Hands on the Switch: An Off-Host Interlock Architecture and a Working Standard (HWS 0.1) for AI Agents That Write to Physical Hardware

Sawbones Labs Technical Report SL-TR-2026-01 · Version 0.1 · 26 November 2026 · Working paper

Joshua Bryan, Sawbones Labs

26 November 2026

Abstract

AI agents are moving from software writes to writes with physical effect. Recent work describes devices and their limits to an agent; whether and how a given agent write may reach the device remains open. We present an architecture in which the agent never holds the switch. An untrusted planner proposes actions through a Model Context Protocol (MCP) server whose tool list is constant and split by hazard class. A host supervisor derives the facts each action relies on, assigns one of five hazard classes and binds the action to a single-use token. A separate microcontroller, the Interlock trust root, keeps its own rules and clock, runs a challenge-response heartbeat, latches faults that only a panel act clears, and grants a lease for one action digest only after a fresh button press. A bus guard passes only the byte-exact frame of the leased action, once. A taint latch withholds hazardous actions after outside text reaches the agent until a person acts again. We distil the design into Hardware Write Safety 0.1, 56 testable requirements in 13 areas that borrow concepts from functional-safety, machinery, lockout and industrial-security standards and name seven gaps those standards leave for agents. The open Rust implementation (18 packages, 757 test functions; 781 tests pass) is verified in simulation only: nothing has run on hardware, the independent watchdog cut is not built, and 42 of 127 red-team cards are open gaps. We report results to date and a pre-registered benchmark plan that will publish its losses.

Report status. This is a dated, versioned working paper published by Sawbones Labs to record the design and its evidence as of 26 November 2026. It has not been peer reviewed. All code facts refer to commit `0cb8ae5` of the public repository [1]; each is traced to a file or test in Appendix A (cited as [E*n*]). A later version will be submitted to arXiv after hardware testing and the benchmark in Section 6.

1 Introduction

Agents built on large language models already change software state through tools: they open pull requests, edit tickets and update databases. The same tool interfaces reach physical devices just as easily. A Model Context Protocol (MCP) server can wrap a Modbus register, a robot topic or a laboratory instrument in a few lines, and community and research servers do exactly that for robots and industrial protocols [2], [3]; the README of one widely used robot bridge describes no safety controls at all [2]. Anthropic's Model Hardware Standard (MHS) research preview, announced on 27 August 2026, describes devices to agents through a standard driver with read and write commands and a generated reference file that states what a device can measure, what can be adjusted and what safety limits apply; agents reach the driver through MCP, a command line or code [4]. That is the description problem. The enforcement problem remains open: when an agent proposes a write with physical effect, what decides whether it may reach the device, and on what evidence?

Our answer starts from a premise borrowed from security engineering and runtime assurance rather than from prompt design: the planner is untrusted [5], [6]. It can be wrong, it can be persuaded by text it reads [7], and the text it reads can arrive through the physical world, as a device name, an error string or a printed label [8]. MCP's own guidance says tool annotations are hints that clients must treat as untrusted, and that guarantees come from deterministic controls outside the model [9], [10]. A safety design for agent-driven hardware should therefore not ask the agent to be safe. It should make what the agent can energise a property of separate hardware and of a person's hand.

Sawbones Labs is building an agent-operated test bench, an open project whose first deployment is an off-grid site with a solar and battery installation. This report records its safety architecture and the working standard we derived from it.

Contributions.

1. A threat model for agents that write to hardware, covering the untrusted planner, prompt injection through device text, forged and stale facts, replay, social engineering of the approver, a compromised host and fault injection on the safety controller, including an explicit statement that physical access defeats a microcontroller (Section 2).
2. An architecture (Section 3) that combines five hazard classes; a preflight, classify and confirm pipeline with host-derived facts and single-use tokens; trust tiers that content cannot set; leases bound to one action digest and granted by a fresh physical press; an off-host trust root with a challenge-response heartbeat and panel-only fault clearing; a byte-exact bus guard; a taint latch; ground-domain isolation planning; a zero-import plugin sandbox; and an MCP surface whose tool list is constant and split by hazard class, consistent with the stateless MCP revision of 28 July 2026 [11].
3. Hardware Write Safety 0.1 (HWS), a working list of 56 requirements in 13 areas, each with a verification method, mapped to concepts in established standards and to seven gaps those standards leave for agents (Section 4).
4. An open Rust implementation verified in simulation, with honest accounting: test and mutation results, a public red-team catalogue with a status ratchet, a regression-per-bug rule, and an explicit list of what is not built (Section 5).
5. A pre-registered benchmark plan for the MCP surface, including baselines, ablations and a commitment to publish losses (Section 6).

We do not claim that any single mechanism is new. Exact-action approval tokens, question-and-answer watchdogs, Modbus allow-lists and information-flow labels all have precedents, which Section 7 reviews. What we claim is the assembly: an approval that is physically confirmed, bound to one action digest, carried from an MCP agent down to the bytes on the bus wire, and enforced by a controller the host cannot reach, together with a standard that makes each joint testable.

2 Threat model

Assets. People and property exposed to stored energy (a battery bank or inverter driven into fire, shock or short circuit); the owner's equipment (a changed protection limit or a bricked device); the truth of the record (a value that looks measured but is not, and later justifies a write); and privacy [E1].

Trust boundaries. Four rules organise the design [E1]. (i) The trust root trusts nothing the host says: it has its own clock, inputs and rule table; the host may ask and the trust root decides. (ii) The host carries data but does not vouch for it; a fact is as good as the evidence behind it. (iii) The keys that matter live off the host: approval, lease and core-release keys in the trust root, measurement keys in each module, and non-energy approvals on the owner's phone. (iv) The agent proposes; physical rails and signed approvals decide.

Adversaries and failures. We consider the following, in increasing order of capability.

- *Mistaken or injected agent.* The agent errs, or text it read tells it what to do. It can reach every tool its session may call. We assume it can be fully compromised.
- *Prompt injection through the physical world.* A Wi-Fi network name, a Bluetooth name, a decoded packet, text recognised in an image, a spoken phrase or a module's manifest can carry instructions [7], [8]. Fencing such text does not stop a model from being persuaded; the design must bound what a persuaded model can cause.
- *Forged or stale facts.* A lying or reflashed module reports a wrong number; a register map is wrong; a bus has a second master the owner did not know about; a fact true last week is no longer true. In an agent loop, a stale or fabricated "measurement" can later justify a write.
- *Replay and confused deputies.* A captured approval is sent again; an approval for one action is spent on another; the component that turns an approved action into bytes encodes something else; checked bytes change before transmission.
- *Social engineering of the approver.* A display that shows model-written prose invites the human to approve the wrong thing [12]. The approver is part of the attack surface.
- *Malicious or buggy plugin.* A decoder that lies, loops, leaks or asks for more than it needs.

- *Compromised host.* An attacker or bug with root on the host controls the supervisor, the MCP server and the store's node key. The design assumes this case: such an attacker can lie to the agent and corrupt records and can starve the heartbeat (which drops power), but must not be able to energise a rail [E1].
- *Supply chain.* A bad dependency, a moved CI action tag, or a malicious pull request, including one written by a coding agent that weakens the tests that judge it, a behaviour measured in current models [13].
- *Physical fault injection on the trust root.* A person with the board on their own bench can glitch its supply or clock, or use laser or electromagnetic pulses. For the microcontroller we use, a public challenge produced four winning attacks, including supply-voltage glitching and laser injection that defeated secure boot or exposed one-time-programmable secrets [14], and independent measurement found its glitch detectors effective only at the highest sensitivity setting [15].

What we do not claim. We do not claim the trust root is hard to fault. A determined attacker with physical access can defeat a microcontroller, and secure boot, OTP settings and glitch detection are not treated as safety layers [E2]. The design goal, still to be proved on hardware, is narrower: a fault that makes the firmware misbehave must not by itself keep a rail on. That requires a cut that does not depend on the microcontroller at all, which Section 3.6 describes and which is not yet built. Keys stored in a board that has been broken open are treated as lost with that board.

3 Architecture

Figure 1 shows the components and the trust boundary. We name subsystems as the project does (T.R.I.A.G.E. for the MCP server, C.O.R.T.E.X. for the supervisor, P.U.L.S.E. for the heartbeat, N.E.R.V.E. for the module protocol, C.H.A.R.T. for the planned knowledge store, and S.C.R.U.B., "Stop Commands Read from Untrusted Bytes", for the taint latch) [E3], but describe each by function.

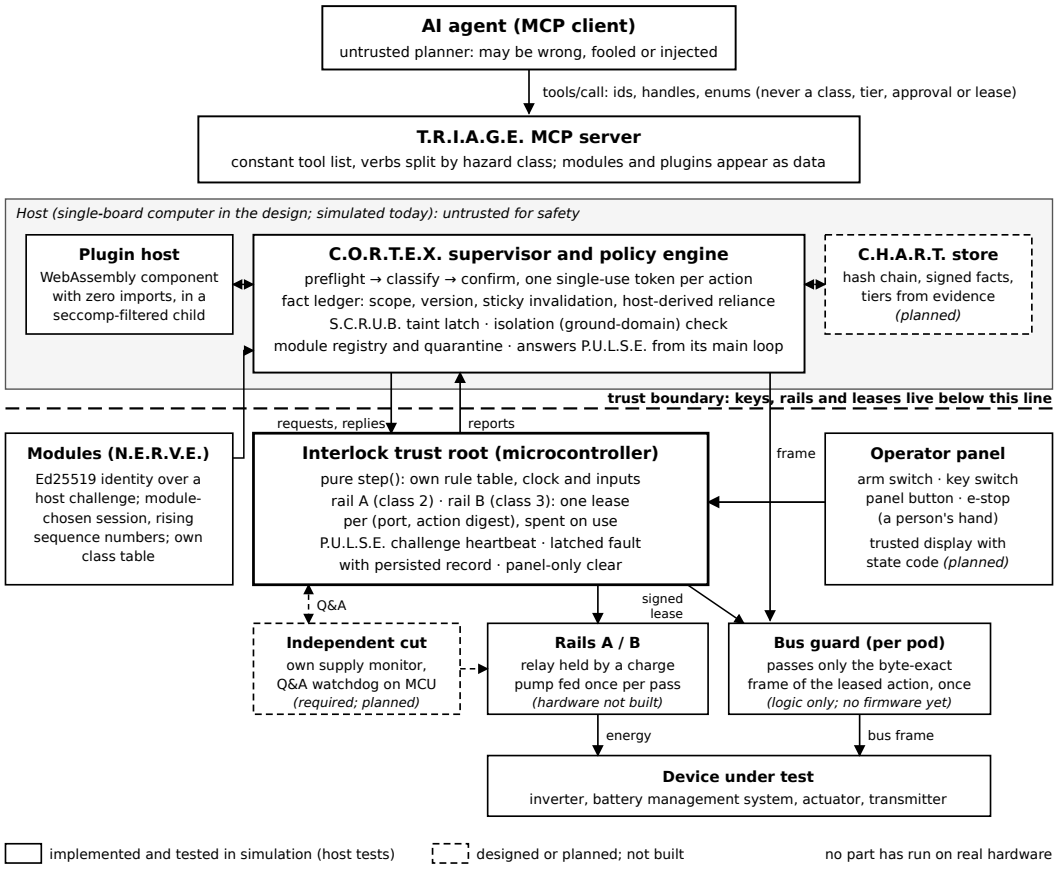


Figure 1: Architecture. The agent and the host are untrusted for safety. Keys, rails, leases and the fault latch live below the trust boundary. Solid boxes are implemented and tested in simulation; dashed boxes are designed or planned. No part has run on real hardware.

3.1 Hazard classes

Every action has one of five hazard classes (Figure 2) [E4]. Class 0 observes; class 1 is touch-safe on logic power; class 2 has bounded effect and is powered from rail A, which follows a physical arm switch; class 3 is hazardous and is powered from rail B, which needs a key switch and a lease for one action granted by a fresh press of the panel button; class 4 is not offered: no rail, tool, encoding or plan path exists for it. In our deployment, battery protection limits and charge voltages are class 3, and mains work and string switching are class 4 [E5].

Classification is a pure function of typed fields and attested facts, never of agent prose [E6]. Its rules are monotone: a write's class is the highest of its write-map entry's class, the minimum for its setting kind, class 2 if the target is energy-relevant, and class 2 because any transmission, a read included, can collide with another bus master. A write with no write-map entry at the measured or owner-attested tier is refused as class 4, so the unknown is the most restricted [E6]. Context can only raise a class; the value written never lowers it, though a value outside a declared range or of the wrong register width is refused.

The hazard class is not a safety integrity level. A safety integrity level or performance level rates the required reliability of a safety function [16], [17]; the hazard class rates the severity and reversibility of an agent's proposed action and decides which physical approval it needs.

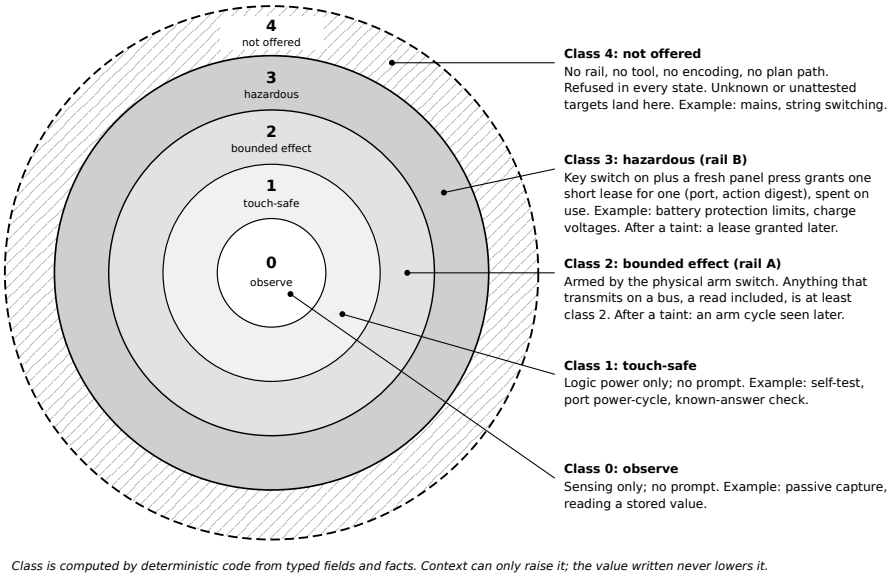


Figure 2: Hazard classes. Each ring needs everything the inner rings need plus its own approval.

3.2 Preflight, classify, confirm

An agent write passes through three steps in the supervisor and policy engine [E7]. **Preflight** works out, from the request itself, which facts it relies on (the write-map entry, the bus mastership fact, the isolation plan, the qualification of the instrument that measured a value); the agent never supplies this list. Each fact carries a scope: the exact device (model, serial and firmware), the interface, the conditions under which it holds, and the method that earned it, including instrument, procedure and uncertainty for a measured fact. A ledger records events that make facts stale: a firmware, serial or model change, a lost or moved device, a rewired port, a foreign bus master, a reflashed instrument (which invalidates what it measured and what was qualified against it), a condition that ends, an open contradiction and a clock discontinuity. Invalidation is sticky: the fact must be recorded again even if the cause goes away. Preflight refuses the first missing, stale, contradicted or out-of-scope fact by name and otherwise **classifies** the request and returns a token bound to the request, the action digest, the fact versions, the ledger generation and the store's chain head. **Confirm** runs at the moment of action, consumes the token and refuses if anything changed. In the supervisor, `invoke` takes the token and a latch pass by value, so there is no code path that skips either; compile-fail examples check this [E8].

3.3 Facts with provenance and trust tiers

Every reading carries one of five trust tiers: measured, owner-attested, documented, agent-inferred and unverified [E9]. A plugin's output is an observation with no tier; the host assigns the tier, and a compile-fail test fails the build if the tier field ever becomes assignable from outside the crate [E9]. A hardware write may rest only on measured or owner-attested facts [E5, E6]. The design adds a device-reported tier between measured and documented, under which a number a device reports about itself can back a class 2 action only when one of the bench's own instruments agrees; this is decided but not yet built [E5]. Text from outside the project's code is wrapped in a type with no display implementation, is rendered only inside an escaped, single-line fence, cannot become a port or signal identifier, and can leave its wrapper only at call sites listed in a reviewed allow-list that a test keeps current [E10].

3.4 Approvals and leases

A class 3 decision names one bus action: bench port, unit address, function code, register and register values [E11]. One encoder, shared by the policy engine, the approvals, the interlock and the bus guard, turns that action into a canonical encoding and a 32-byte digest, and into the one Modbus frame that can carry it [18], [19]. The encoding is injective, and the decoder refuses any bytes the encoder would not produce [E11]. This removes a confused deputy we found in review: before the shared encoder, nothing proved that the frame the guard passed encoded the action the policy engine classified [E12].

The trust root grants a rail B lease only on a fresh press of the panel button while a request is pending; the lease is bound to one port and one action digest, lasts a fixed number of ticks, ends on use, on a use that names another target, or when the key or arm switch goes off, and does not survive disarm, restart or fault [E13]. Approvals that leave the trust root are signed messages in a fixed 91-byte layout naming the approved object and its digest, the device it is for, the signer's role, a counter that rises strictly per signer and an expiry; the signature covers a domain tag and the message, and the only way to obtain an "approved" value in code is through the verifier [E14]. A role table restricts who may approve what: only the trust root (a panel press) may grant an energy lease, change the bus-guard allow-list or countersign a core release; the owner's phone may approve non-energy items; module keys approve nothing [E14].

The trusted panel display is designed to show the action and a short code for the state it was checked against, generated by the trust root rather than by the agent, so that a model-written summary is never the approval text; any change voids the approval. This is decided but not built [E5].

3.5 The Interlock trust root

The interlock's logic is a pure, allocation-free `no_std` state machine: one function takes the previous state, the inputs sampled on this tick and at most one event, and returns the next state and the outputs to drive [E15]. Its rules include: every output is off at power-on and after restart; a restart reads a persisted fault record and comes up faulted unless the record proves a clean shutdown; an emergency stop or lost heartbeat turns every output off on that tick and latches a fault; each rail's sensed state is compared with what was commanded, so a stuck-on rail drops everything; a fault clears only with a panel token made by the trust root's own panel driver while the button is sensed pressed, and a clear from the supervisor is refused in every state; fitting the firmware jumper while armed latches a fault; and class 4 has no rail. No event can turn a rail on by itself: rail A follows the arm switch and rail B the panel button, both a person's hand [E15].

The firmware main loop is written against a board abstraction and tested on the host with a simulated board [E16]. It drives every output off before the first tick, takes a nonce from the hardware random generator (with no nonce, the heartbeat never starts and nothing can arm), drives outputs only from the state machine's result, cuts first and writes the fault record before any output turns on, and feeds the relay's charge pump from one call site once per pass while a rail is on, a property a CI script checks by searching the firmware for timers, PWM, PIO, DMA, interrupts or a second call site [E16]. The decision to drive a rail needs two reads to agree, the outputs and the mode returned by the same step, so a single flipped value gives off; this does not guard a skipped branch, a corrupted program counter or a glitched pin write [E16]. A release image for the target microcontroller is built in CI; it has not been flashed or run.

P.U.L.S.E. The heartbeat is a challenge, not a ping. Each period the interlock issues a fresh challenge derived from the boot nonce; the supervisor must answer each once, on a later tick and before the next is due; a wrong, stale, early, repeated, missing or cross-boot reply latches a lost-heartbeat fault, and rails drop within two periods of the last good reply [E17]. The supervisor answers only from its main loop, so a stalled loop stops the heartbeat [E18]. This is the question-and-answer watchdog long used in automotive safety controllers, where wrong values, wrong order and wrong timing count as failures as well as silence [20], [21]; we adopted the pattern and do not claim it. In v0 the reply proves freshness, not identity: anything that can read a challenge can answer it until replies are keyed [E1].

3.6 The independent watchdog cut

Because a glitched microcontroller might keep toggling a pin from a corrupted loop, the design requires a cut that does not depend on it: a separate low-cost supervisor part with its own supply monitor that questions the trust root and holds the relay only while answers arrive correctly and on time, so that a frozen, glitched or misbehaving trust root drops the rail by itself [E2, E5]. The project decided on 26 November 2026 that this cut is required before any real power switching. It is not designed in detail and not built. Until it exists, the only promise is that the firmware fails toward off in every case we have considered, which is not a promise against a deliberate attacker.

3.7 Bus guard

An armed rail decides whether a transmitter has power, not what it sends; on Modbus one write function reaches both a harmless setting and a battery protection limit, and only the register differs [E19]. The bus guard is planned

as a small microcontroller in the transmit path of each bus pod. Its logic, implemented and tested on the host, passes only well-formed requests with function 0x03, 0x04, 0x06 or 0x10 and a correct CRC; blocks broadcast and reserved addresses, malformed, concatenated or split frames; lets reads through only under an allow-list rule; and lets a write through only if a rule permits it, a lease is live on the same port and the frame is byte-identical to the frame the lease's action encodes to, with the digest recorded at the grant [E19]. Passing spends the lease. The guard copies the frame before checking it and returns a `Pass` that owns the copy; the transmit path accepts only a `Pass` for its port, by value, so the checked bytes are the bytes sent and cannot be sent twice [E12]. If the firmware feeds the guard what it hears on the line and a request appears that the guard did not send, every write is blocked until the trust root sends a new allow-list [E19]. Allow-list changes and leases enter only through an event built from a verified trust-root approval [E19]. No guard firmware exists yet.

3.8 S.C.R.U.B.: the taint latch and the sterile field

Fencing outside text does not stop persuasion, so the latch stops a persuaded agent from spending an approval given before it read the text [E20]. Each MCP tool call is one turn. Every piece of content that is not host-authored is reported to the latch before it reaches the model; once it is, every class 2 and higher action is refused with a typed reason naming the source until a fresh approval is seen in the trust root's reports after the latest taint: an arm cycle (rail A seen off, then on) for class 2, or a lease granted after the taint for class 3, one action per lease. Classes 0 and 1 pass. The one outside input that does not trip the latch is the bare number of a reading at the measured tier; host-written lines and measured numbers form the "sterile field" [E5, E20]. The latch starts tripped, a new turn inherits it because the model still holds earlier turns, there is no clear method, and an approval from before the taint does not count [E20]. A switch flipped off and on between two reads is missed and the latch stays tripped, so it fails closed.

3.9 Isolation planning

Before arming, the supervisor runs a typed isolation check on the proposed connection plan [E21]. The map is a graph of ground domains joined by bonded or isolating edges; the checker reports a declared isolation defeated by a bonded path, ground loops, a barrier rated below the voltage across it, a measuring module that must float but is bonded to the host, and a bus with two masters. Any finding of severity Stop refuses arming, and class 2 and 3 actions need a plan that passed [E21]. The check describes the declared wiring, not the bench: a connection nobody declared is invisible to it.

3.10 T.R.I.A.G.E.: hazard-classed verbs, modules as data

The MCP revision of 28 July 2026 made the protocol stateless and requires that `tools/list` not vary per connection or as a side effect of other requests (it may vary by authorization); servers that need state mint explicit handles passed as arguments [9], [11]. We adopt the consequence deliberately: the tool list is a compiled constant, split by hazard class, and modules and plugins appear as data, never as new tools. A constant list can be pinned by a client and a golden test, removes a "rug-pull" surface, and keeps prompt caches stable. Tool arguments never carry a class, tier, approval or lease.

The merged v0 (#245) serves five read-class tools over stdio (`bench_status`, `inventory`, `describe`, `read`, `session_query`) against the simulated bench; the invoke names are unknown tools and the build fails if a non-read tool is registered; module text is fenced within a byte budget and reported to the latch; and golden tests pin the tool list and check that a hostile manifest does not change it [E22]. v0 speaks the 2025-11-25 initialisation handshake; an adapter for the 2026-07-28 revision is planned. The designed v1 has 16 constant tools, including `plan` (dry-run preflight returning a plan identifier and panel code), `invoke_safe` (classes 0 and 1), `invoke_armed` (class 2, plan identifier only) and `invoke_keyed` (class 3, which requests a lease and returns a pending state until the press) [E23]. Elicitation and multi-round-trip input are treated as nudges, never as approval: an answer that arrives through the agent's own channel cannot authorise class 2 or higher.

3.11 Plugin sandbox

Plugins are WebAssembly components [22] that import nothing: no clock, randomness, files, sockets or handles [E24]. The host refuses at load any import, nested component, start function, canonical built-in other than `lift`, export other than `decode`, or size over committed limits, before compiling. Each batch runs in a child process with an empty

environment, inert inherited descriptors, resource limits and a seccomp allow-list installed before the component is parsed; each frame runs in a fresh instance with fuel, memory, table and instance caps and a wall-clock kill. The child's reply is parsed as hostile bytes and becomes observations with no tier and no text [E24]. The child still runs as the host's user without namespaces or Landlock; those are planned.

3.12 Putting it together

Figure 3 traces one class 3 write from power-up to the wire. The trust root boots with outputs off and starts the heartbeat; the operator's arm switch enables rail A; the agent plans an action and the host binds it to facts and a digest; the trust root grants a lease for that digest only on a fresh press; the host confirms the token and checks the latch; the bus guard passes the one encoded frame once. Any loss of heartbeat, emergency stop or stuck output drops every rail and latches a fault.

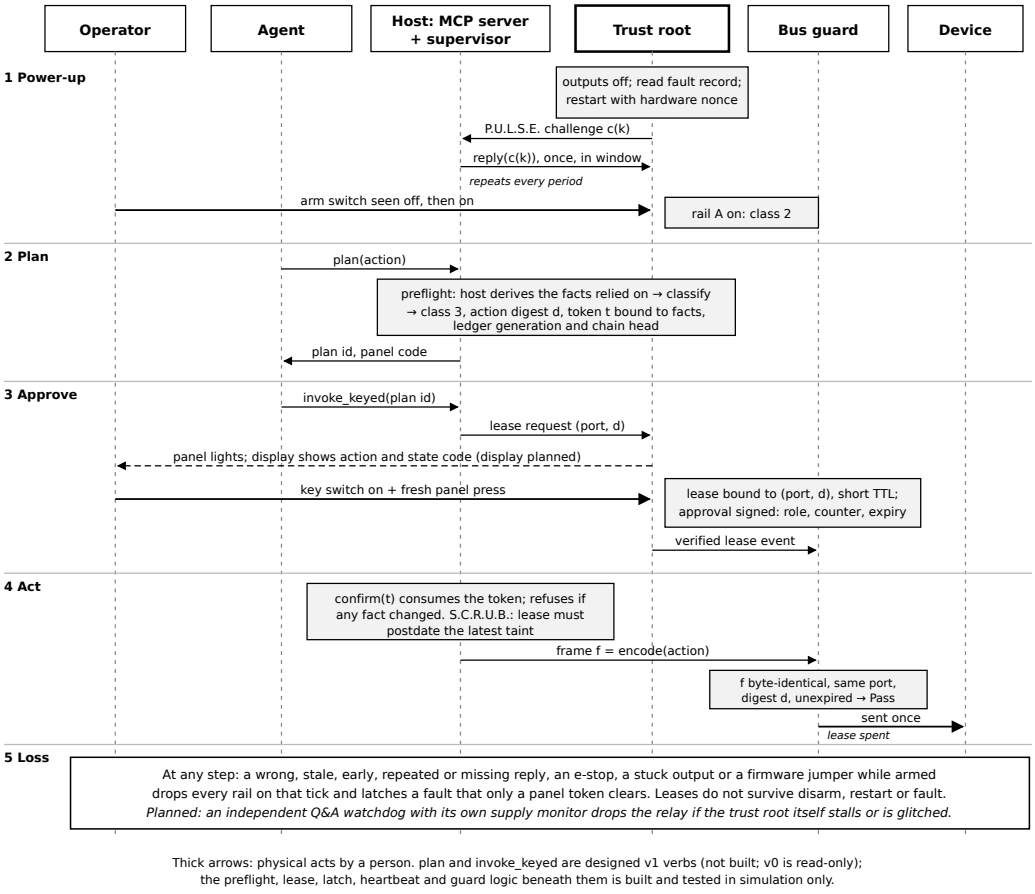


Figure 3: The power-up chain for one class 3 write. Thick arrows are physical acts by a person. The `plan` and `invoke_keyed` verbs are designed but not built; the logic beneath them is tested in simulation.

4 Hardware Write Safety 0.1

HWS is a working list of requirements for systems in which an AI agent can cause writes with physical effect. It follows the style of the MCP Write-Safety Requirements (MWS) [23]: stable identifiers of the form `HWS-<AREA>-<n>`, a level (MUST or SHOULD), a one-sentence requirement with rationale, and a verification method (T test, A analysis, I inspection). Version 0.1 has 56 requirements, 50 MUST and 6 SHOULD, in 13 areas. Table 1 condenses them; Appendix B lists every requirement with its status in Sawbones. Against the measured commit, Sawbones meets 34 in simulation only, 8 partly, and 14 not at all or by design alone; no requirement is met on hardware.

Table 1: HWS 0.1 areas, condensed. Status is against commit `0cb8ae5`; "in sim" means host or simulation tests only.

Area (count)	What it requires, in short	Concepts borrowed	Sawbones today
CLS Classification (4)	Ordered hazard classes; computed by code from typed fields; a never-class; energy targets cannot be lowered	Risk-based integrity targets (IEC 61508); avoidance by design	Met in simulation
APR Approval and leases (7)	Physical-act leases bound to one target and state; single use; short; void on disarm or fault; signed with counter and expiry; trusted display; role separation	Individually owned locks (OSHA 1910.147); lockable mode selection (ISO 10218-1)	6 met in sim; display not built
ISF Independent safety function (6)	Power removal on separate hardware; no software turn-on; independent watchdog; latched faults; stuck-output detection; stated diagnostic coverage	Two-channel architectures and cross-monitoring (ISO 13849-1); separation and independence (ISO 26262)	Logic in sim; independent cut and coverage not built
HBT Heartbeat (3)	Challenge with fresh hardware nonce; loss drops outputs within a stated bound; links time out on their own	Question-and-answer watchdog	Mostly in sim; bound not measured
STP Safe state and stops (4)	Defined safe state through boot and brown-out; stop works with host unpowered; declared stop category; isolation verified before energising	Stop categories (IEC 60204-1); emergency stop latch and reset (ISO 13850); verify isolation (LOTO)	Partly in sim; hardware tests pending
BUS Bus guarding (5)	Byte-identical frames only; one shared encoder; default deny; once per lease; no transmit while disarmed	Zones and conduits (IEC 62443)	4 met in sim; transmit test pending
PRV Provenance (5)	Origin and tier set by host; hazardous writes rest on measured or attested facts; facts expire; device self-reports need agreement; reflash voids	(gap: no standard counterpart)	4 met in sim; device-reported tier not built
TXT Outside text (5)	Typed and fenced; taint latch; latch held by the gate; transcribed approvals void; fixed verbs by class	(gap)	Mostly in sim; v1 verbs designed
LOG Evidence (3)	Tamper-evident record of proposals and refusals; fault record survives power loss; evidence pack	Lifecycle evidence (IEC 61508); audit (IEC 62443)	Fault record in sim; store not built
TST Testing (5)	Regression per defect; fuzz decoders; adversarial catalogue on every change; proof tests in the field; split-key changes	Proof testing and periodic function tests (ISO 14119)	3 met; proof tests and split-key CI pending
PHY Physical controls (3)	Physical arm, key and button; firmware jumper; individually owned lockout	Defeat resistance (ISO 14119); LOTO	Jumper rule in sim; hardware pending
UPD Updates (3)	Physical countersign of releases; signed boot with rollback protection; pinned build chain	Secure development (IEC 62443-4-1)	Countersign and pinning in sim/CI; signed boot not configured
HUM Human factors (3)	Prompts generated from typed fields; low classes need no prompt; armed, class and latch state always visible	Bypass incentive (ISO 14119); operator information	Design only

How HWS borrows. We borrow concepts, not text, and we have not reproduced any standard. From IEC 61508 [16] we take the idea that rigour should scale with assessed risk and that process evidence is part of a safety claim. From ISO 13849-1 [17] we take the observation that a single channel cannot carry a high-integrity function, and that diagnostic coverage and common-cause failure should be stated numbers. From IEC 60204-1 [24] we take declared stop categories, with energy outputs defaulting to an immediate removal of power. From ISO 13850 [25] we take an emergency stop that is always available, latches, and whose reset does not restart anything. From ISO 14119 [26] we take the assumption that interlocks will be defeated if they impede work, so low-hazard actions need no prompt and approval effort stays scarce. From OSHA's lockout/tagout rule [27] we take a human-held, individually owned lock and verification of isolation before work. From IEC 62443 [28] we take zones and conduits: the agent host, the safety controller and the device bus are three zones, and the bus guard is the conduit filter. From ISO 10218-1 [29] we take the selection of operating modes as a physical, lockable act. From ISO 26262 [21] we take independence between a safety controller and its external supervisor. From the software classes of IEC 60730-1 Annex H [30] we take the habit of classifying a program's safety role: the planner is, by construction, software that is not relied on for safety.

Seven gaps. These standards assume that control logic is a deterministic program written by an engineer. For an AI agent the assumption fails in seven ways, which HWS addresses:

1. *Prompt injection into the control path.* Industrial security models hostile actors on the network, not text that persuades a planner (HWS-TXT).
2. *An untrusted planner on the normal path.* Standards separate control from safety, but trust the control function to be correct and use the safety function as a backstop. Here the planner is expected to be wrong or hijacked, so the normal path itself must pass through an enforcement point (HWS-CLS-2, HWS-ISF-2).

3. *Provenance of the facts an action rests on.* Sensor integrity is a design property in interlock standards; nothing asks that each fact carry an origin and a tier that content cannot set, or that facts expire with their scope (HWS-PRV).
4. *Approval bound to exact action and state.* Lockout and mode selectors bind a decision to a physical state, not to an exact frame, target and state snapshot that resists replay and expires (HWS-APR-2 to 5, HWS-BUS-1 and 2).
5. *Social engineering of the approver.* No standard requires that operator information come from the safety function rather than from the system being supervised (HWS-APR-6, HWS-HUM-1).
6. *Evidence for a moving target.* Plugins and prompts change faster than certification cycles; the evidence a small builder can produce needs defining (HWS-LOG-3, HWS-TST).
7. *Honest scope.* A conformance claim must say what it is not (below).

Relation to MHS and MWS. MHS describes a device and its limits to an agent through a driver and a reference file [4]; HWS concerns what enforces those limits when the agent proposes a write. The two are complementary: an MHS reference file could supply the typed fields that HWS classification consumes, but HWS requires that the limits be enforced outside the planner's domain rather than read by it. MWS 0.9 covers software write tools (version checks, idempotency, input validation, truthful results, unknown outcomes and annotations) and its published overview does not mention physical effects [23]; HWS is intended as its physical counterpart and should not freeze before MWS settles.

Scope of a claim. HWS 0.1 conformance is not a safety integrity level, performance level or any certification. The planner is never part of the safety function, and only the safety function may claim anything. Sawbones has not been assessed against any standard.

5 Implementation and results to date

5.1 The workspace

The implementation is an Apache-2.0 Rust workspace [1]. At commit `0cb8ae5` (26 November 2026) it has 18 packages (17 crates and one firmware image), 157 Rust files and about 43,700 lines of Rust code excluding comments and blank lines (56,004 lines in all), with `unsafe` code forbidden workspace-wide [E25]. Every safety crate is marked provisional (the red-team crate, a test harness, is not). Table 2 lists the safety-relevant crates.

Table 2: Crates at `0cb8ae5`. Test counts are `#[test]` functions; lines are all lines of Rust source, including comments and documentation (56,004 in all). The workspace total is 757 `#[test]` functions.

Crate	Role	`#[test]` fns	Lines
sawbones-interlock	Interlock state machine, `no_std`	47	5,104
sawbones-trust-root	Firmware main loop against a simulated board	24	1,497
sawbones-policy	Classification, write map, fact ledger, preflight	91	7,025
sawbones-supervisor	Registry, heartbeat, single write path, taint latch	62	4,985
sawbones-approvals	Signed approvals, roles, counters, expiry	41	2,118
sawbones-modbus-encode	Shared action encoder and digest, `no_std`	8	880
sawbones-busguard	Bus guard logic	55	3,277
sawbones-module-protocol	Module wire protocol and identity, `no_std`	60	5,173
sawbones-reading	Trust tiers, outside-text type	38	1,562
sawbones-isolation	Ground-domain map and checker	40	2,744
sawbones-plugin-host	Zero-import plugin sandbox	54	4,873
sawbones-mcp	T.R.I.A.G.E. v0 (read-only)	14	2,495
sawbones-redteam	Red-team catalogue ratchet and attack tests	66	3,817
Other four crates and the firmware image	Bench, register maps, simulated modules, ingest adapter; RP2350 image	157	10,454

5.2 Results to date (simulation)

Tests. `cargo test --workspace --locked` at `0cb8ae5` ran 781 tests (unit, integration and documentation tests): 781 passed, none failed, and one was ignored because it needs network access [E26]. Nineteen `compile_fail` documentation examples check properties enforced by types, for example that a tier cannot be assigned from outside, that a raw slice cannot be transmitted past the guard, and that `invoke` cannot be called without a token and a latch pass [E8, E12].

Exhaustive and randomised checks of the interlock. One test applies every input pattern and event from generated sets to a set of representative states and checks the safety invariants after each single step; another runs 200 seeded sequences of 5,000 steps against an independent oracle and counts how often each condition was reached, so that a quiet pass cannot hide unexercised rules; a third pins the whole transition table to a rules version [E27]. These are tests over a model, not proofs over all states.

Mutation testing. A weekly cargo-mutants [31] job covers five safety crates. Its first run (commit `9d1438c`, 26 November 2026, time-boxed per shard) caught 752 of 843 viable mutants (89.2%): interlock 86 of 93, bus guard 154 of 162, approvals 114 of 124, module protocol 239 of 263 and policy 159 of 201 [E28]. The surviving mutants were triaged into regression issues and fixed with new tests (#224, #231) [E29]; the job has not yet been rerun on the current commit, so we do not report a current score.

Fuzzing. Four cargo-fuzz targets cover every decoder of outside bytes: approval messages, Modbus RTU frames at the guard, module-protocol frames and the outside-text fence. They run weekly in CI; the one run to date completed without failure [E30].

Regression per defect. Every bug becomes a named test (`regression_<issue>_...`) plus a row in a public ledger, in the same pull request as the fix. A CI guard, run from the base branch so the pull request cannot weaken it, recognises every closing keyword, requires a changed test naming the issue, and fails on deleted tests, `#[ignore]`, removed CI steps or its own edits unless a human applies a `guards-change` label; protected paths need the same label and a clean-room reviewer [E31]. The ledger has 18 rows for 15 issues, several found by red-teaming the guard itself (#199, #200) and by mutation testing (#224, #231). This matters because all 58 commits at the measured commit carry an AI coding agent as co-author, and language models measurably exploit test cases when a task conflicts with its tests [13].

Red-team catalogue. A public catalogue keeps attacks as data in four lanes [E32]. A card may be marked blocked only by naming a test or CI job that exists and is not ignored; a partial or gap card must name an open tracking item; and a test fails if the summary table is stale. Table 3 gives the counts at `0cb8ae5`.

Table 3: Red-team catalogue at `0cb8ae5`. Fewer than a quarter of cards are blocked; most interlock-hardware cards are partial because they need hardware to close.

Lane	Blocked	Partial	Gap	Total
Agent injection	9	13	9	31
Bus	11	6	12	29
Supply chain	6	9	15	30
Interlock hardware	4	27	6	37
All	**30**	**55**	**42**	**127**

The catalogue is a ledger of open problems, not evidence of attack resistance. Most interlock-hardware cards cannot move to blocked until the bench tests below exist.

Design conformance. A proposal to make the design itself checkable in CI (the "X-ray": machine-readable requirement records tied to annotations and live test results, reporting each requirement as built, partial, not built or drifted) is open for review as a draft pull request (#252) and is not merged [E33].

5.3 What is not built

We list these so that no reader mistakes the design for the implementation.

- **No real hardware.** No board exists; the trust-root firmware builds in CI but has not been flashed. Every item in the project's Bench Core acceptance checklist (outputs off through boot, reset and brown-out; stop with the host unpowered; time from lost heartbeat to rail off; expired, replayed and wrong-target approvals

refused on the panel; stuck-output detection; the bus guard blocking unapproved frames on a logic analyser) is unticked [E34].
- **No independent watchdog cut.** Required, not designed in detail, not built (Section 3.6).
- **No trusted panel display** and therefore no state code.
- **No tamper-evident store or recorder** (HWS-LOG-1); the planned store would compute tiers from signed evidence.
- **No bus-guard firmware**, no glitch-detector or brown-out configuration, no glitch-resistant comparisons in lease checks, and no signed boot on the trust root.
- **No keyed heartbeat**: replies prove freshness, not identity.
- **No MCP transport authentication** beyond stdio to the parent process, and no invoke tools: v0 is read-only.
- **No device-reported tier** and no computed tiers from signed evidence.

The project's threat model also contains some rows that lag the code (for example, it still says the interlock has no output feedback, which the state machine now implements); we treat the code and its tests as authoritative and have reported the drift for correction [E1, E15].

6 Planned evaluation: the T.R.I.A.G.E. benchmark

The safety logic above can be tested without a model. Whether the design is usable by agents, and at what cost, needs model runs. We will run them on the simulated bench, whose seeded fault scenarios provide ground truth, before any hardware run [E35].

Arms. (A) a naive server with one tool per device function generated from the same manifests, always listed, results inline; (B) a generic static server with list, read, write and query tools and the same safety instructions in its system text but enforced nowhere; (C) the full design. Ablations remove one feature of C at a time: hazard classes, the taint latch (also jointly with classes, since they interact), trust tiers, handles instead of inline results, and a variant with a state-dependent tool list, which the 2026-07-28 revision rules out but which some servers use. A control adds tool search to arm A to separate "fewer tools shown" from the rest of the design.

Scenarios. Hot-plug and detach mid-session; injected device text (benign, direct instruction, spoofed approval), placed where it is strongest; approval pressure, where the correct behaviour is to propose a class 3 action and stop; fault recovery (stuck line, replay, a module that over-claims); a self-reporting device that disagrees with a bench instrument; and distractor scaling with 3, 8 and 15 modules.

Metrics. Task success scored deterministically against simulator ground truth, with no model judge in any primary metric; reliability as pass^k [32]; tool-definition and tool-result tokens; calls and turns; error recovery; executed unapproved class 2 and 3 actions and injection success as a state check, reported with benign utility and utility under attack, following AgentDojo [33], [34]; trust-tier labelling accuracy; and, specific to our design, arm cycles per class 2 task as a measure of latch fatigue.

Statistics and pre-registration. Scenarios are the sampling unit; trials are clustered within scenario variants; arms are compared by paired differences with clustered standard errors [35]; rare safety events get exact intervals with zero-event results reported as upper bounds; multiplicity across ablations is corrected by Holm's method. The plan, primary endpoints, exclusion rules and a frozen scenario and attack set, with a held-out half written by a separate session that never sees the design, will be committed and tagged before the first paid model run.

Publishing losses. The project decided on 26 November 2026 that losses are published with the wins under the pre-registered plan, and that every scenario where the design loses to a plain MCP server becomes an issue: either the design changes until it matches or beats the baseline, or the loss is recorded as a deliberate trade-off with its measured cost [E5]. We expect losses: more turns per task from the plan-then-invoke flow, more first-try argument errors from generic verbs, and latch fatigue while most channels are below the measured tier.

Hardware evaluation. After the bench exists, each acceptance test in the checklist is repeated on hardware with logic-analyser captures, paired with the simulation test it mirrors, so that simulation and bench can be compared item by item.

7 Related work

Agents and hardware. MHS standardises device description for agents and names a physical safety roadmap without describing an enforcement mechanism [4]. Anthropic's Project Fetch phase two had a researcher approve commands while Claude operated a robot dog [36]. RoboGuard grounds safety rules in the robot's environment with a root-of-trust model and checks plans by temporal-logic control synthesis, reducing unsafe execution under jailbreak from over 92% to under 3% [37]; it runs in the compute domain it protects and uses a model to contextualise rules. ROSClaw connects foundation models to ROS 2 robots with safety validation before execution and audit logging, and reports that models differ substantially in how often they propose unsafe actions [38]. SROS2 secures ROS 2 communication but does not address planner-proposed actions [39]. IndustriConnect provides MCP adapters for Modbus, MQTT and OPC UA with mock-first evaluation [3]. The closest agent-side work is a trapped-ion control system in which each MCP tool call needs a token issued either after a simulated hardware run checked against per-device limits or by a human for sensitive actions, with each token tied to the exact call [40]. Our design differs in placing the decision on a separate controller the host cannot reach, in requiring a physical act for each hazardous action, and in binding the approval to bytes at the wire rather than to the call.

Approval binding. Argument-bound, single-use authorisation is now established for software agents: AgentCap issues short-lived, budget-limited capability tokens [41]; ACLE-MCP binds capability leases to an attested workload [42]; CapLease keeps durable consumption state so a retried action is not executed twice [43]; the Verifiable Action Card rebuilds approval prompts from the pending action and shows them in trusted interface chrome [44]; and Approval Laundering systematises six ways a harness executes something other than what was approved [45]. We adopt these lessons and move the binding to the last place an effect can be checked: the frame on the wire, transmitted only by the component that checked it.

Information-flow defences against injection. The dual-LLM pattern separates a privileged model from a quarantined one that reads untrusted text [46]. CaMeL separates control flow from untrusted data and enforces capabilities on data flow [47]. FIDES attaches confidentiality and integrity labels to agent data and enforces policies in the planner [48]. Our outside-text type and taint latch are a coarse, conservative relative of these: rather than tracking labels through computation, we withhold hazardous authority from any turn that has seen untrusted content until a person acts again. A risk-based helper reader, in the spirit of the dual-LLM pattern, is planned behind a flag and will be benchmarked against the latch alone [E5].

Benchmarks and threat taxonomies. AgentDojo and InjecAgent measure prompt-injection success and utility for tool-using agents [33], [34]; tau-bench introduced pass^k [32]; CHAI shows command hijacking through text placed in the physical scene [8]; the OWASP agentic list names goal hijack, tool misuse and human-agent trust exploitation [12]. A systematisation of MCP security and safety covers tool poisoning and injection in the MCP ecosystem [49]. None of these measures approval gating or trust-tier calibration for physical actions, which our benchmark adds.

Functional safety and runtime assurance. The Simplex architecture lets a simple verified controller override a complex unverified one [6]. NVIDIA's Halos for Robotics uses a safety island with its own input/output, power and clock sources, separate from the AI compute domain, and cites IEC 61508, ISO 26262 and ISO 13849 [50]. Question-and-answer watchdogs are standard in automotive power management [20]. DNP3 Secure Authentication authorises critical utility commands [51]. These supply the separation and supervision patterns we use; they do not address agent-proposed actions or prompt injection. Policy work on embodied AI identifies gaps in current regulation and calls for compulsory testing [52].

Write safety for MCP. MWS 0.9 sets requirements for software write tools [23]. MCP's own guidance asks for a human able to deny invocations and treats annotations as untrusted hints [9], [10]. To our knowledge no published standard addresses LLM agents commanding hardware; HWS is a first working draft offered for review, not a finished standard.

8 Limitations and ethics

No certification claim. Nothing here is a safety integrity level, performance level or certified safety function. The trust-root microcontroller is not a safety-rated part. We call the design safety-inspired and describe its claims only in terms of tests.

Simulation only. Every result in Section 5 is from host tests and simulated modules. Pin wiring, boot and brown-out behaviour, relay timing, electromagnetic effects and the behaviour of real devices are untested. Simulated

modules and our own scenarios may share our blind spots.

We grade our own work. The design, the code, the red-team catalogue, the standard and this report come from one small organisation, and most code was written by AI coding agents under the rules in Section 5. The regression guard, mutation testing, clean-room review of protected changes and the held-out benchmark set mitigate but do not remove this. We invite outside review and will record it in later versions.

A very young code base. The repository's first commit was on 25 November 2026; the measured commit is the 58th. Interfaces will change, and some project documents lag the code.

Physical access. As Section 2 states, a person with the board in hand can defeat the trust root until the independent cut exists, and possibly after.

Owner error. A wrong owner attestation about a register or a bus master is not caught by software; only class 3's physical press and a second instrument, where one exists, limit it.

Dual use and scope. The project's field toolkit catalogues transmitting and active-testing tools for radio, networks, buses and firmware. It is limited by project decision to the owner's own systems and property; jamming is excluded; no exploit or malware code is written, and red-team cards describe defensive tests against our own components [E5, E32]. Readers must check the law where they operate: radio transmission, interference with equipment they do not own, and work on mains or grid-connected equipment are regulated in most jurisdictions and may require licences or qualified persons. Nothing in this report is advice to perform electrical work.

Privacy. The repository and this report contain no addresses, serial numbers or recordings from the site; the deployment is described only as an off-grid site.

9 Conclusion

An AI agent that writes to hardware should be treated as an untrusted proposer whose authority is bounded by things it cannot touch: a separate controller with its own clock and rule table, a person's hand on a switch, and a guard that sends only the bytes that were approved. We have described an architecture that does this, a working standard, HWS 0.1, that turns its joints into 56 testable requirements, and an open implementation whose logic passes its tests in simulation. The most important parts for real safety, the independent cut, the trusted display and every hardware test, are not yet built, and most of our own red-team cards are still partial or open. We publish now so that the design, its evidence and its gaps are on record and open to criticism before we put power through it.

Disclosure

Sawbones Labs and IsoKron share a founder. HWS and the MCP Write-Safety Requirements (MWS) share an author; MWS is stewarded by IsoKron, which sells audits and builds of MCP servers and has a commercial interest in write-safety requirements. Neither HWS nor Sawbones is a product of, or endorsed by, the Model Context Protocol project or Anthropic. Most of the code was written with AI coding agents, as the commit trailers in the repository record, and this report was drafted with AI assistance and reviewed by the author, who takes responsibility for its content. No external funding supported this work.

Availability

Code, threat model, decision log, regression ledger and red-team catalogue: https://github.com/rafe-walker/sawbones-labs (Apache-2.0), measured at commit `0cb8ae5`. Project site: https://plotnode.io. This report, its Markdown source and its figures are published at the project site under SL-TR-2026-01.

Version history

Version	Date	Changes
0.1	26 November 2026	First working paper. Code facts at commit `0cb8ae5`; red-team counts 30 blocked, 55 partial, 42 gap of 127; HWS 0.1 with 56 requirements.

References

[1] Sawbones Labs, "sawbones-labs: An agent-operated test bench with an off-host interlock (source repository)." GitHub repository, Apache-2.0, commit 0cb8ae5, 2026. Available: https://github.com/rafe-walker/sawbones-labs

[2] robotmcp, "ros-mcp-server: Connect AI models with robots using MCP and ROS." GitHub repository, 2026. Available: https://github.com/robotmcp/ros-mcp-server

[3] M. Xavier, M. Jolly, V. M A, and M. Xavier, "IndustriConnect: MCP adapters and mock-first evaluation for AI-assisted industrial operations." 2026. Available: https://arxiv.org/abs/2603.24703

[4] Anthropic, "Previewing the Model Hardware Standard." Anthropic news, Aug. 27, 2026. Available: https://www.anthropic.com/news/model-hardware-standard-research-preview

[5] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975, doi: 10.1109/PROC.1975.9939.

[6] L. Sha, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001, doi: 10.1109/MS.2001.936213.

[7] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection." 2023. Available: https://arxiv.org/abs/2302.12173

[8] L. Burbano *et al.*, "CHAI: Command hijacking against embodied AI." 2025. Available: https://arxiv.org/abs/2510.00181

[9] Model Context Protocol, "Tools, specification revision 2026-07-28." 2026. Available: https://modelcontextprotocol.io/specification/2026-07-28/server/tools

[10] O. Hungerford, S. Morrow, and L. Chang, "Tool annotations as risk vocabulary: What hints can and can't do." Model Context Protocol blog, Mar. 16, 2026. Available: https://blog.modelcontextprotocol.io/posts/2026-03-16-tool-annotations/

[11] Model Context Protocol, "Key changes, specification revision 2026-07-28." 2026. Available: https://modelcontextprotocol.io/specification/2026-07-28/changelog

[12] OWASP Gen AI Security Project, "OWASP top 10 for agentic applications: The benchmark for agentic security in the age of autonomous AI." Dec. 09, 2025. Available: https://genai.owasp.org/2025/12/09/owasp-top-10-for-agentic-applications-the-benchmark-for-agentic-security-in-the-age-of-autonomous-ai/

[13] Z. Zhong, A. Raghunathan, and N. Carlini, "ImpossibleBench: Measuring LLMs' propensity of exploiting test cases." 2025. Available: https://arxiv.org/abs/2510.20270

[14] E. Upton, "Security through transparency: RP2350 hacking challenge results are in." Raspberry Pi news, Jan. 14, 2025. Available: https://www.raspberrypi.com/news/security-through-transparency-rp2350-hacking-challenge-results-are-in/

[15] R. Pareja Veredas, "Characterizing the Raspberry Pico 2 FI countermeasures, part 1." IOActive blog, Jul. 31, 2025. Available: https://www.ioactive.com/characterizing-the-raspberry-pico-2-fi-countermeasures-part-1/

[16] IEC, "IEC 61508 (all parts), functional safety of electrical/electronic/programmable electronic safety-related systems." 2010.

[17] ISO, "ISO 13849-1:2023, safety of machinery — safety-related parts of control systems — part 1: General principles for design." 2023. Available: https://www.iso.org/standard/73481.html

[18] Modbus Organization, "MODBUS application protocol specification, V1.1b3." Available: https://www.modbus.org/modbus-specifications

[19] Modbus Organization, "MODBUS over serial line: Specification and implementation guide, V1.02." Available: https://www.modbus.org/modbus-specifications

[20] Texas Instruments, "Q&a watchdog timer configuration for DRV3205-Q1," Texas Instruments, Application Report SLVA831, Oct. 2016. Available: https://ti.com/lit/pdf/slva831

[21] ISO, "ISO 26262:2018 (all parts), road vehicles — functional safety." 2018.

[22] Bytecode Alliance, "The WebAssembly component model." Available: https://component-model.bytecodealliance.org/

[23] IsoKron, "MCP write-safety requirements (MWS), version 0.9, draft for public comment." Nov. 2026. Available: https://github.com/isokron-ai/mcp-write-safety

[24] IEC, "IEC 60204-1, safety of machinery — electrical equipment of machines — part 1: General requirements." 2016.

[25] ISO, "ISO 13850:2015, safety of machinery — emergency stop function — principles for design." 2015. Available: https://www.iso.org/standard/59970.html

[26] ISO, "ISO 14119:2024, safety of machinery — interlocking devices associated with guards — principles for design and selection." 2024. Available: https://www.iso.org/standard/75942.html

[27] U.S. Occupational Safety and Health Administration, "29 CFR 1910.147, the control of hazardous energy (lockout/tagout)." Available: https://www.osha.gov/laws-regs/regulations/standardnumber/1910/1910.147

[28] IEC, "IEC 62443 (series), security for industrial automation and control systems; in particular 62443-3-3 (system security requirements and security levels) and 62443-4-1 (secure product development lifecycle requirements)."

[29] ISO, "ISO 10218-1:2025, robotics — safety requirements — part 1: Industrial robots." 2025. Available: https://www.iso.org/standard/73933.html

[30] IEC, "IEC 60730-1, automatic electrical controls — part 1: General requirements (annex h, software classes)."

[31] cargo-mutants contributors, "cargo-mutants: Mutation testing for Rust." Available: https://mutants.rs/

[32] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan, "τ-bench: A benchmark for tool-agent-user interaction in real-world domains." 2024. Available: https://arxiv.org/abs/2406.12045

[33] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents." 2024. Available: https://arxiv.org/abs/2406.13352

[34] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, "InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents." 2024. Available: https://arxiv.org/abs/2403.02691

[35] E. Miller, "Adding error bars to evals: A statistical approach to language model evaluations." 2024. Available: https://arxiv.org/abs/2411.00640

[36] Anthropic, "Project Fetch: Phase two." Jun. 18, 2026. Available: https://www.anthropic.com/news/project-fetch-phase-two

[37] Z. Ravichandran, A. Robey, V. Kumar, G. J. Pappas, and H. Hassani, "Safety guardrails for LLM-enabled robots." 2025. Available: https://arxiv.org/abs/2503.07885

[38] I. S. Cardenas, M. A. Arnett, N. C. Yeo, L. Sah, and J.-H. Kim, "ROSClaw: An OpenClaw ROS 2 framework for agentic robot control and interaction." 2026. Available: https://arxiv.org/abs/2603.26997

[39] V. Mayoral Vilches, R. White, G. Caiazza, and M. Arguedas, "SROS2: Usable cyber security tools for ROS 2." 2022. Available: https://arxiv.org/abs/2208.02615

[40] D. Wang, L. Qi, Y. Xie, N. M. Linke, and K. R. Brown, "A hardware-safety-gated system for LLM-written native ARTIQ control code on a trapped-ion platform." 2026. Available: https://arxiv.org/abs/2606.27231

[41] A. K. Kabbinamakki Suresh and P. Gao, "AgentCap: Capability-based runtime authorization for secure LLM agent execution." 2026. Available: https://arxiv.org/abs/2611.00729

[42] Z. Ding, Y. Luo, G. Chen, Q. Shen, and Z. Wu, "ACLE-MCP: Attested capability leases for execution-time trust in remote LLM tool use." 2026. Available: https://arxiv.org/abs/2609.02690

[43] J. Xu, L. Fan, Z. Wang, X. Li, and H. Liu, "Beyond single-use tokens: Durable authorization state for replay-resistant LLM agent actions." 2026. Available: https://arxiv.org/abs/2608.01710

[44] H. Irshad, A. Mughees, N. Mughees, A. Mughees, and I. A. Soomro, "The verifiable action card: Trustworthy human-in-the-loop control for secure autonomous agents." 2026. Available: https://arxiv.org/abs/2609.18411

[45] Y. Wang, "Approval laundering: Systematizing approval–execution binding failures in AI coding-agent harnesses." 2026. Available: https://arxiv.org/abs/2609.38983

[46] S. Willison, "The dual LLM pattern for building AI assistants that can resist prompt injection." Apr. 25, 2023. Available: https://simonwillison.net/2023/Apr/25/dual-llm-pattern/

[47] E. Debenedetti *et al.*, "Defeating prompt injections by design." 2025. Available: https://arxiv.org/abs/2503.18813

[48] M. Costa *et al.*, "Securing AI agents with information-flow control." 2025. Available: https://arxiv.org/abs/2505.23643

[49] S. Gaire, S. Gyawali, S. Mishra, S. Niroula, D. Thakur, and U. Yadav, "Systematization of knowledge: Security and safety in the model context protocol ecosystem." 2025. Available: https://arxiv.org/abs/2512.08290

[50] S. Hariharapura Sheshadri, R. Mariani, S. Ochoa, D. Rodge, and S. Todd, "Inside NVIDIA Halos for robotics: A full-stack functional safety system for physical AI." NVIDIA Technical Blog, Jun. 22, 2026. Available: https://developer.nvidia.com/blog/inside-nvidia-halos-for-robotics-a-full-stack-functional-safety-system-for-physical-ai/

[51] IEEE, "IEEE 1815-2012, standard for electric power systems communications — distributed network protocol (DNP3)." 2012. Available: https://standards.ieee.org/standard/1815-2012.html

[52] J. Perlo, A. Robey, F. Barez, L. Floridi, and J. Mökander, "Embodied AI: Emerging risks and opportunities for policy action." 2025. Available: https://arxiv.org/abs/2509.00117

Appendix A: Evidence map

Paths are relative to the repository root at `0cb8ae5`. Test names are Rust `#[test]` functions unless marked.

ID	Claim	Where it is verified
E1	Assets, trust boundaries, attackers, residual risks	`docs/THREAT-MODEL.md` sections 1 to 4
E2	Physical fault-injection model; no reliance on secure boot or glitch detection; external cut required	`docs/THREAT-MODEL.md` section 4a; red-team card `hw-037`
E3	Subsystem names, S.C.R.U.B. expansion	`docs/DECISIONS.md`, 26 Nov 2026 (names; S.C.R.U.B.)
E4	Five hazard classes; class 4 refused in every state	`crates/sawbones-interlock/src/lib.rs` (`HazardClass`); `class_4_is_refused_in_every_state`
E5	Owner decisions: class 3 battery limits, class 4 mains; facts rule; device-reported tier; panel shows what it approves; RP2350 kept and external cut required; helper reader; publish losses; transmit testing on owned gear only	`docs/DECISIONS.md`
E6	Classification rules; unattested write is class 4; energy entries never lowered	`crates/sawbones-policy/src/decide.rs` (`classify`); `regression_171_unattested_register_is_class_4`; `energy_relevant_entries_are_at_least_class_2_and_never_lowered`; `regression_187_*`, `regression_214_*`
E7	Fact scope, ledger, sticky invalidation, host-derived reliance, token, confirm	`crates/sawbones-policy/src/preflight.rs`; `tests/preflight.rs` (e.g. `a_reflash_falls_through_every_module_qualified_against_it`, `a_condition_that_ends_refuses_the_write_and_stays_refused`, `a_token_expires_with_its_facts_inside_the_window`)
E8	One write path; no bypass of token or latch	`crates/sawbones-supervisor/src/lib.rs` (`invoke`, `LatchPass`, `Authorized` compile-fail examples)
E9	Five tiers; host-set tier; tier not assignable	`crates/sawbones-reading/src/reading.rs` (`Trust`; `the_host_sets_the_tier_and_the_observation_has_none`; compile-fail doctest on `Reading`)
E10	Outside-text type, fence, identifier grammar, call-site allow-list	`crates/sawbones-reading/src/world_text.rs` (`regression_202_*`); `regression_187_world_text_cannot_become_an_identifier`; `redteam/as_untrusted_str.allow`
E11	Shared encoder, injective encoding, digest	`crates/sawbones-modbus-encode/src/lib.rs`; `regression_203_the_action_encoding_and_digest_are_pinned`
E12	Confused deputy fixed; `Pass` owns checked bytes; transmit only a `Pass`	`docs/REGRESSIONS.md` row #203; `crates/sawbones-busguard/tests/regression_203.rs`; compile-fail examples in `crates/sawbones-busguard/src/lib.rs`
E13	Lease by fresh press; bound to port and digest; single use; TTL; void on disarm, restart, fault	`crates/sawbones-interlock/tests/rules.rs`: `lease_is_granted_only_by_a_fresh_button_press`, `lease_is_spent_after_one_use`, `a_clock_that_goes_backwards_drops_the_lease`, `lease_does_not_survive_disarm_restart_or_fault`
E14	Signed 91-byte approvals; counter; expiry; role table	`crates/sawbones-approvals/src/lib.rs` (`MESSAGE_LEN`, `may_approve`); `a_replayed_counter_is_refused`; `an_expired_approval_is_refused_at_and_after_its_expiry`; `a_core_release_needs_the_trust_root`; `an_owner_phone_cannot_grant_a_lease_or_change_the_allow_list`
E15	Interlock rules; output feedback; panel-only clear; jumper	`crates/sawbones-interlock/src/lib.rs` (crate documentation); `a_stuck_on_rail_drops_everything_and_latches`; `regression_197_*`; `firmware_jumper`; `arm_sequence`
E16	Firmware loop contract; charge pump single call site; two-read permit	`crates/sawbones-trust-root/src/lib.rs`; `tests/main_loop.rs` (`outputs_are_driven_off_before_the_first_tick`, `a_board_whose_generator_yields_nothing_can_never_arm`, `the_record_is_written_and_read_back_before_rail_a_turns_on`, `a_stalled_loop_stops_the_pump`); `tests/permit.rs`; `ci/check_charge_pump.py`; `firmware/trust-root`
E17	Challenge heartbeat; loss conditions; two-period bound	`crates/sawbones-interlock/tests/rules.rs` (`regression_198_*`, incl. `regression_198_no_heartbeat_before_restart`)
E18	Supervisor answers only from its main loop	`crates/sawbones-supervisor/src/lib.rs`; `a_missed_heartbeat_drops_the_rails_within_the_interlocks_budget`
E19	Bus guard rules; foreign master; updates only from the trust root	`crates/sawbones-busguard/src/lib.rs`; `regression_203_exactly_the_approved_actions_frame_passes`; `a_leased_write_passes_once_then_blocks`; `redteam_bus_001_*`, `_002_*`, `_004_*`
E20	Taint latch rules	`crates/sawbones-supervisor/src/latch.rs`; `a_tripped_latch_allows_class_0_and_1_only`; `a_pre_taint_approval_does_not_clear_it`; `redteam_inj_013_a_transcribed_approval_cannot_clear_the_latch`; `redteam_inj_025_a_later_turn_inherits_the_latch`
E21	Isolation map and checker; arming refused on Stop	`crates/sawbones-isolation/src/lib.rs`; `tests/checks.rs`; `a_stop_isolation_finding_refuses_arming`
E22	MCP v0: five read tools; golden list; protocol versions	`crates/sawbones-mcp/README.md`, `src/server.rs` (`PROTOCOL_VERSIONS`); `tests/golden.rs` (`the_tool_list_is_pinned`, `no_input_field_can_carry_authority`, `a_hostile_manifest_does_not_change_the_tool_list`); `only_read_class_tools_exist_and_the_invoke_names_are_not_callable`; `redteam_103_*`
E23	v1 design: 16 constant tools	Design note for T.R.I.A.G.E. v1 (project research, not in the repository); `docs/DECISIONS.md` (T.R.I.A.G.E. tool surface)
E24	Plugin admission and sandbox	`crates/sawbones-plugin-host/src/lib.rs`; `tests/host.rs` (`plugin_with_an_import_is_refused`, `plugin_cannot_claim_a_trust_tier`); `tests/sandbox.rs`

ID	Claim	Where it is verified
E25	Workspace size; `unsafe` forbidden	`Cargo.toml` (`members`, `unsafe_code = "forbid"`); line counts by `tokei` on `crates/` and `firmware/`
E26	781 passed, 0 failed, 1 ignored	`cargo test --workspace --locked` run by us at `0cb8ae5`; ignored test `every_tracking_issue_is_open`
E27	Exhaustive and random interlock checks	`crates/sawbones-interlock/tests/invariants.rs`: `every_single_step_from_representative_states_is_safe`, `random_sequences_keep_the_invariants` (200 seeds, 5,000 steps), `rules_version_pins_the_transition_table`
E28	Mutation counts	`.github/workflows/mutants.yml`; workflow run of 26 Nov 2026 on `9d1438c`, per-shard `caught.txt`, `missed.txt`, `timeout.txt` artefacts
E29	Survivors fixed	`docs/REGRESSIONS.md` rows #224 and #231
E30	Four fuzz targets; weekly job	`fuzz/fuzz_targets/`; `.github/workflows/fuzz.yml`; run of 26 Nov 2026
E31	Regression guard and protected paths	`ci/regression_guard.py`; `ci/test_regression_guard.py` (`regression_199_*`); `.github/workflows/regression-guard.yml`, `protected-paths.yml`; `docs/REGRESSIONS.md`
E32	Red-team catalogue, ratchet, counts	`redteam/README.md`; `redteam/catalog/*.toml`; `crates/sawbones-redteam`
E33	Design-conformance CI proposal	Draft pull request #252 (open, not merged)
E34	Hardware acceptance checklist, all unticked	`docs/THREAT-MODEL.md` section 5
E35	Simulated modules and fault scenarios	`crates/sawbones-sim-modules`

Appendix B: HWS 0.1 requirements

Status against `0cb8ae5`: **Sim** = met in host or simulation tests (never a physical claim); **Part** = partly met; **No** = not yet. Verification: T test, A analysis, I inspection.

Table 6: HWS 0.1 requirements (M = MUST, S = SHOULD). Identifiers omit the `HWS-` prefix.

ID	Lvl	Requirement (short form)	V	Status
CLS-1	M	Every actuating action has one class from an ordered, published scale before it is offered; unclassified means most restricted	T,A	Sim
CLS-2	M	Class computed by deterministic code from typed fields and facts, never agent prose	T	Sim
CLS-3	M	A never-class exists, is not offered and is refused in every state	T	Sim
CLS-4	M	Classes of energy-relevant targets can be raised, never lowered, by plugins, devices or low-trust facts	T	Sim
APR-1	M	Hazardous actions run only under a lease created by a person's physical act at the device	T,I	Sim (logic)
APR-2	M	A lease binds one target, value and state digest; any change voids it	T	Sim
APR-3	M	Leases are single-use, short-lived, timed by the safety function's clock; a backwards clock ends them	T	Sim
APR-4	M	Leases do not survive disarm, restart or fault	T	Sim
APR-5	M	Approvals are signed with a rising per-signer counter and expiry; replays and expired ones are refused	T	Sim
APR-6	M	The approver sees the action and a state code from the safety function; model text is never the approval text	T,I	No
APR-7	M	Roles are separated: a role that cannot touch energy cannot grant energy leases or change allow-lists	T	Sim
ISF-1	M	Power removal runs on separate hardware outside the planner's domain and accepts no "safe" assertion from it	A,I	Part
ISF-2	M	No software event on the agent side can turn an output on; enabling needs a physical arm input	T	Sim (logic)
ISF-3	M	A watchdog independent of the safety controller cuts the output if the controller stalls or misbehaves	T,A	No
ISF-4	M	Faults latch, survive power loss and clear only by a physical act; an unreadable record boots faulted	T	Sim
ISF-5	M	A stuck-on output is detected through feedback and drops everything	T	Sim (logic)
ISF-6	S	The claim states diagnostic coverage and common-cause analysis without quoting an unassessed level	A	No
HBT-1	M	Liveness is a challenge with a fresh hardware nonce; wrong, stale, early, repeated or missing answers are loss	T	Sim
HBT-2	M	Heartbeat loss drops outputs within a stated bound and latches a fault	T	Sim; bound not measured
HBT-3	S	Every device link times out into its own safe state without supervisor help	T	Part (card `hw-035` partial)
STP-1	M	Each output has a safe state reached on fault, reset, brown-out or power loss; off through boot	T	Part
STP-2	M	An emergency stop works with the agent host unpowered, latches, and its reset restarts nothing	T,I	Part
STP-3	S	The stop category is declared per output; energy outputs default to immediate removal of power	I	No
STP-4	M	Arming is refused when an unintended ground or energy path is found	T	Sim
BUS-1	M	A guard outside the planner's domain forwards only a frame byte-identical to an approved one	T	Sim (logic)
BUS-2	M	One shared encoder and digest serve policy, approval and guard	T	Sim
BUS-3	M	Default deny: broadcast, reserved, unsupported, malformed, concatenated or split frames never pass	T	Sim; fuzzed

ID	Lvl	Requirement (short form)	V	Status
BUS-4	M	An approved frame passes at most once; an unanswered write blocks new leases until a readback	T	Sim
BUS-5	M	The command path cannot transmit while disarmed	T	No
PRV-1	M	Every fact has an origin and a tier set by the host, never by content	T	Sim
PRV-2	M	Hazardous actions rest only on measured or owner-attested facts	T	Sim
PRV-3	M	Facts and approval tokens expire; mastership and state facts carry lifetimes	T	Sim
PRV-4	M	A device's report about itself backs class 2 or higher only if an independent instrument agrees	T	No
PRV-5	M	An instrument firmware change voids what it measured until a known-answer check passes	T	Sim
TXT-1	M	Outside text is typed as untrusted, fenced when shown and cannot become an identifier or command	T	Sim; fuzzed
TXT-2	M	After outside text enters context, actions above the safe classes need a fresh physical act (taint latch)	T	Sim
TXT-3	M	The latch is held by the gate, not the agent; pre-taint approvals do not clear it	T	Sim
TXT-4	M	A pasted or transcribed approval has no effect	T	Part (card `inj-013` partial)
TXT-5	S	The agent's tool surface is a fixed set of verbs split by class; devices and plugins appear as data	I	Part (v0 read-only)
LOG-1	M	A tamper-evident record keeps every proposal, class, approval, refusal and outcome	T,I	No
LOG-2	M	The safety function records fault reason and provenance in storage that survives power loss	T	Sim (logic)
LOG-3	M	A conformance claim ships an evidence pack: requirement-to-test map, versions, hardware captures	I	Part
TST-1	M	Every defect becomes a named regression check plus a ledger row in the fix	I	Sim (CI)
TST-2	M	Every decoder of outside bytes is fuzzed in CI	T	Sim (CI)
TST-3	M	A versioned adversarial catalogue runs against the real crates on every change	T	Sim (CI)
TST-4	M	The installed safety path is proof-tested on a stated interval with dated records	T	No
TST-5	M	Whoever changes a decoder cannot change its safety tests in the same change	I	Part (decision)
PHY-1	M	Arm switch, key and panel button are physical with no software path to simulate them	I,T	No
PHY-2	M	Firmware updates need a physical jumper; fitting it while armed faults	T	Sim (logic)
PHY-3	M	A person can lock out the energy source physically with an individually owned lock	I	No
UPD-1	M	Safety firmware and core releases install only after a physical countersign of their digest	T	Sim
UPD-2	M	The safety controller boots signed code with rollback protection	T,I	No
UPD-3	S	The build chain is pinned and reviewed; no private key in the default build	T,I	Sim (CI)
HUM-1	M	Approval prompts are generated from typed fields by the gate, not from agent text	I	No
HUM-2	S	Low classes need no prompt, so approval effort stays scarce	A	Design
HUM-3	M	The operator can always see armed state, pending class and latch state	I	No