Proof of Write: An Evidentiary Standard for Agent Action Capability

Shubham Raghav, Cresva

CRV-2026-001 Version 1.4 Published 2026-08-05 Revised 2026-10-12

Abstract

Agentic products describe their ability to act on external systems in the language of breadth: platforms connected, verbs shipped, integrations closed. Breadth answers a question nobody asked. Running one hostile, evidence-gated audit against one system on one day produced four different coverage figures from the same 130-capability matrix, differing only in what the auditor accepted as evidence: a breadth narrative for which no value was ever computed, 56.6% for code that exists and looks contract-correct, 5.4% once a requested permission scope had to be enforced before the call rather than merely requested at consent, and 0.0% for writes proven to have reached a vendor. We propose an evidentiary standard that resolves the ambiguity: seven tiers, each with a criterion a third party can apply without access to the code, the artifact that satisfies it, and who is able to verify it. We then apply the standard to the system it was derived from. The top-tier count was zero at publication and was reported as one in September. Re-measured on 12 October 2026 with the per-row grader now in the codebase, graded under the ladder as corrected in revision 1.3 and counting only brands recorded as real, seven production rows on the operator's own account grade at the executed tier, covering five distinct capabilities, none grades at the confirmed tier, and no customer account has a live write. The reasons the count moved, and the corrections made to the standard and its grader since publication, say more about the standard than about the system.

---

1. Introduction

An agent that can act on a system is worth more than an agent that can only describe what should be done, and the industry has settled on breadth as the way to say so. Platforms connected. Verbs shipped. Integrations closed. None of those statements is false, and none of them answers the question a buyer, an auditor, or a regulator is actually asking, which is whether a particular action, on a particular account, has ever been observed to happen.

In July 2026 an evidence-gated audit was run against the system described in this paper. Its posture was hostile by construction: the default verdict for every capability was failure, and a capability moved up only on a file and line reference. The corpus was 130 canonical write capabilities across six vendor surfaces, weighted by tier so that money-moving operations counted for more than configuration ones.[1]

The audit produced four coverage figures. Not four estimates of one quantity, and not the output of four methods. Four figures computed from the same matrix by the same script, differing only in the definition of evidence supplied to it.

Counted as breadth, coverage was a narrative rather than a number. The internal record is explicit that the figure quoted in commercial material was a paraphrase of the closed-platform and verb-count story, and that no value of that kind was ever computed from the capability map.[2]

Counted as code that exists, is reachable from a real trigger, is authorized, and whose vendor contract appears correct, coverage at the audit commit was 56.6%.[3]

Counted with the additional requirement that the permission scope be enforced at the writer boundary before the call, rather than merely requested at consent and stored, the same corpus scored 5.4%. On one platform the figure moved from 73.7% down to 1.3%, and the distance between those two numbers is precisely one unshipped check.[4]

Counted as writes proven to have reached a vendor and returned a success status, coverage was 0.0%, on every platform and every tier.[5]

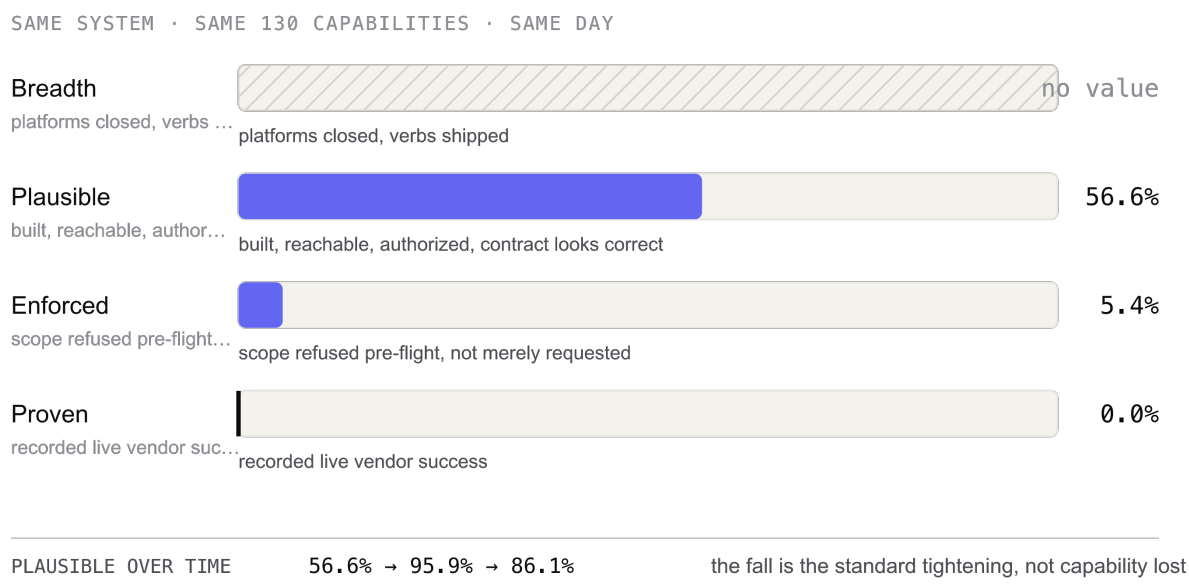


Figure 1. One capability map, four coverage figures. The breadth bar is drawn hatched and unlabelled because no value of that kind was ever computed; the other three are recomputed values from the deterministic score scripts named in the notes. The inset trajectory shows what happened to the plausible figure across two later engineering sprints.

The trajectory in that figure carries the paper's second argument. Four remediation sprints moved the plausible figure from 56.6% to 95.9%. The next sprint moved it down to 86.1%, because one platform's writers were reclassified as vendor-blocked after a scope-matching defect made it impossible for any of them to execute. Nothing was deleted and no capability was lost. The measurement got stricter, so the number got worse, and it got worse by getting truer.[6]

We take three positions in this paper. First, that action capability without a stated evidentiary basis is not a claim at all, because any figure between a breadth narrative and zero can be defended from the same code. Second, that the evidence classes the field currently accepts, including passing tests and a clean typecheck, fail in ways that are specific, reproducible, and worth naming individually. Third, that a standard is only worth proposing if its author is willing to be graded by it first, which is what Section 7 does.

2. Scope and terminology

The definitions below are numbered so later sections can cite them precisely. They are narrower than ordinary usage on purpose: most of the ambiguity this paper is about lives in words that felt unambiguous until somebody tried to grade them.

D1. Action. A single intended mutation of state held by a system outside the acting party's own datastore, addressed by a named verb. Changing a campaign budget is an action. Recording an intention to change it is not, and neither is flipping an internal flag that a scheduled job will later read.[7]

D2. Actor. The principal on whose authority an action is issued. Three are distinguished throughout: a human operating a surface directly, a human approving a proposal that software composed, and software firing on a schedule with no human in the loop at that moment. An evidentiary record that does not name the actor cannot support any claim about autonomy, because the strongest and weakest cases produce identical rows. Only the second is sufficient for Tier 5. A proposal that software composed and a named human approved leaves an approval record, and that record is what the ladder accepts as the actor. A human operating a surface directly leaves no approval record, and software firing on a schedule has no human to leave one; both are recorded, and both stop at Tier 4 however strong the rest of their evidence is. Until revision 1.3 this definition named all three as actors without saying which of them Tier 5 accepts, while the per-row grader required the approval record. The grader was right, and this definition was changed to match it (Section 9).

D3. Surface. The fully-qualified vendor entity an action addresses, including the API version. Not the endpoint family and not the resource name in isolation. A later section explains why this precision is load-bearing: in one vendor's schema, three distinct messages share a single unqualified name, and a field verified against the global vocabulary can be absent from the specific message a writer actually sends.

D4. Write mode. What the terminal call actually did. Four values are distinguished here, and the fourth is the one usually missing: live, meaning the vendor was called; shadow, meaning a real vendor call was deliberately suppressed; test, meaning a live call against a disposable target; and local-configuration, meaning the verb has no vendor call in its definition at all. Collapsing the fourth into either of the first two corrupts any later count of executions, in opposite directions.[8]

D5. Reversibility class. A declared property of a verb, with three values. Full means an inverse verb exists that restores the prior state. Semi means an inverse exists that stops the effect without restoring what it consumed, which is the correct grade for undoing a duplicate that already spent money. None means no programmatic inverse is shipped, which describes the acting party's capability rather than the vendor's. Two obligations attach to the class: a verb graded none requires an explicit second-stage confirmation before it may fire, and a verb that creates a new entity never fires automatically regardless of its class.[9]

D6. Idempotency. A property of the action's lifecycle, not of the request. A duplicate-suppression check that only recognises completed prior attempts leaves a window open between the moment a vendor call is issued and the moment its result is recorded, and a crash inside that window makes a retry look like a first attempt. The correct scope of the check is every state an attempt can be in, including in-flight and unresolved.[10]

D7. Requested scope against enforced scope. A requested scope is one the acting party asked for at consent. A stored scope is one a provider said it granted. An enforced scope is one that will cause the acting party's own code to refuse the call before it is made. These are three different things, only the third has any effect on what actually happens, and a system can hold the first two for years without holding the third.

D8. Evidence artifact. A durable record that a specified party can inspect without trusting the acting party's narration. The specification of who can inspect it is part of the definition. A log line the vendor cannot see, a screenshot the reader cannot re-derive, and a test the author wrote against their own implementation are all artifacts in the loose sense and none of them is an evidence artifact in this one.

D9. Grade. The highest tier in Section 4 whose criterion a given capability satisfies, together with the artifact that satisfied it. A grade is per capability, per surface, and per point in time. It is not a property of a product.

3. Why existing evidence fails

Each subsection below names an evidence class in current use, the mechanism by which it fails, and an instance from our own record. Every instance is a defect we shipped. That is the only reason they can be described this precisely.

3.1 The demonstration

A recorded demonstration shows a user interface reaching a state. It does not show a vendor receiving a request, because the interface is downstream of the acting party's own code and renders whatever that code returns. Our write layer is gated by a per-platform mode switch whose default suppresses the vendor call and returns a tagged envelope. That envelope is deliberately explicit, and it logs at critical severity rather than pretending, but a surface that renders the result as success would look identical on video to one that changed a real account. A demonstration is evidence about a rendering.

3.2 The sandbox run

A sandbox run is a real execution and is a genuine improvement on a demonstration. Its limit is scope: it establishes that the code path works against the sandbox, and vendor sandboxes routinely differ from production in permission model, rate limiting, and object lifecycle. A sandbox result is admissible evidence for the tier it supports and is not evidence of production capability. The standard in Section 4 therefore treats the target as part of the artifact rather than as a footnote to it.

3.3 The self-reported log

A log written by the acting party is evidence only to the extent that its fields are hard to write incorrectly. Two of our verbs stamped an execution mode of live on operations that made no vendor call whatsoever: they flipped an internal flag that a scheduled job would later act on. Any query asking how many live writes had occurred was counting them. The fix was a distinct mode value for the case, chosen deliberately over reusing the shadow value, because shadow asserts that a real write was suppressed and nothing had been suppressed.[8]

The second failure of self-reported logs is subtler and we have three instances of it. A stored vendor response that reads as a bare success object, with no vendor-assigned identifier and no request id, is indistinguishable from an envelope the application synthesised for itself. Under our own standard those rows grade as ungradeable, and the backfill script we wrote to classify historical rows leaves them null rather than guessing.[11]

3.4 The passing test suite

This is the failure with the highest ratio of confidence to information, and it is the one we most want other teams to reproduce on their own code.

Our integration code, written with LLM assistance, called a Klaviyo endpoint that has never existed. The path it used was a send sub-resource on the campaign object. The real operation is a separate top-level resource, scheduling is a two-step flow that first patches a send strategy onto the campaign and then creates the send job, and the vendor documents no idempotency header for it, so duplicate suppression has to be built at the application level for an irreversible operation. This was our defect and not the vendor's: the documentation was correct and public the whole time, and the code did not consult it.[12]

The suite was green throughout. It was green because the transport mock was shaped to the implementation rather than to the vendor. A mock that resolves successfully regardless of the path it is handed blesses every path, including paths that do not exist. We classified our own writer tests against this criterion and found the class was not rare: two Meta test files could not have caught an invented endpoint, one because it asserted only that the transport had been called and the other because it asserted request bodies while never asserting the URL.[13]

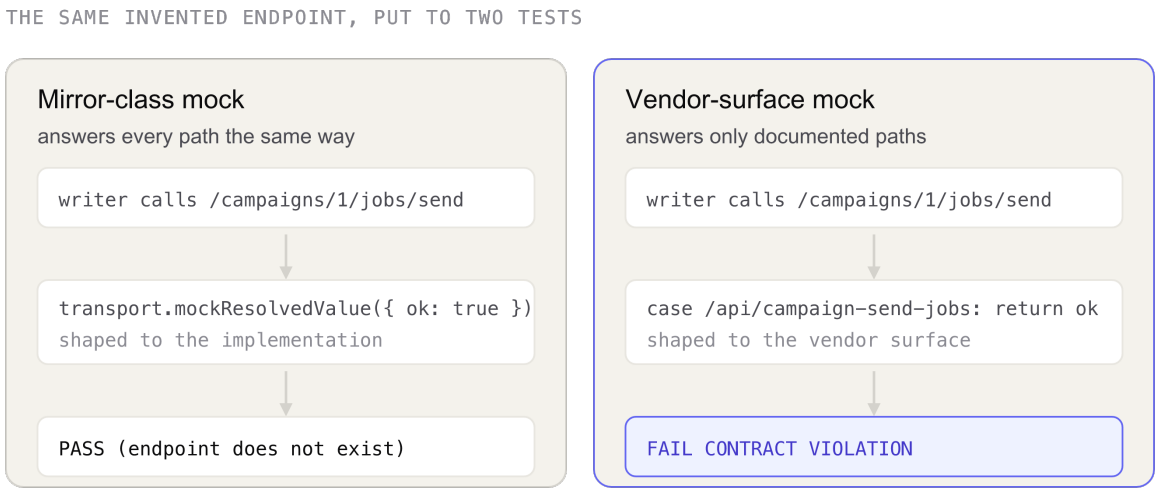


Figure 2. The same invented endpoint under two mock shapes. The left panel is not a weaker test of the same thing; it is a test of a different thing, namely whether the implementation calls what the implementation calls.

The mutation test. The remedy is a mock shaped to the published vendor surface, and the only way to know it is shaped correctly is to break the implementation on purpose and watch the test notice. The procedure below is reproducible on any codebase with a single transport chokepoint per vendor.

1. Identify the one function every writer for a vendor calls to reach the network. If there is more than one, that is the first finding.
2. Fetch the vendor reference for each endpoint the writers use, and record the URL and the retrieval date alongside the writer.
3. Replace the transport mock with one that answers only the paths and methods named in that reference, and that throws on anything else with a message containing a fixed, greppable marker.
4. Run the suite. It must pass. If it fails here, the implementation is already calling something undocumented, and that is the whole finding.

5. Change one writer's endpoint to a path that is plausible and wrong. Prefer a real endpoint of the same vendor over an invented one, because the interesting failure is a wrong real path rather than a typo.
6. Run the suite again. It must fail, and the failure message must contain the marker and name the wrong path. If it passes, the test is mirror-class and proves nothing about the endpoint.
7. Revert the change and confirm the working tree is clean before the result is recorded.

Run against our own writers, this produced the following. Changing the pixel-creation endpoint to a plausible neighbour caused two tests to fail with the marker naming the wrong path. Changing the campaign-creation path to a legacy name caused a failure. On a second vendor, changing an inventory write to a different and genuinely real endpoint of the same vendor produced three failures. On Klaviyo we re-introduced the program's actual historical invention, the send sub-resource, and it now fails loudly where the identical change had previously stayed green. Each change was reverted and the empty diff verified.[14]

The rule this yields is short. Shape the transport mock to the vendor surface, never to whatever the implementation happens to call.

3.5 The clean typecheck

A type system checks the code against its own declarations. When the declarations are generic, it checks almost nothing about the vendor. Upgrading our Google Ads client across three major versions renamed two fields on the campaign resource, replacing a start date and an end date with date-time equivalents carrying new field numbers. The compiler reported no errors, because the mutation helper is generic in its resource type and accepts any object literal. One writer would have created a campaign whose start date was silently dropped, so the campaign would have begun serving immediately rather than on the requested day. The other selected the removed field in a query and would have failed outright before reaching the mutation.[15]

The first verification pass missed this. It checked field names against the vendor's global vocabulary for that version, and both removed names still exist on other message types, so the check came back clean. A bare name lookup also lies in a second way: three distinct messages in that schema share the unqualified name Campaign. The method that survives is verification against the fully-qualified message or endpoint, never against the vocabulary, and never with a clean compile standing in for it. Re-run that way, the audit covered 22 of 22 mutate entity names, 7 of 7 service methods, 26 of 26 enum types, 0 renumbered values across all 348 enums, and 77 of 77 field names in use.[15]

3.6 The requested OAuth scope

A consent screen records what was asked for. It does not record what will happen when the code runs. Our writers requested and stored an ads-management scope on one platform for months without any check that refused a call when the connection lacked it, so a connection that had consented before the scope was added would attempt the mutation and receive a refusal from the vendor, while the operator saw a success. The shared pre-flight that closes this shipped later, wired into every write-preparation chokepoint and failing closed on a read error.[16]

Reading a granted scope back turns out to be its own family of defects, and the four we found do not share a root cause, which is the point.[17]

- **A sentinel matched as a substring.** One platform's check tested whether the granted-scope list contained a short digit string. Any scope identifier containing that digit satisfied it, so the check passed for connections that had never been granted anything.
- **A comparison that can never hold.** The same platform is documented to return numeric scope identifiers, which would be stored as strings and compared against scope names. That mismatch is documented behaviour we have not observed on a real token, and we say so rather than fixing it against identifiers we would have to invent.
- **A readback list shorter than the consent request.** A third platform's readback omitted two of the scopes its own consent flow asks for, so the interface reported a store as write-capable while it lacked them.
- **No readback at all.** A fourth vendor publishes no endpoint that returns an API key's own scopes, which leaves inventing one, probing with a real write, or asking a human. We chose the third, and store it as an attestation carrying who recorded it and when, deliberately kept out of the field that means the provider told us this.

Evidence class	How it fails	Instance from our record
Demonstration or screenshot	Shows a rendering, which is downstream of the acting party's own code	Shadow mode suppresses the vendor call and the surface still renders a result
Sandbox run	Establishes the sandbox, which differs from production in permissions and lifecycle	Treated as tier-limited evidence rather than proof of production capability
Self-reported log	Fields that are easy to write incorrectly, and envelopes indistinguishable from synthesized ones	Two verbs stamped live with no vendor call; three stored responses ungradeable
Passing unit tests	A transport mock shaped to the implementation blesses any path, including one that does not exist	An invented Klaviyo send sub-resource stayed green until the mock was reshaped
Clean typecheck	Generic resource parameters carry no vendor schema, so removed fields are invisible	Two renamed campaign date fields survived a three-major upgrade with zero compiler errors
Requested OAuth scope	Consent is not enforcement, and readback has its own failure family	A scope requested and stored for months with no pre-flight refusal anywhere

4. The standard

The ladder below has seven tiers. Each is stated as a criterion a third party can apply without access to the code, the artifact that satisfies it, and the party able to verify that artifact. Tiers are cumulative: a capability graded at a tier satisfies every tier beneath it. A grade attaches to one capability on one surface at one time, never to a product.

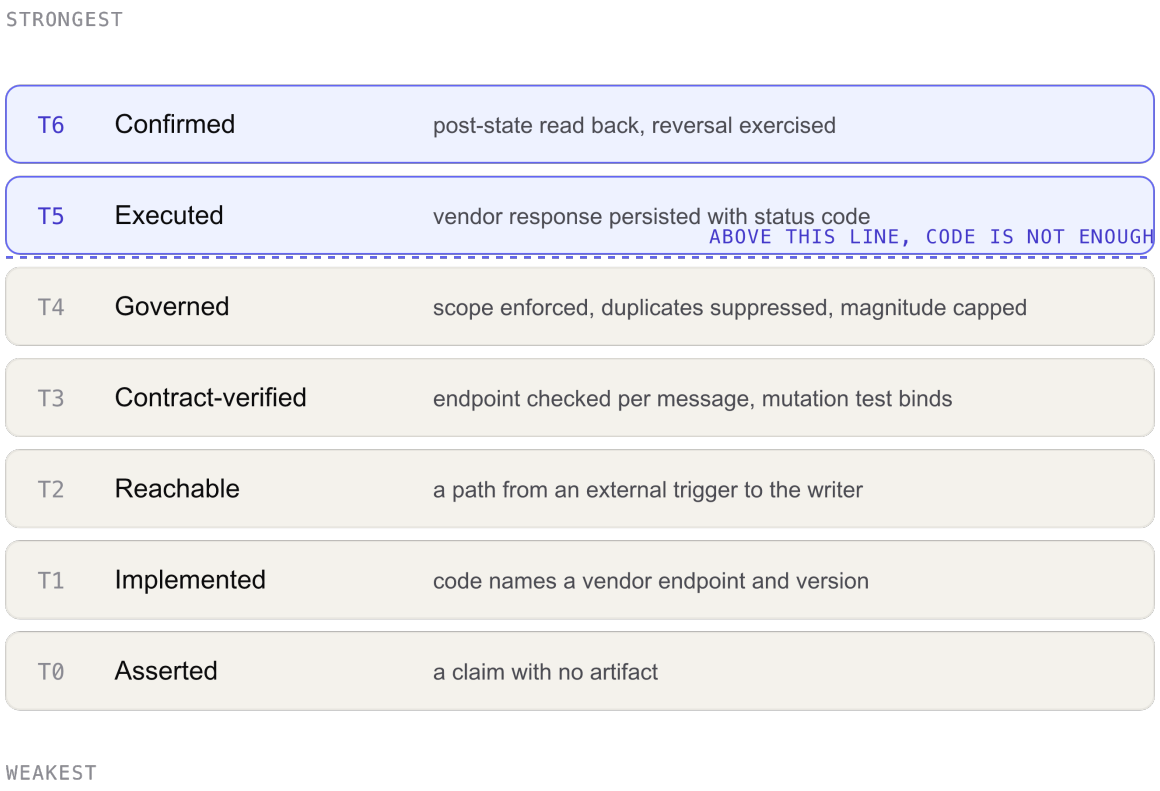


Figure 3. The evidence ladder. The dashed boundary is the load-bearing feature: everything beneath it can be reached by writing code, and nothing above it can.

Tier 0, Asserted. A capability appears in a claim with no artifact behind it. Criterion: none. This tier exists so that the ungraded case has a name, because in practice most published capability claims sit here and are not labelled.

Tier 1, Implemented. Code exists that names a specific vendor endpoint, method, and API version for the capability. Artifact: a published manifest listing, per capability, the endpoint, the method, the version, and where the version is sourced from. A version written as a literal in the code rather than read from configuration is a defect at this tier, because it is a scheduled failure with no owner. Verifier: anyone, once the manifest is published; the manifest is checkable against the vendor's reference without seeing the code.

Tier 2, Reachable. A path exists from an external trigger to the writer, and it is the path a user would actually take. Artifact: the trigger chain, named end to end. Verifier: anyone with access to the product surface. A writer with no trigger is a library, and a great deal of published capability is exactly that.

Tier 3, Contract-verified. The endpoint, method, and request fields have been checked against the vendor's published reference for the fully-qualified message, and a mutation test demonstrates that the check binds. Artifact: the reference URL with its retrieval date, plus a recorded mutation result showing the test failing on a deliberately wrong path and passing after revert. Verifier: anyone who can re-run the procedure in Section 3.4. This is the lowest tier at which a test result means anything, because it is the first tier where the test is shaped by something other than the implementation.

Tier 4, Governed. Four conditions, all of which precede the vendor call. **4a, scope enforcement:** the required permission scope is enforced at the writer boundary and refuses before the call, failing closed when the scope cannot be read. **4b, duplicate suppression:** the check covers every state a prior attempt can be in, not only completed ones, and an unresolvable in-flight attempt blocks rather than proceeds. **4c, magnitude ceiling:** a limit applies to the operation, expressed in the unit the operation moves rather than in a count of operations. **4d, explicit grant:** the capability is granted to each actor per verb, because a pattern-matched grant admits a new verb silently the moment it is registered. Artifact: a refusal record for each condition, produced by exercising it. Verifier: anyone able to present a deliberately unscoped or duplicate request and observe the refusal.

A capability that satisfies some of the four but not all of them grades at Tier 3, with the satisfied sub-conditions listed alongside the grade. Partial governance is worth recording and is not worth a tier, because the conditions do not substitute for one another: a magnitude ceiling does not help a write that should have been refused for scope, and an enforced scope does not stop a duplicate.

Tier 5, Executed. A vendor response has been received and persisted with its transport status code, in a field that cannot be confused with an application-generated envelope, alongside an approval record naming the human who approved it (D2), the timestamp, the write mode, and the fully-qualified target. Artifact: the persisted row. Verifier: an auditor with read access to the record, or a counterparty who can be shown it. A row that omits any of those fields grades ungradeable, which is a distinct outcome from failing and must be reported as its own category.

Evidence strength within a tier. A Tier 5 grade records where its post-state came from. A post-state the vendor reported on a separate read call after the write ranks above one computed from the write's own parameters, and Tier 6 requires the first. Evidence strength orders records that meet the same conditions and never stands in for a missing one: a read-back record with no approval record stays at Tier 4, below an approved record whose post-state was computed, for the same reason the sub-conditions of Tier 4 do not substitute for one another. Until revision 1.3 a tier was the whole grade, and Section 7.7 records what that cost.

Tier 6, Confirmed. After the write, an independent read call confirms the intended state, and the declared reversal has been exercised against a real entity and confirmed the same way. Artifact: the pre-state, the post-state, the reversal, and the post-reversal state, all captured by read calls rather than inferred from the write response. Verifier: an auditor, or the account holder. A reversibility class that has never been exercised is a design intention, and a rollback path that has never run is the least trustworthy code in any write system.

Two properties of this ladder matter more than the tier names. The first is the boundary in the figure: Tiers 1 through 4 can be reached by writing code, and Tiers 5 and 6 cannot be reached by any amount of it. A team can work for months and move nothing above the line. The second is that the tiers above the line are the only ones a counterparty can verify without trusting the acting party's description of its own internals, which is why they are the only ones worth putting in a claim.

The standard also settles what counts as a write at all, which sounds pedantic until a count depends on it. An operation that changes only the acting party's own state is not an action under D1, cannot reach Tier 5, and must be recorded in a mode value distinct from both live and suppressed. Any capability count that includes such operations is measuring something else.

5. Verification procedure

The following establishes Tier 6 for exactly one capability, on one surface, once. It is deliberately boring and deliberately narrow. Producing this record for a capability is cheap; the reason it is rare is that it cannot be done by the people who wrote the code without also being done by somebody who controls an account.

1. Nominate a disposable target: an account, an entity, and where relevant a property, that nobody depends on. Record its identifier in configuration, never in the writer, so the writer is the same code that will run against a real account.
2. Establish the token's scope. Read the granted scopes from the provider and record the response. Where the provider exposes no such endpoint, record an attestation naming the person, the source they read, and the time, and store it in a field distinct from provider-reported scopes so the two can never be confused later.
3. Capture the pre-state with a read call against the target, and persist it. Do not derive it from local state, which is the copy most likely to be wrong.
4. Issue the write. Persist the request path, method, and API version actually used, resolved at call time rather than transcribed from the source.
5. Persist the vendor response verbatim, with its transport status code, in a field reserved for it. If the response carries a vendor-assigned identifier, persist that separately, because it is the field that later distinguishes a real response from a synthesized one.
6. Read the post-state back with an independent read call and compare it against the intent. A write response that echoes the request is not a confirmation.
7. Exercise the declared reversal against the same entity, then read back a third time and confirm the prior state was restored, or record precisely what the reversal did not restore.
8. Grade the resulting record. Actor, timestamp, write mode, status code, pre-state, post-state, reversal, and target must all be present. If any is missing, the grade is ungradeable, and ungradeable is reported as its own outcome rather than folded into either success or failure.

Two notes on the procedure. The reversal in step seven is the step most often skipped, and it is the one that converts a reversibility class from a declaration into a fact. The grading in step eight has to admit a third outcome, because a system that reports only success and failure will silently classify its own missing evidence as one or the other, and which one it picks tells you about the reporter rather than the write.

6. What the standard cannot establish

The standard grades whether an action happened and whether it can be shown to have happened. It is silent on four questions that matter at least as much, and reading a grade as an answer to any of them is a misuse.

It says nothing about frequency. A capability graded at the top tier has been proven once. Once is the entire content of the claim, and a system with one confirmed write and a system with a million of them earn the same grade on that capability.

It says nothing about safety. A confirmed write is a write that happened and was recorded, which is orthogonal to whether it should have. The governance conditions at Tier 4 constrain the blast radius of a bad decision; they do not evaluate the decision.

It says nothing about economic value. Whether an action improved anything is a question of measurement design, counterfactuals, and time horizon, and none of that is in scope here.

It says nothing about unattended behaviour. Every tier is defined for an action, and an action has an actor. A system that has proven a capability under human approval has proven exactly that, and the case where no human is present is a different claim requiring different evidence, which we do not attempt to specify. Section 7.11 records the first unattended live write in our own table, and it illustrates the point: it proves the mechanism once and says nothing about the decisions the system would have made.

7. Applying the standard to Cresva

We wrote the standard, so this section is where it is worth the least and has to be done anyway. Everything below is graded by the ladder in Section 4 against the state of the repository at the time of writing, and section 7.11 grades the production record as it stood on 12 October 2026.

At the audit commit, the count of capabilities at Tier 5 or above was zero. No capability in our write surface had a persisted vendor response with a transport status code tied to an actor and a target. Not one, on any of six platforms. Every capability we shipped sat at Tier 4 or below, which meant every claim we could honestly make about it was a claim about code rather than about anything that had happened. Section 7.6 records the September revision, which reported one and found that engineering was not what moved it. Section 7.11 records a re-measurement on 12 October 2026, which grades rows with the shipped grader, puts seven at Tier 5 covering five capabilities and none at Tier 6.

That is the finding, and the rest of this section is context for it rather than mitigation of it.

7.1 What the enforcement collapse measured

The 5.4% figure in Section 1 was a recompute of the same matrix under one changed rule: a capability earns credit for authorization only if the writer refuses a mis-scoped connection before calling the vendor. Under that rule, three of the six platforms scored 0.0%, one fell from 73.7% to 1.3% on the strength of a single writer that enforced its own scope, and the two that were unchanged were unchanged for opposite reasons, one because it already failed closed and one because its vendor has no scope layer to drift.[4]

That gap is now closed in code by a shared pre-flight at every write-preparation chokepoint.[16] It is not closed in evidence, and under this paper's own standard those are different states. The refusal path has unit tests. Measured on 27 September 2026, the production action-audit table holds no refusal for scope: the refusals it does hold are for a brand whose live writes were not armed, for proposals the merchant had already rejected, and for a list that had gone stale. So live writes now happen, and the scope check has still never been observed refusing a real mis-scoped connection.[22]

7.2 The cost of failing closed

Failing closed is not free and we paid for it in the expected direction. The scope pre-flight read a top-level column that the OAuth callback had never populated, because the callback wrote the granted scopes into a metadata field instead. Every Meta connection therefore read as having no scopes at all, and on an armed platform the pre-flight would have refused every write, including all the correctly scoped ones. The fix was a resolution chain across the places the data actually lived, plus a canonical write going forward and a backfill for existing rows.[18]

We record this as the correct failure. A control that refuses when it should have proceeded costs an operator an approval and a support conversation. The same control failing open costs a customer's account. A standard that demands fail-closed behaviour should be honest that it is buying the first outcome with the second, rather than pretending the first does not happen.

7.3 Three writes we cannot grade

The production action-audit table is small enough to read row by row, and at the time of the migration verification it held five rows.[11] Three of them carry a stored vendor response that reads as a bare success object with no vendor-assigned identifier. One of those three is a write against a real campaign whose captured pre-state carries a lifetime-budget value of a shape only a real vendor would return, which is suggestive and is not evidence. All three were issued against internal targets rather than customer accounts.

Under the standard in Section 4 those three rows grade ungradeable, not executed. We could argue for them. The pre-state looks real, the timing lines up with a period when one platform was configured live, and the operations were the kind that would have succeeded. Every one of those is an inference about our own system offered by us, which is the precise thing D8 excludes. The backfill script that classifies historical rows leaves them null and says so in its output rather than guessing, and this paper does the same.

There is a second-order finding here. Our own audit trail was, until recently, capable of recording a write in a way that could not later be graded, and nobody noticed because nothing queried it for grading. An evidence system that is never queried degrades silently.

7.4 The auditor is not exempt

Two of our own audit findings were wrong, in both directions, and both were caught by reading code rather than documents.

The first was pessimistic. An audit reported that the spending guardrail was a count cap rather than a currency cap and that it failed open on a database read error, and a remediation sprint was scoped to rebuild it. Inspection of the evaluator showed a cumulative per-day currency cap with a binding default, read strictly, enforced on every path including the scheduled one. The finding had been carried forward from an internal document that predated the fix. The sprint was rescoped to extend the existing control rather than rebuild it, and the stale documents were flagged for correction.[19]

The second was optimistic, and it is the more instructive one. The original audit concluded that production arming was unverifiable from the repository, and scored a platform accordingly. It was verifiable, from the deployment platform's environment listing, which the audit did not consult. The correction is recorded in an addendum rather than folded into the original text, because an audit that silently rewrites its own conclusions is not an audit.[20]

The rule both cases produce is the same one that governs Tier 3. Documentation is not evidence, in either direction. A document asserting a defect and a document asserting a fix are the same class of artifact, and the code and the deployment are the authorities.

7.5 Where each platform actually stands

Every remaining blocker is an operator action or a vendor observation. None of them is a code change, which is a comfortable thing for an engineering team to say and was also why the count at the top tier stayed at zero through the period this table describes: the work that remained was not the work we are good at. One of those operator actions was taken on the Shopify row, which is the whole of the movement recorded in 7.6. The Meta and Shopify rows below are updated to 12 October 2026 from the grading in 7.11. The Google Ads, Klaviyo, GA4 and TikTok rows are as the sprint records left them: their blockers were not re-measured for this revision, and the production table holds no live write on any of the four.[21]

Surface	Highest tier reached	What stands between it and Tier 5
Meta	Tier 5	For Tier 6: an approved write whose post-state is read back and whose declared reversal is recorded live and read back the same way. Reversal chains exist, on rows run directly by an operator that carry no approval record
Google Ads	Tier 4	Re-consent the connections whose scope provenance is unprovable, and nominate a test customer and base entities
Shopify	Tier 5	For Tier 6: a post-state read back from the vendor rather than computed from the write, and a reversal recorded with an execution mode and a vendor status code
Klaviyo	Tier 4	Record the key's scopes as an attestation, since the vendor publishes no endpoint that reports them
GA4	Tier 4	Nominate a writable test property and record the measurement credential against it
TikTok	Tier 1	Capture one real token response and read its scope value; the writers are vendor-blocked until the identifier shape is known

One reading of this table is that the system is nearly there. That reading is available because the table stops at the boundary in Figure 3, and the boundary is where the difficulty is. A second reading is the one we hold: the table describes a write engine that has been observed to turn over on its operator's own account and on no customer's, and the standard exists so that the first reading cannot be published without the

second.

If this section reads as favourable, the criterion is wrong. That is not modesty. A standard authored by a participant is only useful if it can produce an unflattering result about the author, and the test of whether it can is whether it did.

7.6 The count moved, and not for a reason this standard anticipated

Between publication and September 2026 the count of capabilities at Tier 5 or above went from zero to one. No engineering caused it. The capability that crossed had been built, reachable and governed since before the audit, and what changed was that somebody clicked a button in our own dashboard against our own store, and that a different platform's connection was deleted.

That is worth stating first because it is the finding. A standard whose top-tier count is decided by an account deletion is a standard with a defect in it, and the sections below name three.

At that revision, eight rows in the production action-audit table carried a live vendor response with its transport status code, alongside the actor, the timestamp, the write mode and a fully qualified target. Four distinct capabilities were represented, out of a corpus of 130. Seven of the eight rows were on Meta and one was on Shopify. Every one was issued by a human on the operator's own account. None was issued for a customer. Section 7.11 gives the same counts as of 12 October 2026.

The denominator is 130 and it is worth saying why. That is the count of canonical write capabilities in the audit corpus, which is the unit this paper has counted in throughout. It is not 129, the number of rows the published capability map held until recently, because a later edit merged two Shopify capabilities into one; the audit scored them separately and the notes record the reconciliation. It is not the number of write verbs the system registers, which is larger by a factor of about two and describes a different thing at a finer granularity. A paper whose subject is that the unit decides the number does not get to change units when the ratio is unflattering. One of 130 is 0.77%.

7.7 The standard ranked its own evidence backwards

Of the eight rows, five carry a post-state captured by an independent read call against the vendor after the write. Three do not: their post-state is computed, meaning the intended state was copied forward from the write's own parameters and never read back. Independent read-back is the provenance requirement that separates Tier 6 from Tier 5, so on evidence quality the five are the stronger rows by the standard's own definition.

All five of the read-back rows are on Meta. The row that gets graded is one of the computed three.

The reason given at the time was that no Meta connection existed in production for the brand those five ran on, and a capability that cannot be re-checked against the vendor cannot be graded. The reason is defensible in principle and its premise did not hold: section 7.11 records that the check looked only for a connection attached to the brand, while a Meta connection held at the company level was active throughout. The result is that the standard promoted its weakest evidence and discarded its strongest, and did so without any of the tiers noticing, because tier and provenance are tracked separately and nothing compares them.

We record this as a defect in the ladder rather than an accident of our data. A grading scheme in which a row with a read-back post-state can rank below a row without one has an ordering problem, and the ordering is the entire content of a ladder. Revision 1.3 fixes it in Section 4: the post-state's provenance is now part of the grade, and a read-back record ranks above a computed one at the same tier.

7.8 A rule that decides the count in both directions, and is not in the ladder

The rule that disqualified the five is that a capability requires a live, currently checkable connection to be graded at all. It first appears in an addendum dated 29 August 2026. It is not in Section 4. It was not part of the standard when the standard was published.

It is not a small rule. It is doing all the work in both directions: it is why seven Meta rows do not count, and it is why the Shopify row does. Remove it and the count is four, on rows we consider better evidenced. Keep it and the count is one, on a row we consider worse evidenced. A criterion introduced after publication that determines the headline in both directions either belongs in Section 4 with a justification a third party can apply, or it does not belong.

It was applied before it was argued, and a rule applied before it is argued is indistinguishable from a rule fitted to a result. Section 4 does not carry it yet. Revision 1.2 updates the measurements only, and on 27 September 2026 the rule's premise, a missing Meta connection, turned out not to hold, so the five rows it disqualified are no longer disqualified by it. Revision 1.3 drops the rule. It was never in Section 4, it is not in Section 4 now, and no grade in this paper depends on it; Section 7.11 gives the count without it.

7.9 The graded capability carries an obligation it cannot meet

The Shopify write set a product to draft, returned a 200, and was reverted eleven hours later, well inside its reversibility window. The reversal ran. It is recorded.

It is recorded with no execution mode, no vendor status code and no vendor-ok flag. Whatever reached Shopify, our own audit trail does not show it as a live vendor call, so the reversal cannot be graded at Tier 5, let alone satisfy Tier 6's requirement that a declared reversal be exercised against a real entity and confirmed by a read call.

Read again on 27 September and on 12 October 2026, that reversal row is unchanged: still no execution mode and no vendor status code.[23] Section 4 calls a rollback path that has never run the least trustworthy code in any write system. Ours has run. We cannot prove what it did. The single capability at the top of our ladder is therefore a capability whose reversal is unproven, which is a worse position than the sentence "the count is one" conveys, and it is the position.

7.10 What we take from it

At the September revision the count was one, out of 130, on our own account, by our own hand, on the row with the weakest provenance of the eight, with an unrecorded reversal. Every clause in that sentence is load-bearing and we would rather print all of them than the first.

One measurement moved in the other direction and is worth recording next to the count. The remediation sprint that followed the audit left the plausible figure exactly where it was, at 56.6%, and moved the maturity figure down from 45.2% to 44.0%. The sprint's only gain came from promoting a single Klaviyo capability, and that promotion has been withdrawn on the row's own evidence: it was recorded as contract-correct in the same breath as a note reading that the endpoint likely returns a 404. What remains is six Shopify capabilities correctly demoted a tier by the same review. A sprint that makes the measurement stricter and leaves the headline unchanged is the better outcome of the two, and it is the one that happened.

Section 7 predicted that a favourable reading of this section would mean the criterion was wrong. This revision is the first test of that and it half fails: the count went up, and the three defects above are why the increase is not evidence of progress. A standard that its author can move by deleting an OAuth connection is not yet doing the job. That revision promised three things: to fix the ordering, to move the gradeability rule into Section 4, and not to report a top-tier count again until a reversal is recorded with the same fields as the write it reverses. Section 7.11 is the next revision. It keeps the third promise, and revision 1.3 keeps the first two: the ordering is fixed in Section 4, and the gradeability rule is dropped rather than moved there.

7.11 Re-measured on 12 October 2026

This subsection grades the production record as it stood on 12 October 2026, queried through a database role that can select and cannot write, and it counts only rows on brands recorded as real. Revisions 1.2 and 1.3 measured the same table on 27 September 2026; where a figure has moved since, the earlier one is given beside it. Revision 1.2 did not change Section 4; revision 1.3 did, and every grade below uses the ladder as revised there. It uses the per-row grader that has been in the codebase since 30 August 2026, which applies the row-level conditions of Tiers 5 and 6 to each action-audit row, instead of grading by hand as the September revision did.[24]

Measured on 12 October 2026, the production action-audit table holds 90 rows on brands recorded as real. Eighty-four of them are the actions the home page counts on its record (81 on 27 September). The other six are refusals, where an agent declined to raise again something the merchant had already rejected, or a list was too old to run, so nothing was proposed, approved or sent. Five rows the 27 September count included now sit on two brands that have since been recorded as synthetic, and leave every figure here. The table also holds rows on our synthetic demo brands, which no figure in this paper counts. Twenty-three rows carry an execution mode of live (17 on 27 September). Twenty-one are on Meta and two on Shopify, the first created on 5 August 2026 and the latest on 9 October 2026. They span the same six verbs: campaign pause, campaign resume, ad creation with a creative and an ad set budget change on

Meta, and a product status change and a collection creation on Shopify. Twenty-one of the twenty-three carry a vendor status code, and every one of those is a success.[22]

All twenty-three are on one brand, and it is ours. Of the 16 brands in production recorded as real rather than seeded on 12 October 2026 (17 on 27 September), one is allowed to write live to a vendor, and it is that one; the other fifteen refuse, which has been the default since 26 September 2026. No live write in the table was made for a customer.[22]

Of the 84 rows the home page counts, the grader puts seven at Tier 5 and none at Tier 6, the same seven as on 27 September. No row written since then grades at Tier 5: the six live rows added are pauses and resumes run directly by an operator, with no approval record. Of the seven, five are on Meta: three campaign pauses, one ad creation and one ad set budget change. Two are on Shopify: the product status change and the collection creation. Under the ladder as revised in Section 4, four of the seven rank above the other three, because their post-state was read back from Meta after the write; the ad creation and both Shopify rows carry a post-state computed from the write's own parameters. The seven cover five distinct capabilities, counting a capability as one verb on one platform.[24,31]

Why none reaches Tier 6. Nineteen of the twenty-three live rows carry a post-state read back from the vendor after the write (13 of 17 on 27 September), and every one of those reads matched the intent. Twelve rows have a declared reversal chained to them that reached Meta and returned a success (seven on 27 September). Ten of the twelve were pauses and resumes run directly by an operator: each is read back, none has an approval record, and one of the ten, a pause on 4 October 2026, records no vendor call of its own. Two of the twelve were approved. On one the post-state was not read back. On the other, the human-approved resume of 23 September described below, the post-state was read back and its reversal reached Meta on 4 October 2026, but the resume itself sent nothing to Meta, so there is no vendor response for Tier 5 to grade. Revision 1.2 recorded this as a disagreement: the grader required an approval record before it granted Tier 5, and D2 as then written named a human operating a surface directly as an actor. Revision 1.3 settles it for the grader. Under D2 as revised, a direct operator action with no approval record stops at Tier 4, so those ten stop there however strong their evidence is, and each still names its operator in its principal and user fields. **Tier 6 is zero**, and it is zero because no approved write in the table that reached the vendor has both a post-state read back from the vendor and a reversal that reached the vendor. That is a statement about what we have done, and the ladder no longer lets the strength of an unapproved row argue with it.

The grader and the September hand grade disagreed about Shopify, and the grader was wrong. Revision 1.2 graded both Shopify rows at Tier 4, including the product status change the September revision counted as its one. The grader's target check looked for a numeric identifier at the very end of the mutating call's path, and a Shopify request path ends in a file extension, so the product id in `products/8984659329221.json` could never match. The collection creation failed differently: its vendor-assigned identifier sat in the stored response inside the resource Shopify wraps it in, and in the post-state, and the grader read neither place. Both rows name their entity. Revision 1.3 corrects the check to find an identifier anywhere in the mutating path, in the stored response, or in the post-state, and both rows grade at Tier 5. The correction is listed in Section 9, because a grade that moves when its grader is fixed is a grade whose earlier value somebody may have quoted.[31]

The other 77 are Tier 4 by default, and most were never sent. The grader returns Tier 4 for any row it does not credit, so the figure is not a count of governed writes. Sixteen of the 77 are live writes (ten on 27 September). Fourteen carry no approval record: the ten direct operator actions above, three more operator pauses with no reversal chained to them, and the unattended goal-pacing pause described below. The other two were approved and made no vendor call, so there is no vendor status to grade. The other 61 never reached a vendor: 31 expired before anybody decided, 17 were rejected, 9 failed before the call, and 4 carry no live execution mode. On 27 September those four figures were 34, 15, 9 and 6; the two rejections are new, and the other five are the rows that left with their brands.[31]

The connection that was reported gone was not gone. The September revision set the Meta rows aside because no Meta connection existed for their brand. That search looked only for a connection attached to the brand. Measured on 27 September 2026, and unchanged on 12 October, the account that holds every live row has an active Meta connection held at the company level, created on 23 June 2026, before the first live Meta row, and live Meta writes on 3 September 2026 returned success two days before the revision that called the connection gone. A search bounded to the wrong scope produced the premise of a grading rule, and the rule decided the headline.[22]

The first unattended write, and a row that says less than it should. One of the twenty-three was issued with no human in the loop, and it is still the only one. On 22 September 2026 the goal-pacing tier paused a campaign on its own authority; Meta returned a success and the post-state was read back. The row carries no principal type, because that tier did not stamp one until a code fix on 26 September 2026, which does not reach rows already written. The first unattended live write in the table is identifiable as unattended only through a naming convention in its user field, which is the self-reported log failure of Section 3.3. A human-approved resume of the same campaign, executed on 23 September 2026, is stamped live and carries no vendor status: it read the campaign twice, found it already active, and sent nothing. A row stamped live that made no vendor call is the case D4 exists to name, and the table does not record what reactivated the campaign in between.[22]

What else changed, in code rather than in evidence. At repository HEAD on 12 October 2026, every vendor write outside two audited permanent exceptions passes through one dispatcher, which is the single chokepoint step one of the mutation test in Section 3.4 asks for, and a live write needs its brand on an allowlist that defaults to refusal. The verb-to-platform map lists 269 write verbs across fifteen platform lanes on 12 October 2026 (252 across ten on 27 September), and ten of the fifteen postdate the audit corpus. No new lane has a live row: every live row in the table is on Meta or Shopify. Separately from the action-audit table, nine agent checkout orders were placed on our own development store on 13 and 14 September 2026, across all four checkout sources, and recorded as transactions, and there were still nine on 12 October. They are writes to Shopify, recorded in a different table, and this revision does not grade them against the ladder.[22]

Since 2 October 2026, every write names who authorised it. Before the dispatcher calls a vendor it records on the write's own row who authorised it: a named person, or a named autonomy rule together with the brand's autonomy level, the goal and the grant's ceiling. A write that names neither is refused, and so is one whose authorisation cannot be recorded. Rollbacks now run through the same dispatcher, authorised by the person who pressed the button, so a reversal is recorded with the same fields as the write it reverses. On 12 October 2026 all six live rows written since 2 October name a person this way, and five of them are reversals of an earlier write. This moves no grade. The grader reads the approval record that D2 requires, not this field, and a person who runs a change directly is named by it without having approved a proposal, so under D2 those rows stay at Tier 4.[22]

So the sentence that replaces September's is this one: on 12 October 2026, graded under the ladder as revised in 1.3, seven rows on our own account grade at Tier 5, covering five capabilities, none grades at Tier 6, and no customer account has a live write. The five are counted as verbs on platforms. They were not re-mapped onto the 130 corpus capabilities, so five is not a fraction of 130, and printing it as one would be the substitution 7.6 refused to make. Under revision 1.2's grader the same rows gave five at Tier 5 and three capabilities; the difference is entirely the target check in Section 9.

8. Related work and open questions

The agent evaluation literature measures task success: given a goal and an environment, did the agent reach the goal state. WebArena builds functional websites across four domains and scores functional correctness of the completion.[25] SWE-bench draws its tasks from real repository issues and scores whether the resulting patch passes the project's tests.[26] GAIA scores answers to questions built to be easy for a person and hard for an assistant.[27] tau-bench is the closest of the four to the concern of this paper, because its agents call tools against a domain database and it reports a consistency measure over repeated trials rather than a single pass rate.[28] All four are well designed for what they measure, and their reported numbers are sobering on their own terms: the best agent in the WebArena paper completes 14.41% of tasks against 78.24% for humans, and the strongest model in the SWE-bench paper resolves 1.96% of issues.

None of them answers the question here, and the reason is structural rather than a shortfall. In every case the environment is instrumented by the benchmark author, so the side effect and the record of it come from the same source, and the record is true by construction. In production the side effect lands on a system controlled by a third party while the record is written by the agent's operator, and the gap between those two authors is where every failure in Section 3 lives. The two lines of work are complementary: a benchmark grades the decision, and this standard grades the evidence that the decision was carried out. A system could score well on either while failing the other completely, and knowing which one a claim rests on is the whole point.

One published result bears directly on Section 3.4 and is worth stating precisely, because it establishes that the failure described there is a property of the method rather than an anecdote about one team. A study of package hallucinations in generated code sampled 576,000 programs across 16 models and found 205,474 distinct non-existent package names, at an average rate of at least 5.2% for commercial models and 21.7% for open-source ones.[29] Its concern is supply-chain risk, which is not ours. The shared mechanism is the one that matters: a model will name a dependency that does not exist, confidently and in syntactically valid code, and nothing downstream of the model objects. An invented API endpoint is the same failure aimed at a different target, and the reason ours survived a green test suite is that the test was written against the same assumption as the code. We know of no comparable published measurement for invented endpoints specifically, which is a gap worth someone filling.

Three questions we cannot currently answer are worth naming.

Multi-actor writes. The ladder assumes one actor per action. A plan that fans out into dependent writes, some approved individually and some inherited from the parent approval, has no obvious grading. Grading the plan hides which leg was proven; grading each leg loses the fact that a human approved only the root. We currently record both and have no principled rule for which one a claim should cite.

Unattended operation. Section 6 declines to specify evidence for it, and the reason is that we do not know what would be sufficient. A confirmed write with no human present proves the mechanism and says nothing about the population of decisions the system would have made. This is the question that matters most commercially and is the one this standard is furthest from answering.

A shared registry. A grade is only useful to a counterparty if the grading is comparable across vendors, and comparability requires a shared vocabulary of capabilities, a shared tier definition, and somebody willing to hold the register. This paper supplies the second of those three. We would rather the first and third existed than that we owned them.

Artifacts. The first version of this paper cited its own numbers to paths inside a private repository, which is a failure of definition D8 by the paper's own terms: a reader outside the company could check nothing. Published alongside it, at cresva.ai/research/artifacts/crv-2026-001, are the capability matrix every figure traces to, the three deterministic score scripts that reproduce those figures from a clean directory with no other inputs, the verbatim passage behind every footnote that cites an internal document, and this paper as a PDF with its figures, notes, references and BibTeX entry. Identifiers were replaced with stable placeholders before publication: account and record ids, credentials, infrastructure names, and the display name of any customer or brand. The substitution map is withheld, because publishing it would reverse the redaction in one step, and the bundle README states the policy and its limits in full. The full audit report and the full sprint records are not published, so a reader is entitled to note that the party choosing which passages to excerpt is the party being audited.[30]

One closing observation about the trajectory in the inset of Figure 1. The number went down when the measurement improved, and there was a moment where the honest reporting of that felt like a regression to explain rather than a result to publish. Any standard that a participant applies to itself will produce that moment. Whether it survives it is the only test of the standard that matters.

9. Errata

Corrections to the standard and to the grader that applies it are listed here with their date, so that anybody who quoted an earlier grade can see what moved and why. Changes that only re-measure are not errata and are described where they occur.

12 October 2026, revision 1.4. Section 7.11 is re-measured on that date, which is not an erratum. Three corrections are.

1. **The Tier 6 sentence in Section 7.11.** It said no approved write in the table had both a read-back post-state and a reversal that reached the vendor. Since 4 October 2026 one does: a resume approved in September whose post-state was read back and whose reversal reached Meta, but which itself sent nothing to Meta, because the campaign was already active. It stays at Tier 4, because Tier 5 needs a vendor response. The sentence now says the approved write has to have reached the vendor. No grade moves.

2. **Note 22's query.** It counted the whole action-audit table and set aside the one row then on our synthetic demo brand. After 27 September our demo seeder wrote 223 rows into that table dated 1 to 27 September, so counting the table, or only its rows dated up to 27 September, no longer gave the paper's numbers. The query now counts only rows on brands recorded as real, which the seeded rows are not, and the seeder dates every audit row it writes at the moment it writes it. Those rows were never in a figure this paper prints.
3. **The artifact bundle.** Its archive still held the capability matrix as it stood before the 5 September correction, 129 rows against the published file's 130, and the bundle's own address, which this paper links, returned not found. The archive is rebuilt from the current files, the address serves an index of them, the README names this revision, and the bundle now holds a PDF of the paper.

27 September 2026, revision 1.3. Four corrections, made together.

1. **D2, the actor Tier 5 accepts.** D2 named three actors and did not say which of them Tier 5 accepts, while the per-row grader accepted only a human approving a proposal. The grader was right. D2 and the Tier 5 criterion now say that the actor is shown by an approval record, and that a direct operator action with no approval record, or a write fired by software with no human in the loop, stops at Tier 4. Six Meta pauses and resumes run directly by an operator, each read back and reversed, stay at Tier 4, and Tier 6 stays at zero.
2. **The ladder's ordering.** A tier was the whole grade, so a record whose post-state was computed from the write could rank level with one whose post-state was read back from the vendor, and under the rule in item 3, above it. Section 4 now makes the post-state's provenance part of the grade: at the same tier a read-back record ranks above a computed one, and Tier 6 requires a read-back. Evidence strength orders records that meet the same conditions and never substitutes for a missing one (Section 7.7).
3. **The gradeability rule is dropped.** The rule that a capability needs a live, currently checkable connection to be graded at all was introduced in an addendum after publication, was never in Section 4, and decided the headline count in both directions. Its premise, a missing Meta connection, did not hold. It is not moved into Section 4; it is withdrawn, and no grade here depends on it (Section 7.8).
4. **The grader's target check.** The check for a fully-qualified target looked for an identifier only at the end of the mutating call's path and at the top of the stored response, so it could not read a Shopify path ending in a file extension or a vendor identifier inside the resource Shopify wraps it in. It now finds an identifier anywhere in the mutating path, in the stored response, or in the post-state. Two Shopify rows move from Tier 4 to Tier 5: the product status change and the collection creation (Section 7.11).

Effect on the counts. Over the 81 rows the home page counts, revision 1.2's grader gave five rows at Tier 5, all on Meta, three capabilities, and none at Tier 6. Revision 1.3 gives seven rows at Tier 5, four with a read-back post-state and three computed, five capabilities, and none at Tier 6. The 74 rows at Tier 4 are ten live writes and 64 that never reached a vendor. Items 1 to 3 move no row. Item 4 moves two.[31]

Notes

1. Audit report at `audit/write-access/REPORT.md`, run against repository HEAD `2f1dbc36c`, audit date 2026-07-29. Corpus of 130 canonical capabilities across Meta, Google Ads, Shopify, Klaviyo, TikTok and GA4, tier-weighted three, two and one. Per-capability rows in `audit/write-access/matrix.json`; raw per-platform traces in the sibling EVIDENCE files. Public copy: artifact bundle, note 1. The matrix itself is published at matrix.json.
2. `audit/write-access/PROGRESS.md` records the breadth figure as a paraphrase of the closed-platform and verb-count narrative rather than a computed value, and the report's claim-versus-reality table records that the literal figure appeared in commercial material rather than in the code documentation. Public copy: artifact bundle, note 2.
3. Recomputed at authoring time by re-running `audit/write-access/compute.mjs`, which prints 56.6% overall and 68.8% for the money-moving tier. Per platform: 73.7%, 70.8%, 61.7%, 45.7%, 34.8%, 15.4%. A CORRECTION TO THE FIRST VERSION OF THIS NOTE, MADE 2026-09-05. It reported a discrepancy between the report prose on one side and `matrix.json` and the script on the other. That was wrong, and the direction matters. `compute.mjs` never reads `matrix.json`; it carries its own hardcoded capability lists, so the two are independent encodings of one corpus and they drifted apart after publication. Ten rows in `matrix.json` were edited by a later contract-fix sprint while the script kept the gates the audit scored. Eight of the ten record why. Two, both Klaviyo, were changed with no annotation at all, and are now stamped as such rather than left silent. Restoring all ten to their audit gates, and splitting one Shopify capability that a later edit had merged into another, reproduces every published figure from the matrix alone: 130 capabilities, total weight 244, 0.0% strict, 56.6% plausible, 45.2% maturity, and the gate histogram already recorded in the file. The matrix now carries both states, each changed row stamped with its audit gate and the sprint that moved it, so a reader can check the July figure and see what has moved since. THE POST-FIX FIGURES ARE THE MORE INTERESTING HALF. Plausible after the sprint is 56.6%, unchanged from the audit, because the sprint's only gain came from promoting one Klaviyo capability and that promotion has been withdrawn on the row's own evidence. Maturity falls from 45.2% to 44.0%, because six Shopify capabilities were correctly demoted from G5 to G4 by the same review. The sprint made the measurement stricter and left the headline where it was. Public copy: artifact bundle, note 3. The script is published at compute.mjs and reproduces the audit figures from a clean directory. Last re-run 2026-09-30; its overall line reads `n=130 totalWeight=244 Strict=0.0% Plausible(G4+)=56.6% Maturity=45.2%`.
4. `audit/write-access/REPORT-ADDENDUM.md`, section 4(b), the enforcement-sensitivity recompute. A second discrepancy, recorded rather than smoothed over: the addendum's baseline row reads 57.0% where `compute.mjs` yields 56.6%. The enforcement figure is quoted as published; the difference does not affect the conclusion, and neither figure is the point of the comparison. Public copy: artifact bundle, note 4.
5. `audit/write-access/REPORT.md`, section 3. Three independent facts each cap the strict score at zero: the live-verification harness had never been run, every ad platform ships with the vendor call suppressed by default, and no production telemetry surfaced a single row carrying a vendor success. Public copy: artifact bundle, note 5.
6. `audit/write-program/SPRINT-4.md` and `SPRINT-5.md`, with the score movement reproduced at authoring time by re-running `compute-sprint4.mjs` and `compute-sprint5.mjs`. The reclassification that lowered the figure is commit `cc0cd5b84`. Public copy: artifact bundle, note 6. Both scripts are published: compute-sprint4.mjs and compute-sprint5.mjs. Last re-run 2026-09-30; their overall lines read `G4+=95.9%` and `G4+=86.1%`.
7. The distinction is enforced in the type system at `lib/ads/writers/types.ts`, where the audit-facing execution mode is deliberately wider than the mode the environment gate can resolve to. Public copy: artifact bundle, note 7.
8. `audit/write-program/SPRINT-5.md`, Part A.1. The two verbs enable and disable a catalogue sync flag that a scheduled job later reads. Public copy: artifact bundle, note 8.
9. `lib/actions/reversibility.ts`. The explicit-confirmation requirement defaults to the value of the reversibility class, so a verb graded none inherits the gate rather than having to opt into it. Public copy: artifact bundle, note 9.
10. `audit/write-program/SPRINT-6.md`, Part A. Ten duplicate-suppression implementations across six platforms guarded 66 verbs, all filtering on completed attempts only. The residual window between marking an attempt in flight and issuing the call remains open and fails in the refusing direction. Public copy: artifact bundle, note 10.
11. `audit/write-program/SPRINT-2E.md`, Task 3, for the grading rule and the three rows it leaves null. The five-row table size is recorded in `docs/audits/live-verification/UNVERIFIED_INVENTORY.md` as of the 2026-07-08 migration verification. The pre-state value that reads as real is recorded in the same sprint document, Task 2. Public copy: artifact bundle, note 11.
12. Commits `28608fa16` and `327a77a6d`; `audit/write-program/SPRINT-2.md`. Corrected paths: send now creates a top-level campaign send job; scheduling first patches a send strategy onto the campaign and then creates the job; cancellation patches the job. Vendor reference: developers.klaviyo.com, operations create_campaign_send_job and cancel_campaign_send. Public copy: artifact bundle, note 12.
13. `audit/write-program/SPRINT-2C.md`, Task 4, which classified each writer test file as contract-asserting or implementation-mirroring and recorded which ones were structurally incapable of catching an invented endpoint. Public copy: artifact bundle, note 13.
14. Commits `b12a5e197` and `bbefcb90e`; `audit/write-program/SPRINT-3.md` Task 1 and `SPRINT-4.md` section D2. Each deliberate change was reverted and the empty diff confirmed before the result was recorded. Public copy: artifact bundle, note 14.

15. `audit/write-program/SPRINT-5.md`, Part B. The renamed fields are on the campaign resource, replacing field 63 with field 104 and field 64 with field 105. The re-verification totals are drawn from the client library's own descriptor rather than from the vendor's prose reference. Public copy: artifact bundle, note 15.
16. Commits `d5bfda266`, `45017a82e` and `acaaab825`; `audit/write-program/SPRINT-1.md`. The shared helper is `lib/guardrails/assert-write-scope.ts`, wired into four write-preparation chokepoints and failing closed on a read error. Public copy: artifact bundle, note 16.
17. Sentinel and readback defects: `audit/write-program/SPRINT-1.md`, Task 2. Documented numeric scope identifiers: `audit/write-program/SPRINT-5.md`, restated in `SPRINT-6.md` as evidence-gathering rather than a code change, because the mapping cannot be written without observing a real response. Attestation storage: `SPRINT-6.md`, Part D. Public copy: artifact bundle, note 17.
18. `audit/write-program/SPRINT-2E.md`, Task 1. The resolution chain reads the top-level column first, then the metadata field where the callback had actually been writing. Public copy: artifact bundle, note 18.
19. `audit/write-program/SPRINT-1.md`, premise correction. The evaluator is `lib/executor/guardrailEvaluator.ts`, called on every path. Public copy: artifact bundle, note 19.
20. `audit/write-access/REPORT-ADDENDUM.md`, section 1, which states the error and its class explicitly: asserting from what was in front of the auditor rather than checking the authoritative source. Public copy: artifact bundle, note 20.
21. `audit/write-program/SPRINT-6.md`, final arm-readiness table. Every row is a human action; none requires a code change. Public copy: artifact bundle, note 21.
22. Production queried on 2026-10-12 through a database role with select access only (on 2026-09-27 for revisions 1.2 and 1.3). Every count runs over rows on brands recorded as real: `SELECT a.* FROM "ActionAudit" a JOIN "Brand" b ON b.id = a."brandId" WHERE b."dataOrigin" <> 'synthetic'`. It filters on the brand's origin rather than on dates because our demo seeder wrote audit rows dated before they existed until 12 October 2026 (Section 9). Queries on those rows: row count; rows with `executionMode = 'live'` grouped by platform, verb and status, with first and latest `createdAt`, `vendorStatusCode`, `vendorOk`, `principalType`, `postState.source` and the reversal link `parentAuditId`; refusal and failure rows by `errorMessage`; all 23 live rows being on one brand, which is our own; and `metadata.authorisation` on the live rows written since 2026-10-02. `Brand.liveWritesAllowed` grouped by `dataOrigin`: one real brand true, fifteen real brands false. `Connection` rows for that account: a Meta connection with no brand attached, status active, created 2026-06-23. `AcTransaction` on brands recorded as real: nine transactions, all on that brand, created 2026-09-13 and 2026-09-14. The unattended row's user field begins `divergence:goal_pacing`; the fix that stamps a principal type for that tier is recorded in `lib/actions/principalType.ts`. The allowlist commits are `ec4c8c4b7` and `9322db95f`; the dispatcher change is `9cce955e1`; the verb count is the number of entries in `PLATFORM_BY_VERB` in `lib/actions/verb-platform.ts`, and the lane count the number of distinct platforms it maps to. The authorisation record and the rollbacks through the dispatcher are commit `9d03817e4`. Not published in the artifact bundle, which describes the July audit.
23. Production read on 2026-09-27 and again on 2026-10-12: the rollback row whose `parentAuditId` is the Shopify product status change has `executionMode` null, `vendorStatusCode` null and a stored response with no vendor call trace.
24. The grader is `gradeActionAuditRow` in `lib/actions/audit/write-record-grade.ts`, introduced in commit `a6bd78643`. On 2026-09-27 it was run, as it stood at revision 1.2, over every production row with a live execution mode, every Shopify row, and every row with a reversal chained to it, passing each row's own fields and its chained reversal rows. On 2026-10-12 it was run, as corrected in revision 1.3 and unchanged since, over every row on a brand recorded as real. It returns Tier 4, 5 or 6; the tiers below 4 are questions about code and it does not evaluate them.
25. Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., and Neubig, G. (2023). WebArena: A Realistic Web Environment for Building Autonomous Agents. arXiv:2307.13854. The abstract states that the best GPT-4-based agent reaches an end-to-end task success rate of 14.41%, against 78.24% for humans. Retrieved 2026-08-05.
26. Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. (2023). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770. The benchmark draws 2,294 problems from real issues and pull requests across 12 Python repositories, and the abstract reports that the best-performing model resolves 1.96% of them. Retrieved 2026-08-05.
27. Mialon, G., Fourrier, C., Swift, C., Wolf, T., LeCun, Y., and Scialom, T. (2023). GAIA: a benchmark for General AI Assistants. arXiv:2311.12983. The abstract reports 92% for human respondents against 15% for GPT-4 with plugins. Retrieved 2026-08-05.
28. Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. (2024). tau-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. arXiv:2406.12045. The abstract reports that state-of-the-art function-calling agents succeed on under half of the tasks, and that the pass-to-the-eighth consistency measure falls below a quarter in the retail domain. Retrieved 2026-08-05.
29. Spracklen, J., Wijewickrama, R., Sakib, A. H. M. N., Maiti, A., Viswanath, B., and Jadliwala, M. (2024). We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs. arXiv:2406.10279. Retrieved 2026-08-05. The figures quoted here are the abstract's own: 576,000 code samples in two languages across 16 models, 205,474 unique hallucinated package names, and an average hallucination rate of at least 5.2% for commercial models and 21.7% for open-source models. We cite the preprint because that is the version we retrieved and verified; we make no claim here about its publication venue.

30. The bundle is at cresva.ai/research/artifacts/crv-2026-001, with the policy in README.md, and an archive at crv-2026-001-artifacts.tar.gz, and this paper as crv-2026-001.pdf. Before publication the three scripts were extracted to a directory outside the repository and run with no other inputs; their output matched the figures printed here, which is what establishes that redaction changed no result.

31. Production read on 2026-09-27 through a database role with select access only: every `ActionAudit` row with its brand's `dataOrigin`, its reversal rows by `parentAuditId`, and the fields the grader reads. The 81 are the 88 rows less the one on the synthetic demo brand and the six with status `refused`. Each row was graded twice from the same read, by the grader as of revision 1.2 (commit `a6bd78643` and its later fixes) and by `gradeActionAuditRow` as corrected in revision 1.3, and the results were tallied by tier, by post-state provenance, by execution mode and status, and by platform and verb for the capability count. Before: Tier 5, 5 rows; Tier 4, 76. After: Tier 5, 7 rows (4 read back, 3 computed); Tier 4, 74 (10 live; 34 expired, 15 rejected, 9 failed, 6 not live). Tier 6, zero in both. Re-run on 2026-10-12 over the 84 rows the home page counts on brands recorded as real (the 90 such rows less six refusals): Tier 5, the same 7 rows (4 read back, 3 computed); Tier 4, 77 (16 live; 31 expired, 17 rejected, 9 failed, 4 not live); Tier 6, zero.

References

[1] Meta, Marketing API reference: ads pixel. Retrieved 2026-07-30 during the contract-verification pass. https://developers.facebook.com/docs/marketing-api/reference/ads-pixel/

[2] Meta, Conversions API: using the API. Retrieved 2026-07-30. https://developers.facebook.com/docs/marketing-api/conversions-api/using-the-api/

[3] Klaviyo, API reference: create campaign send job. The operation that replaced the invented send sub-resource. https://developers.klaviyo.com/en/reference/create_campaign_send_job

[4] Klaviyo, API reference: cancel campaign send. https://developers.klaviyo.com/en/reference/cancel_campaign_send

[5] Google, Google Ads API reference: CampaignService, version 24. The message whose date fields were renamed. https://developers.google.com/google-ads/api/reference/rpc/v24/CampaignService

[6] Google, Google Ads API guide: create campaigns. The source for the date-time format used at the writer boundary. https://developers.google.com/google-ads/api/docs/campaigns/create-campaigns

[7] Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., and Neubig, G. (2023). WebArena: A Realistic Web Environment for Building Autonomous Agents. arXiv:2307.13854. Retrieved 2026-08-05. https://arxiv.org/abs/2307.13854

[8] Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. (2023). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770. Retrieved 2026-08-05. https://arxiv.org/abs/2310.06770

[9] Mialon, G., Fourrier, C., Swift, C., Wolf, T., LeCun, Y., and Scialom, T. (2023). GAIA: a benchmark for General AI Assistants. arXiv:2311.12983. Retrieved 2026-08-05. https://arxiv.org/abs/2311.12983

[10] Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. (2024). tau-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. arXiv:2406.12045. Retrieved 2026-08-05. https://arxiv.org/abs/2406.12045

[11] Spracklen, J., Wijewickrama, R., Sakib, A. H. M. N., Maiti, A., Viswanath, B., and Jadliwala, M. (2024). We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs. arXiv:2406.10279. Retrieved 2026-08-05. https://arxiv.org/abs/2406.10279

[12] Cresva. CRV-2026-001 artifact bundle: the capability matrix, the three deterministic score scripts, the redacted excerpts behind every footnote citing an internal document, and a README stating the redaction policy. Published alongside this paper; the scripts run offline with no inputs beyond the bundle. https://cresva.ai/research/artifacts/crv-2026-001/

[13] Cresva, write-access truth audit. Internal artifact, repository path audit/write-access/, including REPORT.md, REPORT-ADDENDUM.md, matrix.json and compute.mjs. The published subset is in the artifact bundle above.

[14] Cresva, write program sprint records. Internal artifacts, repository path audit/write-program/, SPRINT-1 through SPRINT-6 with the two deterministic rescore scripts. The published subset is in the artifact bundle above.

How to cite

```
@techreport{raghav2026proofofwrite,
  title       = {{Proof of Write: An Evidentiary Standard for Agent Action Capability}},
  author      = {Shubham Raghav},
  institution = {Cresva},
  year        = {2026},
  number      = {CRV-2026-001},
  version     = {1.4},
  url         = {https://cresva.ai/research/proof-of-write}
}
```

Canonical version: https://cresva.ai/research/proof-of-write
Artifact bundle: https://cresva.ai/research/artifacts/crv-2026-001
CRV-2026-001 version 1.4. Cite as raghav2026proofofwrite.