AltSecCON

Nobody Approved This:

Hijacking Approval Authority Across AI Agents and Workflows

What we're covering today

01 **The Approval That Never Happened**

A perfect audit record. A real reviewer name. A transfer that nobody approved.

02 **The Hidden Trap**

Why every stored approval is a loaded gun and why "they already have DB access" is the wrong objection.

03 **The Five Attacks**

Forge, Rebind, Replay, Suppress, Launder - defined, diagrammed, and demonstrated.

04 **The Scale of the Problem**

43 systems tested. 30 exposed. 11 resistant. What separates them.

05 **The Escape Plan**

Bind approval to execution. Three detections for your blue team. One question to take home.

Regulators promised meaningful human oversight.

I checked whether the approval path actually proves human authority at resume time.

01

Oversight promise

High-risk AI should allow review, approval, or override before impact.

02

Agentic gate

The agent pauses. A human reviews. The workflow resumes.

03

The missing test

At resume, does it verify the human decision or just read approved=true?

What happens between pause and resume?

THE QUESTION THAT STARTED EVERYTHING

> *If a regulator audits your agentic workflow tomorrow and asks who approved the last 100 transfers - **does your framework actually know?***

Almost every framework I tested answers "yes, we know who approved it" - and is wrong. The audit record exists. The approval does not.

Why does this matter at all?

Because the systems that enterprises are betting on to "keep humans in control"
are the same systems that will execute without human control - and leave a paper trail saying otherwise.

WHAT A FORGED APPROVAL CAN REACH

The terminal effects are real.

CLOUD IAM

Assumable administrator role

- Gitea approval flows into Atlantis
- Terraform apply creates admin role
- Runs under deployment service account

GITOPS

Cluster-admin binding

- Tracked branch looks approved
- Reconciliation applies change
- Attacker service account gets cluster-admin

IDENTITY

Admin group membership

- Governance case outcome says approved
- Connector writes to live directory
- Attacker becomes IdP administrator

FINANCE

System-of-record transaction

- Human-task gate releases workflow
- Transfer commits with service credential
- Audit names a human who never acted

A forged approval is not cosmetic. It reaches cloud, identity, finance, and production control planes.

01

The Approval That Never Happened

A perfect audit record. A real reviewer name. A transfer that nobody approved.

PRODUCTION AUDIT LOG | TUESDAY 03:09

Every field an auditor expects. No human ever saw it.

```
AUDIT EVENT  ->  wire_transfer
action     : release_wire_transfer
amount     : USD 100,000.00
decision   : approved
approver   : compliance officer
timestamp  : 2026-06-03T14:03:11Z
gate       : human_approval
status     : PASSED HUMAN REVIEW
```

The attacker did not call the payment API. Did not touch the model. Did not hold the approver credential. One write into the state store.

So who approved it?

The record names a real reviewer. The timestamp is real. The action executed correctly through the framework's own documented resume path. The funds moved. The gate status says **PASSED**.

But no human ever saw this transfer. No approval interface was opened. No credential was used. The framework cannot tell you who approved it - **because nobody did**.

02

The Hidden Trap

Why every stored approval is a loaded gun - and why "they already have DB access" is the wrong objection.

THE CORE INSIGHT

Two jobs collapsed into one stored value.

JOB 01

Record evidence

Store what happened: approver, decision, action, and timestamp.

JOB 02

Authorize execution

Decide whether the framework may release the protected action.

FAILURE

Store becomes authority

On resume, whatever is in state becomes the ruling.

Resistant design → **Storage can remember the decision. It cannot authorize the decision without independent verification.**

Core rule: recording a decision and verifying a decision are different trust operations.

THE QUESTION EVERYBODY ASKS

“But the attacker already has DB access.”

Why is this a vulnerability?

WHAT THE STATE WRITER HAS

Narrow maintenance capability

- Can write workflow state
- Cannot call payment API
- Cannot run terraform apply
- Cannot read secrets
- Cannot impersonate approver.

WHAT THE FRAMEWORK HAS

Protected service authority

- Holds the service credential
- Can provision infrastructure
- Can grant IAM roles
- Can write system of record
- Runs the action after resume

This is escalation of effect, not escalation of access. The framework turns a narrow write into a human-gated effect.

THE VULNERABILITY CLASS

Approval-Provenance Failure

The violation requires three conditions to hold at the same time. Break any one of them and the forge stops working.

1

The framework persists pending state at the gate

This is necessary and legitimate. Approval may take hours. The run must survive a restart.

2

The resume path reads it as a finalized decision

The stored value stops being a record and becomes the ruling. Whatever is in the store on resume is what happens.

3

Nothing verifies it against a trusted path

No origin check. No binding to the execution context. No proof that the value came from the approval flow for this specific action.

> Mapped to CWE-345 (Insufficient Verification of Data Authenticity), CWE-346 (Origin Validation Error), and CWE-501 (Trust Boundary Violation).

Five ways one missing check is exercised.

These are not five separate bugs. They are five ways the same missing verification that a decision came from the authorized approver and covers the exact action gets exploited.

Forge	Fabricate an approval that never happened
Rebind	Keep a genuine approval but change what it authorizes.
Replay	Use one genuine approval outside its intended run, target, or lifetime
Suppress	Change the rule that decides whether approval is required
Launder	Carry a weak or forged approval across trust boundaries until it becomes stronger authority downstream.

EXPLOIT CLASS 01

Forge: making an approval out of nothing.

No human approves anything. The protected action still executes.

01

Write state

Set approval-shaped value in checkpoint, session state, or execution data.

02

Resume trusts

Framework treats the stored value as the decision of record.

03

Service executes

Funds move, access changes, or infrastructure is provisioned.

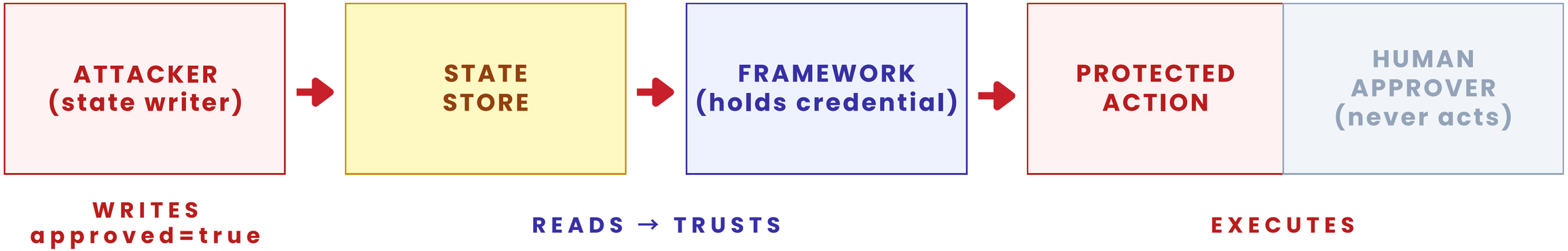
04

Audit looks clean

Native record names a reviewer who never acted.

Attacker needs one thing: write access to the workflow state store. No approver credential. No code change.

The approver is greyed out. The framework doesn't notice.



The framework cannot distinguish a genuine human decision from a state-writer mutation. Both produce the same stored value.

Control. Forge. Record.

STEP 01 · CONTROL

The gate is real

```
$ run --workflow wire_transfer
  amount: USD 100,000
  gate:   human_approval

> reaches gate ...pause
> resume without approval

x BLOCKED · approved=False
  transfer not executed
```

STEP 02 · THE FORGE

One write. Nothing else.

```
# state write only
# no host, prompt, or code
sqlite3 checkpoints.db <<SQL
UPDATE checkpoints SET
  channel_values = json_set(
    '$.approved', 1,
    '$.approver',
    'compliance officer');
> Command(resume) · EXECUTED
```

STEP 03 · THE RECORD

The audit trail

```
AUDIT EVENT
action    : release_wire
amount    : USD 100,000
decision  : approved
approver  : compliance
            officer
timestamp : 14:03:11Z
status    : PASSED
no anomaly · no flag
```

Every call in this sequence is a supported, documented framework operation.

EXPLOIT CLASS 02

Rebind: real approval, wrong thing.

The human approved A. The stored execution changes to B.

BEFORE REVIEW

A is shown and approved

- $5,000 transfer
- Known vendor account
- Real reviewer says yes

→

AFTER REVIEW

State changes A → B

- Amount increases
- Recipient changes
- Target resource swaps

→

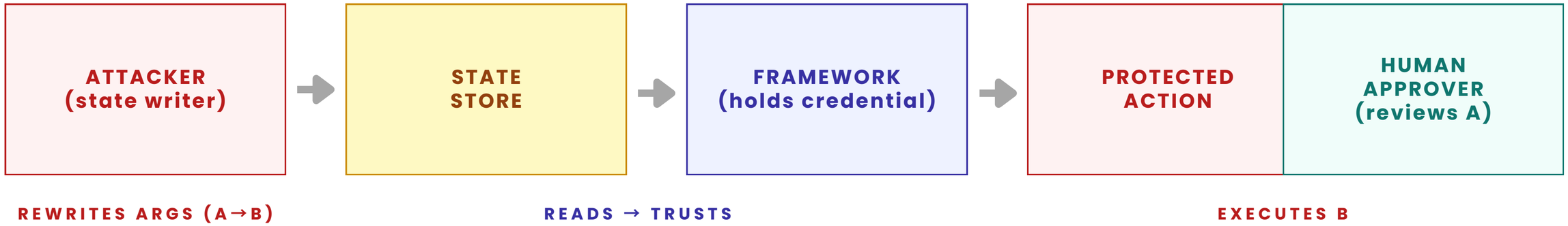
AUDIT RESULT

Reviewer name stays attached

- Same approver
- Same run
- Different action executes

Missing binding: no integrity hash between reviewed context and stored arguments.

The approval is genuine. The arguments change after.



The reviewer's name stays attached to a transaction they never authorized. The native record cannot distinguish what was reviewed from what ran.

Alice approves 5,000. 100,000 executes.

WHAT ALICE REVIEWED

```
event     : approval.granted
run_id    : wf_8831c4
action    : release_wire_transfer
amount    : USD 5,000.00
recipient : acct_4471 (vendor)
decision  : approved
approver  : alice-reviewer
timestamp : 14:03:11Z
```

WHAT EXECUTED

```
event     : approval.granted
run_id    : wf_8831c4
action    : release_wire_transfer
amount    : USD 100,000.00
recipient : acct_9903 (attacker)
decision  : approved
approver  : alice-reviewer
timestamp : 14:03:11Z
```

The approval is cryptographically indistinguishable from a genuine one — because it IS a genuine one. For the wrong thing.

EXPLOIT CLASS 03

Replay: one approval, used forever.

Without run binding and freshness, approval becomes a bearer capability.

01

No run ID

Copied into a different workflow run.

02

No nonce

Same approval can be consumed repeatedly.

03

No expiry

Old approvals keep working later.

04

No audience

Decision crosses resource and service boundaries.

Auditor sees: same reviewer approving many actions. Reality: reviewer approved one.

Replay turns one successful Forge or Rebind into reusable infrastructure.

One approval. Two runs. No binding.

RUN A | MONDAY | LEGITIMATE

```
run_id    : wf_A
amount    : 5,000
approved  : TRUE
by        : alice
```

Alice genuinely approved
this specific transfer.

→ **copy state**

RUN B | TUESDAY | FORKED

```
run_id    : wf_B
amount    : 100,000
approved  : TRUE
by        : alice
```

Alice never saw Run B.
Her name authorizes it anyway.

MISSING

- run_id binding
- single-use nonce
- expiry / freshness
- audience
- argument hash

An approval without run and single-use binding is an unintentionally broad bearer capability.

EXPLOIT CLASS 04

Suppress: delete the requirement for approval.

The attacker does not forge evidence. They prevent evidence from existing.

01

Policy is state

confirmation_policy is persisted and mutable.

02

Rule is downgraded

AlwaysConfirm becomes NeverConfirm.

03

Gate never runs

The action executes without asking anyone.

04

No record exists

Investigator finds nothing to compare.

Fix requires policy integrity: authorized admin path, version binding, audit history, and fail-closed validation.

The gate that never fired.

BEFORE -> CONTROL STATE

AlwaysConfirm

```
// ConversationState/state.json
"confirmation_policy":
  "AlwaysConfirm"

> high-risk action submitted
x BLOCKED · awaiting approval
```

AFTER -> POLICY DOWNGRADED

NeverConfirm

```
// state.json — one write
"confirmation_policy":
  "NeverConfirm"

> high-risk action submitted
> EXECUTED · gate not invoked
```

`approval_record_present = false` `approver = None` `evasion by absence — nothing to alert on`

EXPLOIT CLASS 05

Launder: weak approval becomes strong authority.

Each downstream component behaves correctly. The aggregate does not.

HOP 01

Gitea DB

- INSERT review row
- type = APPROVED
- reviewer_id = 17

→

HOP 02

Gitea API

- PR returns approved
- reviewers = [alice]
- looks legitimate

→

HOP 03

Atlantis

- PullIsApproved(pr) true
- apply requirement met
- normal path continues

→

HOP 04

Terraform

- aws_iam_role.admin
- created / assumable
- admin effect downstream

Check at every hop: correct. | **Provenance verified: zero hops.**

One DB write. Four hops. Cloud admin.

1 · GITEA DB

```
INSERT INTO review
type='APPROVED'
reviewer_id=17
```

▶

2 · GITEA API

```
GET /repos/../pulls/42
→ approved: true
→ reviewers: [alice]
```

▶

3 · ATLANTIS

```
PullIsApproved(pr)
→ true
apply requirement met
```

▶

4 · TERRAFORM

```
aws_iam_role.admin
· created
· assumable
· attached
```

Check at every hop: correct. Provenance verified: at zero hops.

Terminal cloud apply is isolated-lab logic-verified - no live provision performed.

GENERALITY

Three chains. Three trust boundaries crossed.

VCS → IaC → IAM

Code review to cloud administrator

LOGIC-VERIFIED

IGA → Live Directory

Governance case to production admin

REAL WRITE

Agent → Guarded Tool

Run state to external payment

RUNTIME

Each downstream component behaves correctly. The aggregate produces administrator authority nobody signed for.

04

The Scale of the Problem

43 systems tested. 30 exposed. 11 resistant. What separates them.

CORPUS

43 systems. One invariant. One verdict.

26	3	1	11	2
Directly exposed	Conditionally exposed	Transitively exposed	Resistant	No qualifying gate

The corpus spans:

- Agentic AI frameworks (23)
- durable-execution engines (7)
- orchestration and BPMN engines (7)
- CI/CD gates (2)
- code-review systems (2)
- identity governance (1)
- and IaC composition (1)

NEGATIVE CONTROLS

Three systems took the same write and refused it.

Resistant designs keep the approval path and the persistence path as separate trust boundaries. Stored state can exist, but stored state alone is never sufficient authorization.

PydanticAI

pydantic-ai 1.107.0

DeferredToolResults keyed to the pending tool-call ID

- Persisted message history cannot authorize tool execution.
- Require a genuine DeferredToolResults object.
- Bind it to the pending tool-call ID.

SUITE 5/5 PASSED

BeeAI

beeai-framework 0.1.81

Live permission-handler callback

- Approval is resolved by a live handler callback.
- Resume does not trust persisted approval state.
- No persisted approval artifact can be mutated.

SUITE 8/8 PASSED

Mastra

@mastra/core 1.45.0

Live, schema-validated resumeData

- approved=true in snapshots does not authorize execution.
- Resume requires valid, live resumeData.
- resumeData must pass schema validation.

SUITE 3/3 PASSED

AUDIT SURFACE

The record lies. Or the record doesn't exist.

FORGE / REBIND / REPLAY

The record lies

Decision: approved. Approver: alice-reviewer. Timestamp: 14:03:11Z. Status: PASSED HUMAN REVIEW.

- Indistinguishable from a genuine approval on native fields. Operators, auditors, and compliance teams read the log and conclude a human approved it.

SUPPRESS

The record doesn't exist

approval_record_present = false
approver = None

- There is nothing to compare. Nothing to alert on. Nothing to preserve. The gate was suppressed before the record could exist.

Deployments that correlate an external approval-service issuance event can detect this. Those relying on native workflow evidence cannot.

05

The Escape Plan

Bind approval to execution. Three detections for your blue team. One question to take home.

Authenticate the decision, not the store.

The approval authority mints a signed artifact. The workflow verifies it before execution. The store may hold it — but the store never authorizes with it.

```
approval = {
  decision       : approved,
  approver       : human_or_role_id,
  action         : release_wire_transfer,
  args_hash      : sha256(canonical_args),
  run_id         : current_run,
  checkpoint_id  : pending_checkpoint,
  context_hash   : sha256(context_shown),
  policy_version : current_policy,
  audience       : resume_service,
  nonce          : single_use_nonce,
  issued_at, expires_at }

sig = Sign(approval_key, canonicalize(approval))
```

WHAT EACH BINDING DEFEATS

signature	FORGE
args_hash	REBIND
run_id + checkpoint	REPLAY
nonce + expires_at	REPLAY
context_hash	REBIND
policy_version	SUPPRESS
audience	LAUNDER

TWO MORE REQUIREMENTS THE ARTIFACT ALONE DOES NOT SOLVE

Policy integrity. Downstream provenance.

DEFEATS SUPPRESS

Policy integrity

- Policy changes require a distinct authorized principal.
- Every policy version must be signed and logged.
- State restore must reject unauthenticated policy state.
- Invalid policy state must enforce strictest settings.
- Fail closed by default.

DEFEATS LAUNDER

Downstream provenance

- Verify signed attestations, not bare approval responses.
- Bind attestations to the exact reviewed object.
- Include reviewer identity, decision, and freshness.
- Reject stale or unbound attestations.
- Follow supply-chain provenance principles like in-toto.

SAME STORE · SAME WRITE · DIFFERENT VERDICT

Repeat the write. Watch it fail.

`Unsigned SQLite write`	REJECTED	*missing signature*
`Signed artifact, args_hash altered`	REJECTED	*argument mismatch*
`Signed artifact, wrong run_id`	REJECTED	*run binding failed*
`Consumed nonce (replay)`	REJECTED	*single-use enforced*
`Policy downgrade attempt`	REJECTED	*unauthenticated policy - strict fallback*
`Legitimate signed approval`	ACCEPTED	*all bindings match - executes once*

HMAC-SHA-256 verify: 1.665 µs median | Ed25519 verify: 103.360 µs median | 469-byte payload → local microbenchmark

FOR THE BLUE TEAM IN THE ROOM

Three things to hunt for.

01

Protected action completed without a verified approval-artifact identifier

02

Confirmation-policy change from outside the admin path

03

Approval record written by an operational principal with no matching issuance event

COORDINATED DISCLOSURE · SNAPSHOT AT SUBMISSION

Where every affected system stands.

FlowiseAI

GHSA-2hjv-g5fh-6w4j · Moderate · fixed and published

VENDOR CONFIRMED

DOC RESPONSE

Microsoft (Semantic Kernel, Agent Framework)
OpenAI Agents SDK
Haystack

ACKNOWLEDGED

Google ADK (VRP)
n8n

REVIEW PENDING

Agno
CrewAI
LangGraph

NOTIFIED

Airflow, Conductor, Flowable, Kestra, Kogito, DBOS, Dapr, Gitea, midPoint, Strands, OpenHands, and others

TAKE THIS BACK TO WORK

Two questions. Ask them of any workflow.

QUESTION 01

*When the workflow resumes, does it **trust a stored approval value**, or does it **verify that the decision came from the authorized approval path** for this exact action and execution?*

QUESTION 02

*Can a principal **outside the approval or policy-administration path** change the setting that decides whether the gate fires at all?*

FINAL RULE

*A state writer may **store** an approval record, but must never be able to **create approval authority**.*

AltSecCON

Questions.