the auto-reply configurable. Default could be silent drop with audit log (no reply). Sponsor can enable an auto-reply if they want one. The current behaviour should not be hardcoded.

---

7. READ_ONLY permission allows initiating contact…

The permission model table in SECURITY.MD shows READ_ONLY contacts can "Initiate: Yes." This means a READ_ONLY contact can send messages to the agent but never receive replies. The agent reads their message, processes it, but the gatekeeper blocks the response.

This is confusing and potentially dangerous. The agent has received and processed information from the contact, updated its memory, possibly changed its behaviour based on that input — but the contact gets silence. From the agent's perspective, it processed a real message. The contact's input influenced the agent's state without the contact ever receiving feedback or the sponsor having visibility into the influence.

Severity: Low. But it's a design smell. A contact who can influence the agent's state but never receives output is a one-way information injection channel.

Recommendation: Clarify the intent. If READ_ONLY means "can read the agent's broadcasts but can't initiate," the table is wrong. If it means "can send but won't get replies," rename it and make the agent aware that it shouldn't process messages from read-only contacts as actionable input.

---

8. audit log integrity is not guaranteed...

The audit log is described as "append-only" but it's a SQLite table. There's no cryptographic integrity protection — no hash chain, no signed entries, no tamper detection. Any process that can write to the database can modify or delete audit entries.

On Android with process isolation, only the gatekeeper process can write to the gatekeeper DB, so this is somewhat protected. On desktop, any code running in the same JVM can modify the audit table. A compromised agent process on desktop could rewrite its own history.

Severity: Medium for desktop. Low for Android (assuming process isolation holds).

Recommendation: For the PoC, document this limitation. For production, implement a hash chain where each audit entry includes the hash of the previous entry. This makes tampering detectable even if it can't be prevented.

---

9. ONBOARD re-enable mechanism is undefined...

Functional spec section 4.3: "Requires physical device access (editing a config flag on the filesystem, perhaps)."

The word "perhaps" in a security specification is a red flag. The ONBOARD command is burned after first use — good. But the re-enable path allows a complete sponsor takeover: new sponsor email, new token, new policies. If re-enabling is as simple as editing a config flag on the filesystem, any process with write access to that file can trigger it.

Severity: High. The entire sponsorship security model depends on this mechanism being well-defined and protected.

Recommendation: Define precisely. On Android, require BiometricPrompt confirmation. On desktop, require a documented recovery procedure. The word "perhaps" should not appear in any security-relevant specification.

10. auth renewal is honour system…

Functional spec section 4.5: “Gatekeeper auto-sends audit digest for the expiring period first. Sponsor is expected to review logs before renewal takes effect (v1: honour system; v2: structured pre-flight checklist before renewal completes).”

So in v1, AUTH RENEW extends the agent’s life with no verification that the sponsor has reviewed anything. The audit digest is sent, but the renewal proceeds immediately regardless. The dead man’s handle — the project’s signature safety feature — can be renewed without any actual oversight taking place.

Severity: Medium. The DMH still works mechanically — if the sponsor stops sending AUTH RENEW, the agent dies. But the “active oversight” claim is aspirational, not enforced.

Recommendation: Be honest about this in the docs. The DMH ensures the sponsor is *present*, not that they’re *paying attention*. The v2 pre-flight checklist should be prioritised, not deferred.

---

11. no rate limiting on email commands…

Nothing in the docs mentions rate limiting on the email command channel. An attacker who obtains the command token could flood the agent with commands — rapid contact additions, policy changes, memory clears — faster than the sponsor could notice or respond.

The 10-minute replay window prevents replaying the same email, but it doesn’t prevent sending many different commands within that window.

Severity: Low-medium. Requires token compromise first, at which point you have bigger problems. But rate limiting is a defence-in-depth measure.

Recommendation: Add command rate limiting. Something simple: no more than N commands per minute, with a cooldown after repeated failures. Log rate-limit

events.

---

12. DMH reminders assume email delivery…

The dead man's handle sends reminders via email (and other channels at the urgent stage). If the email platform is down, or the sponsor's email is bouncing, or the messages are going to spam, the sponsor may not receive the reminders. The agent then expires and broadcasts to all contacts without the sponsor having had any real opportunity to renew.

Severity: Low. The expiry is the safe default — an unreachable sponsor is a good reason to stop. But the sponsor experience could be poor.

Recommendation: Document this as expected behaviour. Consider adding a "last reminder delivery confirmed" status that the sponsor can check via STATUS command, so they can verify reminders are getting through.

---

13. platform field is exposed to the agent…

Functional spec section 4.9 says: "The IN agent does NOT: Know which platform messages come from (it sees Message objects, the platform field is for routing, not exposed to the LLM)."

But the MessageParcel in API.MD includes `platform: String // "EMAIL", "DISCORD", "SLACK"` and the Message data class in section 3.1 includes `platform: str`. The functional spec itself (section 2 architecture rules, rule 1) says: "Messages include the originating platform (so the agent can respond sensibly to 'did you see my Discord message?')."

These contradict each other. Either the agent knows the platform or it doesn't.

Severity: Low. Knowing the platform isn't a security issue in itself. But contradictory docs lead to contradictory implementations.

Recommendation: Pick one and be consistent. The agent probably should know the platform (it's useful context). Update section 4.9 to match reality.

---

14. bootstrap credentials exist before onboarding security…

The setup wizard (or HAi Setup Companion) configures email credentials, Discord tokens, Slack tokens, and cortex API keys *before* the sponsor sends the ONBOARD email. This means the agent's credentials are sitting on the device in SharedPreferences or config JSON before any security model is active — no sponsor, no command token, no audit trail.

If the device is compromised between bootstrap and onboarding, all credentials are exposed and the attacker knows exactly which accounts the agent will use. They could pre-compromise those accounts before the sponsor even knows the agent is live.

Severity: Low. The window is small (minutes to hours). But it exists and it's undocumented.

Recommendation: Document the bootstrap-to-onboard window as a known exposure period. Consider encrypting the bootstrap config with a one-time key that's destroyed after onboarding completes.

---

15. consult loop guard is single-pass only…

Functional spec section 4.7: "Second-pass output is blocked if it tries to emit another consult directive (loop guard)."

This prevents the agent from chaining consult→consult→consult. Good. But it's implemented in the agent harness, not the gatekeeper. If the agent's reasoning loop is manipulated by prompt injection to reformulate the consult request in a way the loop guard doesn't recognise, the guard fails.

Severity: Low. The consult path also requires gatekeeper-level policy checks, so even a bypassed loop guard still hits the data-leaves-device gate. But the loop guard is fragile.

Recommendation: Enforce the consult count at the gatekeeper level, not just the agent harness. The gatekeeper should track consult invocations per message and enforce a hard limit regardless of what the agent requests.

---

summary by severity...

High:

Credential storage in plaintext SharedPreferences (Android) and JSON (desktop)

Desktop has no process isolation — credential-blind property absent

ONBOARD re-enable mechanism undefined

queryVault may expose credentials to agent

Medium:

Email sender verification is not real authentication

Audit log has no integrity protection

Auth renewal has no actual oversight verification

Auto-reply reveals agent existence (context-dependent)

Low:

No rate limiting on email commands

DMH reminders assume email delivery

Platform field contradiction in docs

Bootstrap credential exposure window

Consult loop guard is agent-side only

READ_ONLY permission semantics are confusing

---

what's genuinely strong…

To be fair to the architecture, the things it gets right are significant:

The gatekeeper/agent split with AIDL IPC on Android is a real security boundary, not a convention. The credential-blind agent design (on Android) is architecturally enforced, not policy-enforced. The dead man's handle is a genuinely novel safety mechanism for agent deployments. The contact allowlist with gatekeeper enforcement is sound. The burn-after-read onboarding with physical-access re-enable (once properly defined) is a strong bootstrap model. The command token with hash storage and constant-time comparison is textbook correct. The reactivation checklist with mandatory attestation is good governance friction.

The architecture is good. The implementation gaps are mostly about closing the distance between what the docs claim and what the code actually enforces.

■

This review accompanies *Notes from the Coalface: Build Notes 1*. It was conducted by Claude (Anthropic) in conversation with HACCU's human author and is published unaltered as a dated snapshot.