WHITE PAPER

CHANGE CONTROL

Preventing Unapproved Changes & Compliance Claims

How to tell whether an automated system can make a change — or a compliance claim — that nobody approved, and how to verify the controls that are supposed to stop it.

For compliance, vendor-risk, procurement, and process owners · Updated August 2026

The short version

The single failure mode a regulated buyer has to design against is “nobody approved this.” It shows up two ways: the system makes a **change** no one authorized, or it makes a **claim** no one substantiated — and both have the same root cause.

An action or an assertion reached the outside world with no named human decision behind it and no record that survives to prove the decision was, or wasn’t, made. Automation sharpens this: a system that chains steps at machine speed propagates an unapproved change before anyone notices, and one that writes fluent, confident language makes an unsubstantiated claim read more credibly than a human’s error would. The evaluator’s real question is therefore not “is the AI accurate?” — accuracy is a quality problem — but “can this system take a consequential action, or emit a compliance claim, with no human in the decision and no durable trace?” That is a control problem, and unlike accuracy, controls can be verified. This paper names the failure mode precisely, sets out the six controls that stop it, shows which control

catches which failure, and hands you a scorecard and a four-week plan to confirm the controls actually operate rather than merely exist on a diagram.

1. What “nobody approved this” actually means

Two hazards share one root cause. **Hazard A — the unapproved change:** the system edits a record, deletes a file, moves money, sends an external email, or alters a customer’s status with no one having authorized it. **Hazard B — the unsubstantiated claim:** the system states in an output, a filing, or a customer-facing message that something is compliant, complete, reconciled, or approved when no human established a reasonable basis for saying so. In both, something crossed into the world without a named decision behind it and without a record that proves whether the decision was made.

The load-bearing distinction is **reversible versus irreversible**. A reversible action needs a guaranteed way back. An irreversible action needs a gate *before* it happens, because there is no way back afterward — a rollback plan cannot help where nothing can be rolled back. Any vendor answer that blurs the two — treating “we can restore from backup” as if it covered a wire transfer or an external send — is a red flag, because it means the vendor has not actually classified their own actions by consequence.

Decision table: which control is load-bearing for each action

Action	Reversible?	Load-bearing control	What “safe” looks like
Edit a record of note	Usually	A tested undo + a record of the before-state	The prior value is retrievable and the change is logged with who and when
Delete data or files	Rarely	Confirm before commit + a retention hold	A named person confirms; deletion is soft/recoverable for a defined window before it is final
Send an external email	No	Confirm before commit	The message halts for a named human before it

Action	Reversible?	Load-bearing control	What “safe” looks like
			leaves; there is no “unsend” to rely on
Post to a ledger / move money	No	Confirm before commit + no self-approval	A person authorizes the specific amount; the system cannot mark the authorization as given on its own
Publish a compliance statement	No, in practice	Substantiation + no self-approval	A person converts a finding into the asserted claim, tied to the evidence behind it
Grant or change access	Technically, not in effect	Confirm before commit + least access	A person approves the grant; whatever the change touched during the window is reviewable

“Reversible” is about consequence, not mechanics. An email can be deleted from a mailbox, but the recipient has already read it — so for control purposes it is irreversible. Judge each action by whether the *effect* can be undone, not whether a database row can.

2. No claim without a reasonable basis

Compliance claims deserve their own principle, because a fluent system produces convincing wording effortlessly and wording is easy to mistake for substance. The principle, stated plainly: before any objective claim is made, whoever makes it must already hold a reasonable basis for it — the supporting evidence has to exist *first*, not be assembled after someone challenges the claim. This is long-settled in advertising and consumer-protection practice, and it applies identically whether a person or an automated system produced the sentence. It is a principle to apply, not legal advice; confirm how it bears on your obligations with your own counsel.

“Reasonable basis” scales with the stakes. The more damage a false claim would do — to safety, to finances, to a regulatory or contractual position — the more support is expected before it goes out. A throwaway internal note and a customer-facing attestation are not held to the same bar.

> **The distinction that matters for automation**
>
> An AI that generates the *sentence* “this is compliant / reconciled / complete / approved” does not thereby create the *basis* for it. The safe pattern is that automated systems surface findings and drafts — “here is what I found, here is the evidence” — and a named human converts a finding into an asserted claim. The dangerous pattern is that the system’s fluent summary *is* the claim, published with no one having checked the basis.

This gives you a sharp test. Ask the vendor to trace one specific compliance-flavored output back to the evidence and the person who authorized it as an assertion. If the trace ends at “the model wrote it,” there was no reasonable basis — which means it is an unsubstantiated claim by construction, not by accident.

3. The six controls that prevent it

Underneath the two hazards are six controls that separate a platform that is genuinely safe from one that merely intends to be careful. Treat these as principles to look for in any vendor, not as one product’s feature list — and ask the vendor to *demonstrate* each on a real action rather than describe it.

Undo by design

Every reversible action is built together with a defined, tested way to reverse it, so an unwind path always exists and has actually been exercised — not improvised after something goes wrong.

Make them show it: pick one action and ask them to reverse it live, and to show evidence the undo has been run before.

Confirm before commit

Consequential or irreversible actions do not run unattended; they halt into a durable waiting state and require a named human to confirm before anything commits.

Make them show it: ask whether the pause is a real stop in execution that survives a restart, or just a notification the system can blow past.

No self-approval

The system must be unable to fabricate its own sign-off. A genuine human act is required to approve, so the automation can never invent, infer, or auto-fill the approval it needs to proceed.

Make them show it: ask how the platform proves an approval came from a person — and whether the model can ever decide the approval condition is met on its own.

A record that can't be rewritten

Every action and every reversal is written to a record no one — including the vendor's own staff — can edit or delete after the fact, capturing the approval, not just the action.

Make them show it: ask who *can* alter the log, and how you could verify its integrity yourself without trusting their word.

Stops when unsure

When a permission, a check, or a version doesn't line up, the step stops and escalates rather than pushing through. The default on uncertainty is to do nothing irreversible.

Make them show it: ask what happens on a mismatch. The safe answer is "it halts," not "it retries" or "it proceeds with a default."

Least access per task

Each function runs with only the access it needs, on separate credentials, so a fault or leak in one workload cannot reach another — a compromised drafting function cannot move money or grant access.

Make them show it: ask whether one compromised component exposes everything, or only its own slice.

4. Where each control does its work

The six controls are not interchangeable. Each stops a specific way "nobody approved this" happens, which is exactly why you need all of them — and why spotting the one a vendor is missing tells you which failure they are exposed to. Read this table as a map from the failure you fear to the control that catches it.

Failure mode	Control(s) that stop it	What its absence looks like in production
Unapproved irreversible change	Confirm before commit + no self-approval	A one-way action — a send, a payment, a deletion — happens and is only discovered afterward
Unapproved reversible change	Undo by design + a record that can't be rewritten	A record is altered; nobody can say cleanly what the prior state was or who changed it
Approval faked or inferred	No self-approval	The system "decided" the approval condition was met and proceeded as if a human had signed off
Unsubstantiated compliance claim	Substantiation pattern + a record that can't be rewritten	A settled-sounding attestation goes out that no one can trace to evidence or to a person
Silent partial failure	Stops when unsure	A step half-completes, swallows the error, and reports success
Blast radius when a fault occurs	Least access per task	One broken workload becomes a tenant-wide incident because everything shared one credential
Dispute over what happened	A record that can't be rewritten	Two parties disagree and there is no independently trustworthy account to settle it

Note the record that can't be rewritten appears twice and underwrites the rest: every other control is only as trustworthy as the record that proves it operated. The audit trail is what makes the other five auditable at all.

5. How a buyer verifies the control actually works

This is where most evaluations fail. A control that is *designed* is not a control that *operates*. A policy document, an architecture diagram, or a ticked box in a questionnaire is a design assertion. Evidence of operation is a log entry you can retrieve, a stop you watched happen, an undo the vendor produced on demand, an

independent attestation covering a real period. Push every claim from the first category into the second.

Ask for the negative test, not the happy path

"Show me a consequential action being blocked, paused, and escalated" proves a gate far better than "show me the system working." A control you can only ever watch succeed is a control you have not actually tested. Insist on seeing it refuse.

Demand a single end-to-end trace

Pick one real consequential action and make the vendor walk it start to finish: what initiated it, which permission scope it ran under, where it paused, who confirmed it, what the undo would be, and the untouchable record of all of the above. Vagueness at any one step is the finding — note which step, because it names the missing control.

> **Tamper-evident, not tamper-proof**
>
> Mature audit systems don't claim a log can never be touched; they claim any tampering is *detectable* and independently checkable. A common approach is to seal each entry against the one before it, so any later edit breaks the chain and anyone holding the log can see it — ideally checked against an external reference point. Ask specifically: *how could I, the customer, verify the log's integrity without taking your word for it?* A vendor who has only "our admins are careful" has a logging feature, not an audit control.

Prefer independent, period-covering evidence — and watch the scope

A report describing how controls operated over months beats a point-in-time snapshot; a live demonstration beats a slide; a scoped test in your own environment beats both. Then read the scope carefully. The most common trap is evidence that quietly covers the wrong thing — an attestation that includes the marketing site but not the action-taking runtime, or "AI safety" language that never touches change control. Confirm the evidence covers the exact component that can take the action, for a period that has not expired.

6. The vendor evaluation scorecard

This is the instrument to circulate. Score each row on *evidence of operation*, not on the answer alone: **0** if the answer only restates policy, **1** if the control is documented, **2** if it is demonstrated with an artifact you can keep. Weight the rows to what your workflows actually do — a read-only tool can carry a low score on "undo"; an unattended one cannot. Fill it in a working session with the vendor present, because the strong answers require them to produce artifacts live rather than promise them later.

Control & question	Weak answer (0–1)	Strong answer (2)
Undo — is there a tested reversal for this action?	"We can restore from backup"	A per-action undo shown live, with evidence it has been run before
Undo — does the record capture the prior state?	"The change is in the history somewhere"	The before-value is retrievable and tied to who changed it
Confirm — do irreversible actions halt for a person?	"It flags important ones"	A real execution stop into a durable pending state, demonstrated
Confirm — who defines "important," and is it enforced?	The model decides case by case	A named, enforced list of gated action classes, not model-judged
No self-approval — can the system sign off for itself?	Unclear, or "the workflow approves it"	No — and they can show approval requires a real human act
No self-approval — is a human in the decision, not just notified?	The person is told after it happens	The person acts *before* commit; a

Control & question	Weak answer (0–1)	Strong answer (2)
		receipt is not approval
Record — is the log append-only and un-editable?	"We log everything"; admins can "tidy" it	Append-only; nobody, vendor included, can alter entries
Record — can you verify integrity without trusting them?	"Trust us, it's secure"	Tamper-evidence you can independently check
Stops when unsure — what happens on a mismatch?	"It retries" / "proceeds with a default"	It halts and escalates — shown on a negative test
Least access — does one credential run everything?	"One service account, it's trusted"	Per-workload separation; one fault can't reach another
Substantiation — can a compliance claim trace to evidence?	"The model is accurate / trained for compliance"	A live trace from the claim to the evidence and the person
Evidence — is there independent, in-scope, current proof?	A snapshot, or a report covering the wrong component	Period-covering evidence over the action-taking runtime

Disqualifiers — end the evaluation regardless of total

- "The system can approve its own actions when it's confident."
- "Admins can edit the log to make corrections."
- "Irreversible actions run unattended for efficiency."
- "One credential runs everything."

Scoring bands. Sum the weighted scores, but treat the total as secondary to the high-weight rows. **Proceed** when every High row scores 2 and no disqualifier appears. **Proceed with conditions** when Highs are strong but one or more Mediums are weak — tie the conditions to the specific weak controls and re-verify them before go-live. **Do not proceed** on any High-weight 0, or any disqualifier, whatever the total: a vendor can have a strong sum and still be undeployable because of a single high-weight gap.

> **Keep the completed sheet as the audit file**
>
> Add an "evidence attached" note to each row — the artifact you were shown, and its date. The finished scorecard then doubles as the record of *why* this vendor was accepted, which is itself the substantiation of the procurement decision. Put every artifact under NDA and set a date to re-request it; evidence expires.

7. Red flags and evaluator mistakes

Two lists. The first is what vendors say that should slow you down; the second is how careful buyers still get it wrong.

> **Vendor answers that should slow or stop a deal**
>
> - **"The model is very accurate / fine-tuned for compliance."** Accuracy is not a control. An accurate system with no gate still takes unapproved actions — just fewer obviously-wrong ones, which is more dangerous because scrutiny drops.
> - **"It always asks before doing anything important."** Ask who defines "important," whether it is enforced in execution or merely advisory, and whether the model decides. A model-judged gate can be reasoned around; a hard execution stop cannot.
> - **"Everything is logged."** Logging is not audit. Ask whether it is append-only, whether an admin can alter it, whether integrity is independently checkable, and whether it captures the *approval*, not just the action.
> - **"A human is always in the loop."** Verify the human is in the *decision*, not merely notified after. Post-hoc notice of an irreversible action is a receipt, not approval.

Mistakes evaluators make

- Evaluating the chat or answer quality and skipping the action runtime. The demo that impresses is rarely the part that can cause harm — insist on evaluating the component that writes, sends, deletes, and asserts.
- Accepting a single point-in-time attestation as proof of ongoing operation, or one whose scope quietly excludes the automated action layer.
- Treating "reversible" as a free pass without confirming the reversal is tested and the record identifies what to reverse. An undo that has never been run is a claim, not a control.

8. A four-week evaluation plan you can run

This turns the paper into a sequence a procurement or compliance team can execute. The order matters: you cannot evaluate controls until you know which actions and claims are actually in play in *your* environment.

Week / step	Task	Owner	Pass / fail	Evidence attached
Week 1 — scope	List every consequential action and every compliance-flavored claim the system could produce in your environment	Process owner	—	—
Week 2 — scorecard	Issue the 12-question instrument; hold a working session where the vendor demonstrates a real stop, a real undo, and a real end-to-end trace	Vendor risk	—	—
Week 3 — negative tests	In a sandbox or scoped pilot, try to make the system take an unapproved irreversible action — confirm it stops. Try to make it emit a settled compliance	Security	—	—

Week / step	Task	Owner	Pass / fail	Evidence attached
	claim — confirm it surfaces a finding for a human instead			
Week 4 — verify the record	Confirm every test left an untouchable, retrievable audit entry; check integrity is verifiable without trusting the vendor; confirm any independent attestation covers the action-taking component for a real period	Compliance	—	—
Decision	Score, attach the evidence, record the conditions; the completed file is both the go/no-go and the substantiation for the decision itself	Sign-off owner	—	—
Re-verify	Set a schedule to re-run the negative tests and re-check attestation scope at renewal — controls drift; verified once is not verified forever	Vendor risk	—	—

How to run the evaluation

Scope the blast radius before the first vendor call. Send the scorecard ahead of a working session so answers arrive with artifacts attached rather than improvised live. In the session, make the vendor *demonstrate* the six controls on a real action — a real stop, a real undo, a real trace — and score on what they showed, not what they claimed. Run the negative tests in a sandbox, not just the happy path. Route the contract and data terms through your own counsel or authorizing official; nothing in

this paper is legal advice. Then keep the evidence under NDA with a date to refresh it — a vendor that passed last year may not pass today.

---

sg1consulting.us · contact@sg1consulting.us