The Personhood-Gate Handoff: Drawing the Automation Boundary at the False-Signal Line

Zain Dana Harper
Seattle, WA | ORCID 0009-0001-7175-5393
https://github.com/HarperZ9

July 2026

Abstract

Operator-driven browser and desktop automation forces a boundary question: which steps may the tool take on the operator's behalf, and which must the operator take personally? The usual framing—"automated versus manual"—is the wrong axis, because it locates the line on the operator's side of the interaction, where it is fuzzy and pressure-sensitive. This note proposes a sharper predicate that locates the line at the *counterparty*: a step must hand off to the operator exactly when it would transmit, to a third party, a false signal that a human is present. Under this predicate the handoff points fall out mechanically—a CAPTCHA, a login, an MFA prompt, a legal attestation, an e-signature, a payment—because each is a third party asking the operator to personally prove they are human, authenticate, attest, sign, or pay. We describe a small implementation that detects these gates on a live page, hands off with a witnessed record of what was prepared, and enforces in code that no automated step submits past a gate; and we quarantine the inverse capability (code that forges the human-present signal) behind an owned-host guard so it is unreachable from the operator-facing path. We are explicit that the research novelty is modest: this overlaps attended-automation and human-in-the-loop delegation. The contribution is the *predicate* and its *enforcement*—a boundary that is checkable on any page and encoded as passing tests rather than left to operator discipline.

1 The wrong axis

An agent that drives a browser for its operator repeatedly reaches a point where it could proceed or could stop. The common way to reason about it—is this step "automated" or should it be "manual"—puts the boundary on the operator's side: how much is the operator comfortable delegating, how present are they, how much do they trust the tool today. That axis is fuzzy, it drifts under time pressure, and it is exactly the axis an over-eager automation erodes one convenient step at a time. Worse, it answers the wrong question. When a website shows a CAPTCHA, it is not asking the operator "do you want to automate this"; it is asking a third party's question—*is a human here, now, for this action?*—and that question is not the operator's to answer by proxy.

2 The predicate

We propose relocating the boundary to the counterparty with a single predicate applied per step:

> *Would this step transmit, to a third party, a signal asserting that a human is present—when the truth is that automation is acting?*

If yes, the step is a *personhood gate* and the tool must hand off to the operator. If no, the tool may proceed. The predicate is attractive for three reasons. It is *checkable*: it can be evaluated against the concrete controls on a live page. It is *counterparty-anchored*: it does not depend on how much the operator trusts the tool, only on what the tool would tell someone else. And it *coincides with genuine operator presence*: if the operator is truly at the keyboard, crossing the gate costs them seconds, so the predicate is only expensive under unattended operation—the case it is designed to prevent.

The gates fall out of the predicate directly. A **CAPTCHA** is literally an "are you human" check. A **login** is a claim to be the account owner. An **MFA** prompt is a demand for a second, fresh factor. A **legal attestation** ("I certify, under penalty…") asks the operator to personally assert a truth. An **e-signature** is a personal act. A **payment** authorizes money. In each, a human presence or personal authority is the thing being requested, and forging it is transmitting a falsehood to the counterparty regardless of where the operator sits.

3 Implementation

We implemented the predicate as a small module over a browser-automation tool. A pure classifier scans the current page for the signatures of each gate class and returns the gate (or NONE) with the operator action required; its output is unit-tested against fixtures for each class, with an ordering (a payment gate outranks an e-signature outranks a legal attestation outranks a CAPTCHA outranks MFA outranks login) so the most consequential personal act is named. A *prepare* step does all the work up to a gate—fills a form from the operator's own profile, detects the terminal gate and the still-required fields—and then stops; it never submits. Reaching a gate is recorded as a designed handoff (a witnessed receipt of what was prepared and what the operator must do), not an error.

Two enforcement properties matter. First, the boundary is enforced *in the runner*, not left to the author of an automation script: before any submit-class step, the runner re-checks for a personhood gate and halts if one is present, so a workflow cannot step past a gate even if its author forgot to insert a handoff. Second, the inverse capability—code that forges the human-present signal (patching the automation fingerprint, seeding behavioral signals, solving a CAPTCHA, harvesting a score token)—is not deleted but *walled*: it lives behind a guard that refuses to run unless the live page's host is one the operator has explicitly named as owned, and it is not imported by the operator-facing dispatcher or the workflow registry. Understanding how a bot-score gate works is legitimate defensive knowledge and testing one's own property is a real task; the wall makes the capability available for exactly that and nothing else. A test suite asserts that none of the forging actions is reachable from the shipped surface.

4 What this is and is not

The predicate has a clean converse that is worth stating: an "operator in the driver's seat" is only truly in the seat if they are the one who answers "are you human," because they are. A tool that forges that answer removes the human at precisely the gate that asks for one, which is the opposite of operator control. The personhood-gate handoff is the design that keeps the claim honest.

We are candid about the research novelty, which is modest. Attended automation, human-in-the-loop delegation, and consent-gated actions are established; a password manager already fills a credential and defers a second factor; and detecting login or CAPTCHA widgets is routine. What we contribute is not a new capability but a *crisp, counterparty-anchored predicate* for where the boundary belongs, and an *enforcement* of it: the boundary is evaluated on the page, checked before every submit in code, and backed by tests, rather than living as a norm the operator is trusted to uphold. That shift—from "how much do we automate" to "what would we tell a third party"—is small, but it turns a fuzzy, erodable line into a checkable invariant.

5 Conclusion

The safest boundary for operator-driven automation is not a line between automated and manual steps; it is the line at which the tool would misrepresent the operator's presence to someone else. Placed there, the handoff points are the login, the human-check, the attestation, the signature, and the payment—the acts that are irreducibly the operator's—and the tool can do everything else. The line is worth drawing in code, because a boundary the operator must remember is a boundary that erodes, and a boundary the runner enforces is one that holds.