

# From Policy to Commit:

## Execution-Boundary Control for Governed AI Systems

---

Ricky Dean Jones

AlvianTech / TrinityOS Governance Series

Working Paper v0.1

2 May 2026

Contact: ricky.mcjones@gmail.com

Repository: [github.com/AlvianTech/commit-gate-core](https://github.com/AlvianTech/commit-gate-core)

### ABSTRACT

AI governance frameworks describe intent. They define policies, assign responsibilities, establish monitoring pipelines, and mandate audit trails. Yet none of these mechanisms answer the question that matters most in an agentic system: *was the invalid action physically unreachable?*

This paper argues that governance is only enforceable where a runtime commit boundary exists: a structural control point through which every consequence-producing action must pass before it becomes effect.

We introduce the **CommitGate model**: an execution-boundary control architecture in which every consequential mutation must present a signed DecisionRecord, pass scope and expiry checks, pass nonce-based replay protection, and be allowed or refused by an authority gate before any downstream state changes.

Denials are not silent. Every refused action generates a receipt. The audit trail is append-only and replayable.

This paper does not claim to solve AI safety broadly. It claims to give the standards room one missing object: a precisely defined control point where a governed AI-enabled system can still be stopped before consequence.

**Keywords:** AI governance, runtime control, commit boundary, refusal semantics, agentic AI, execution boundary, audit evidence, EU AI Act, ISO/IEC 42001, NIST AI RMF.

---

### Claim Boundary

This paper does not claim that CommitGate proves AI safety, model correctness, organisational compliance, or trustworthy judgement. It claims only this: for a correctly implemented gate, a consequence-producing action cannot execute unless fresh, scoped, attributable, non-replayable authority is presented and admitted at the commit boundary.

# 1. The Governance Gap

---

## 1.1 What Frameworks Currently Provide

The dominant AI governance frameworks share a common architecture. ISO/IEC 42001 establishes an AI management system: it defines organisational roles, lifecycle controls, risk processes, and a continual improvement cycle<sup>[1]</sup>. The NIST AI Risk Management Framework structures governance across four functions — Govern, Map, Measure, Manage — providing vocabulary and process discipline<sup>[2]</sup>. The EU AI Act imposes obligations around transparency, technical documentation, human oversight, and conformity assessment<sup>[3]</sup>.

These are serious instruments. They are not the problem. The problem is what they do not contain.

## 1.2 Policy Documents Describe Intent

A policy document states what an organisation intends to do. It cannot enforce anything at the moment of execution. A policy that says *an AI system shall not modify financial records without authorisation* does not prevent that modification. It records that the modification was prohibited. The distinction matters enormously in agentic systems.

## 1.3 Audit Trails Describe History

Audit logging records what happened. **It is not a control.** An audit trail showing an unauthorised mutation was logged is evidence of the mutation, not evidence it was prevented. This is the **audit theatre problem**: organisations demonstrate governance by producing logs; neither party confirms whether the log represents a boundary that held or a boundary that was crossed and recorded.

## 1.4 Monitoring Describes Behaviour

Runtime monitoring observes behaviour. In most deployed architectures it is advisory. If the action has already reached an execution layer, the state change may already be in progress. Monitoring catches. It does not structurally prevent.

## 1.5 The Agentic AI Forcing Function

These limitations became operationally dangerous when AI systems acquired the ability to chain tool calls, orchestrate sub-agents, write to persistent storage, and accumulate consequence across multi-step pipelines without human review at every step. An agent may execute dozens of consequential actions between any two human checkpoints. The agentic architecture transforms the governance gap from a theoretical concern into a structural hazard.

## 1.6 None Prove the Invalid Action Was Unreachable

The question governance must ultimately answer is not *Did we intend to prevent this?* nor *Did we log it?* It is:

***Was the invalid action physically unreachable?***

ISO/IEC 42001<sup>[1]</sup> does not define a commit boundary. The NIST AI RMF<sup>[2]</sup> does not specify where a control structurally binds at runtime. The EU AI Act's<sup>[3]</sup> human oversight provisions assume a stopping mechanism without specifying its architecture. This gap, in agentic AI deployment, is now load-bearing.

### 1.7 The Missing Object

Current governance frameworks have mature language for policy, process, risk, lifecycle, and audit. They do not yet specify a single runtime object that proves where consequence was refused.

That object is the **commit boundary**: a structural control point through which every consequence-producing action must pass before it becomes effect. It is not advisory. It is not a log. It is not a classifier. It is the point at which unproven authority cannot proceed.

## 2. Definitions

---

Every term used in the architecture carries a precise meaning, defined once here.

- **2.1 Consequence** — Any action that produces an irreversible or externally observable state change.
- **2.2 Commit** — The act of submitting a proposed action for authorisation at the boundary, before the action reaches the execution layer. Commit is not execution.
- **2.3 Mutation** — Any operation that alters persistent state or causes external action. Internal computations and non-exposing reads are not mutations.
- **2.4 Authority** — The token that grants permission for a specific mutation. Must be Fresh (issued at or near commit time), Scoped (names a specific action type and scope), Non-replayable (single-use nonce), and Attributable (signed by a named issuer).
- **2.5 Refusal** — A structured, receipt-generating denial. Not silence. Not an error. A first-class signed output that prevents the action and records that it was prevented.
- **2.6 Proof Surface** — The observable set of artefacts demonstrating that an invalid action was denied and produced no downstream state change. Verifiable by third parties without access to the AI model.

### 3. The CommitGate Model

#### 3.1 Overview

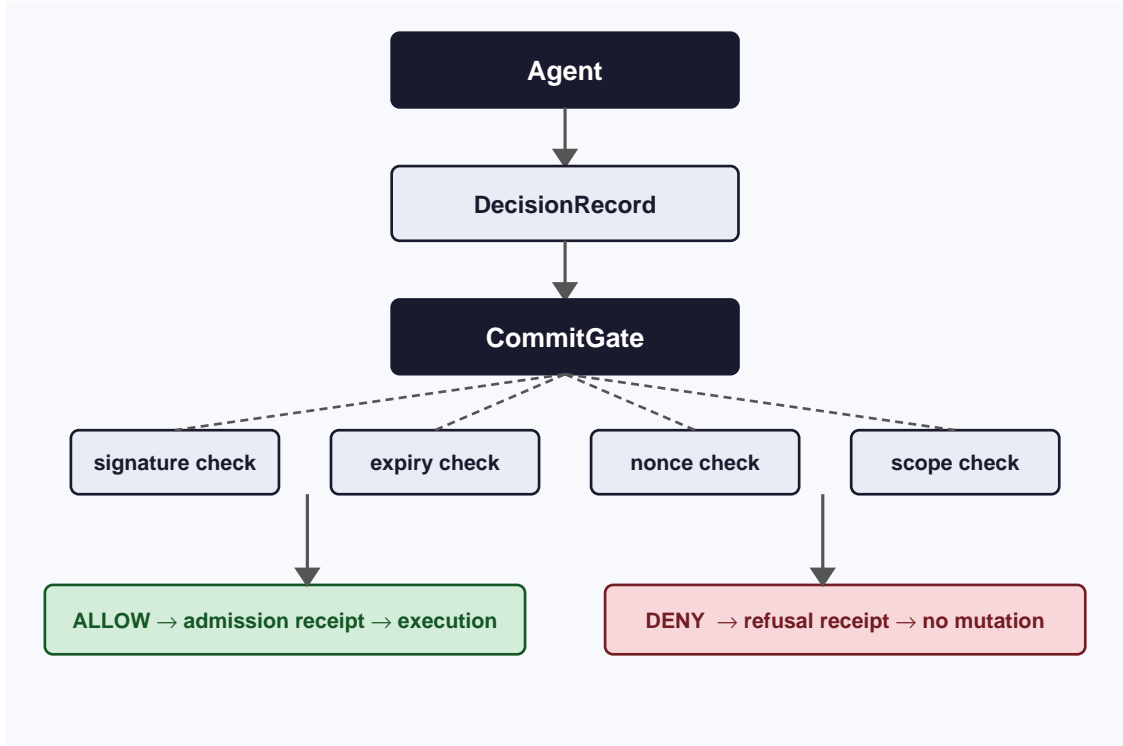
The CommitGate model is a runtime control architecture. Its purpose is to make the commit boundary structurally real: not a policy aspiration, not a monitoring layer, not a logging convention, but a physical prerequisite to execution.

***No mutation may reach the execution layer without first passing through the gate.***

An implementation in which an agent can bypass the gate is not an implementation of the CommitGate model. The gate evaluates whether the presented authority is valid for the presented action at the presented moment. Its narrowness is its strength.

**If any execution pathway exists outside the gate, the model's guarantees do not hold.**

Figure 1 — CommitGate execution flow



#### 3.2 The DecisionRecord

Every commit begins with a **DecisionRecord**: a structured, signed artefact. Every field is required; a missing field causes rejection before any checks begin.

Field	Description
record_id	Globally unique identifier. Generated by submitting agent.
action_type	Declared mutation type. Must match gate action-type registry.
scope	Operational scope claimed. Must match gate scope configuration.

Field	Description
payload_hash	SHA-256 hash of the mutation payload.
authority_token	Signed JWT. Claims: sub, iat, exp, scope, action_type, nonce. RS256/ES256 min.
issuer_id	Identity of authority issuer. Must match sub in token.
issued_at	Token issue timestamp. Must match iat in token.
expires_at	Expiry. Gate evaluates: current_time < expires_at.
nonce	Single-use value. Minimum 128 bits of entropy.
gate_id	Identity of the gate receiving this record.
record_signature	Signature over all preceding fields. RS256/ES256 min.

The DecisionRecord is immutable once submitted. Appended to the audit trail whether admitted or refused.

### 3.3 Gate Checks — First-Fail Semantics

The gate stops at the first failed check and issues a refusal.

- **Check 1 — Signature verification.** Verifies record\_signature. Any failure: immediate rejection.
- **Check 2 — Expiry.** Checks expires\_at against gate clock. No grace window.
- **Check 3 — Nonce.** Checks nonce against persistent register. Previously seen nonce refused permanently.
- **Check 4 — Scope admissibility.** Checks whether action\_type is admissible under declared scope per gate's own configuration. Scope is not self-certifying.

If all four pass, the gate issues an admission receipt and token. Only then may the mutation proceed.

### 3.4 Refusal Semantics

A **refusal receipt** is a signed output, not an error. Written to the append-only audit trail before any downstream action. A refused action produces no mutation.

Field	Description
receipt_id	Unique identifier for this refusal receipt.
record_ref	Reference to the refused DecisionRecord.
failed_check	SIGNATURE_INVALID   EXPIRED   NONCE_REPLAYED   SCOPE_INADMISSIBLE
refusal_reason	Structured reason code within the failed check.
refused_at	Timestamp of refusal.
gate_id	Identity of the refusing gate.

Field	Description
refusal_signature	Signature over all preceding receipt fields.

### 3.5 Admission Semantics

Admission means one thing only: the presented authority was valid for this mutation at this boundary at this time. It does not mean the action is good or the AI is correct.

### 3.6 Audit Trail

Every gate outcome is written to an append-only audit trail at gate time, before any response is returned. The trail is part of the control surface, not a later reporting layer.

### 3.7 Core Invariant

***No proof, no transition. No transition, no execution. No execution, no consequence.***

Implementation details matter, but they serve this invariant.



## 4. Threat Model

---

### 4.1 Purpose

A control architecture is only credible if it names the attacks it is designed to defeat. This section defines six adversarial conditions and the gate's structural defence against each.

### 4.2 Threat 1 — Stale Authority

**Threat:** Legitimately issued token reused after validity window closed.

**Gate defence:** Expiry check. `expires_at` evaluated strictly; no grace window.

**Residual:** Gate enforces the freshness window. It does not judge whether that window was appropriate.

### 4.3 Threat 2 — Scope Drift

**Threat:** Agent acts outside declared operational scope.

**Gate defence:** Scope admissibility check. Gate uses its own configuration. Agent does not self-certify.

**Residual:** Gate enforces configured scope. Incorrect scope definition produces incorrect boundary.

### 4.4 Threat 3 — Replay Attack

**Threat:** Previously admitted record resubmitted to cause same mutation again.

**Gate defence:** Nonce check. Once admitted, nonce permanently registered. Reuse refused.

**Residual:** Does not defeat a compromised issuer generating new tokens with fresh nonces.

### 4.5 Threat 4 — Silent Override

**Threat:** Action executes through a pathway bypassing the gate entirely.

**Gate defence:** Architectural invariant. The gate is the sole path to the execution layer. If any execution pathway exists outside the gate, the model's guarantees do not hold. No exceptions.

**Residual:** Cannot be defeated from inside the gate. Must be verified by structural audit.

### 4.6 Threat 5 — Advisory-Only Control

**Threat:** Gate evaluates actions but its decisions are not enforced.

**Gate defence:** Model definition. A non-binding gate is not a `CommitGate` implementation.

**Residual:** Assurance operators must verify execution layer requires valid admission token.

### 4.7 Threat 6 — Post-Hoc Audit Theatre

**Threat:** Logs written after execution; boundary compliance cannot be proven.

**Gate defence:** Gate-time audit write. Trail written before any downstream action.

**Residual:** Trail must be independently append-only and protected from modification.

#### 4.8 Threat Summary

Threat	Attack Vector	Gate Defence	Failure if Absent
Stale authority	Expired token reused	Expiry check	Historical authority grants ongoing effect
Scope drift	Out-of-scope action submitted	Scope check	Agent self-certifies admissibility
Replay attack	Admitted record resubmitted	Nonce check	One authorisation repeats indefinitely
Silent override	Execution bypasses gate	Architectural invariant	Gate exists but controls nothing
Advisory-only control	Gate consulted, not obeyed	Model definition	Refusals are ignored
Post-hoc audit theatre	Logs written after execution	Gate-time audit write	Audit describes history, not governance

## 5. The Proof Surface

---

### 5.1 What Proof Means Here

The word *proof* is used precisely: a defined, observable, replayable evidentiary record allowing a third party to determine — without relying on the claiming party's testimony — that a specific invalid action was denied and produced no downstream consequence.

### 5.2 The Five Conditions

All five must be present simultaneously.

- **Condition 1 — Attempt recorded.** A record shows the action was presented at the gate, independently of the submitting agent.
- **Condition 2 — Denial observable and attributable.** A signed refusal receipt names the failed check and the gate that issued it.
- **Condition 3 — No downstream mutation.** Downstream mutation log shows no effect from the refused payload hash.
- **Condition 4 — Signed receipt durable.** Receipt exists as a durable, signed artefact in the append-only trail.
- **Condition 5 — Sequence replayable.** Full sequence reconstructable from the audit trail alone.

### 5.3 What the Proof Surface Does Not Prove

- It does not prove: The AI is correct.
- It does not prove: The scope was correctly defined.
- It does not prove: The authority issuer was trustworthy.
- It does not prove: All governance requirements are met.
- It does not prove: Non-mutation harms are governed.

### 5.4 The Single Claim

***“At this gate, at this moment, for this action, the invalid did not become consequence.”***

That claim is verifiable, bounded, and observable by a third party without access to the AI model.

### 5.5 Assurance Operator Verification

Required: the gate's public key, the append-only audit trail, the downstream mutation log. The operator asks five questions corresponding to the five conditions. If all five: yes — **the boundary held.**

## 6. Mapping to Live Standards

The CommitGate model does not replace existing AI governance frameworks. It fills a structural gap that each leaves open.

### 6.2 ISO/IEC 42001<sup>[1]</sup>

**What it provides.** Establishes an AI management system: roles, lifecycle controls, risk processes, operational planning, continual improvement.

**Gap.** Clause 8 requires documented evidence that controls were applied but does not specify a runtime mechanism by which a mutation is structurally conditional on authorisation before execution.

**CommitGate attachment.** Commit boundary is the Clause 8 operational control. Audit trail is documented evidence. Refusal receipt proves the control was tested and held.

### 6.3 NIST AI Risk Management Framework<sup>[2]</sup>

**What it provides.** Structures risk management across Govern, Map, Measure, Manage.

**Gap.** MANAGE function describes risk response without specifying how controls physically bind at the moment of action.

**CommitGate attachment.** CommitGate operationalises MANAGE with a structural enforcement mechanism. Proof surface operationalises MEASURE.

### 6.4 EU AI Act<sup>[3]</sup>

**What it provides.** Imposes obligations for high-risk AI: robustness (Article 15), transparency (Article 13), human oversight (Article 14), technical documentation (Annex IV).

**Gap.** Article 14 requires oversight mechanisms built into the system but does not specify the architecture under adversarial conditions.

**CommitGate attachment.** CommitGate provides the structural mechanism Article 14 assumes. The refusal receipt provides Annex IV documentation as boundary evidence, not description.

### 6.5 AI Assurance Operators

**What it provides.** Reviews documentation, architecture diagrams, risk registers, monitoring outputs.

**Gap.** No defined evidentiary object at the execution boundary.

**CommitGate attachment.** Proof surface is designed for assurance inspection. Assurance shifts from *does documentation describe controls* to *did this boundary hold on this date*.

### 6.6 Standards Mapping Summary

Framework	Strength	Gap	CommitGate Attachment
ISO/IEC 42001 [1]	Management-system governance	No runtime enforcement mechanism	Commit boundary; audit trail as evidence

Framework	Strength	Gap	CommitGate Attachment
NIST AI RMF [2]	Risk-management structure	No binding control at execution	Operationalises Manage and Measure
EU AI Act [3]	Legal obligations for oversight	Stop mechanism assumed, not specified	Commit boundary enforces oversight
AI assurance	External review	No bounded runtime proof object	Proof surface: receipts, refusals, replayable evidence

## 6.7 Claim Boundary

CommitGate provides one missing control object: a verifiable execution boundary where invalid action can be refused before consequence.

## 7. Limit Boundary

---

### 7.1 Purpose

This section states what the CommitGate model does not claim. The limit boundary is not a weakness. It is a condition of the model's credibility.

### 7.2 What This Paper Does Not Claim

- **Does not prove the AI is correct.** The gate evaluates authority, not judgement.
- **Does not prove the scope definition is correct.** The gate enforces the scope it was given. Scope design is a policy responsibility.
- **Does not prove the authority issuer was trustworthy.** A compromised issuer can produce valid tokens for invalid purposes.
- **Does not solve all AI governance problems.** Risk assessment, model evaluation, bias analysis, incident response, and accountability remain unchanged.
- **Does not govern non-mutation harms.** Harmful outputs through inference alone are outside the gate's control surface.

### 7.3 What This Paper Does Claim

- **First.** A commit boundary can be structurally defined and implemented.
- **Second.** Refusal can be made observable and replayable.
- **Third.** No unproven action can become consequence through a correctly implemented gate.

***No proof, no transition. No transition, no execution. No execution, no consequence.***

### 7.4 The Single Claim

***“This boundary held.”***

That statement can be confirmed from the audit trail, verified by a third party, and repeated for every commit at every gate. A system that can prove, for each consequential mutation, that the boundary held has made the invalid unreachable at the execution layer.

### 7.5 Final Scope Note

This paper gives the field one missing object: a name, a definition, a mechanism, a proof surface, and a clean claim boundary. That object is the commit boundary — the place where a governed AI-enabled system can still be stopped. Everything beyond that boundary is, as it always was, a harder problem.

## References

---

- [1] ISO/IEC 42001:2023, *Artificial intelligence — Management system*, International Organization for Standardization, 2023.
- [2] National Institute of Standards and Technology, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, NIST AI 100-1, January 2023.
- [3] Regulation (EU) 2024/1689 of the European Parliament and of the Council laying down harmonised rules on artificial intelligence, *Official Journal of the European Union*, 2024.
- [4] European Commission, *Ethics Guidelines for Trustworthy AI*, High-Level Expert Group on Artificial Intelligence, 2019.
- [5] OWASP, *Top 10 for Large Language Model Applications*, current public release. Available: [owasp.org/www-project-top-10-for-large-language-model-applications](https://owasp.org/www-project-top-10-for-large-language-model-applications)
- [6] MITRE ATLAS, *Adversarial Threat Landscape for Artificial-Intelligence Systems*, current public release. Available: [atlas.mitre.org](https://atlas.mitre.org)

## Appendix A — DecisionRecord Reference Schema

---

Every field is required. Missing or null fields cause rejection before any checks begin.

### A.2 DecisionRecord Schema

```
DecisionRecord {
  record_id:          UUID          // Globally unique. Generated by submitting a
gent.
  action_type:        Enum(String)  // Examples: DATA_WRITE, EXTERNAL_API_CALL, F
ILE_DELETE,
                                // MESSAGE_SEND, PAYMENT_INITIATE, CONFIG_C
HANGE.
  scope:              String        // Must match gate scope configuration.
  payload_hash:       String        // SHA-256 of mutation payload.
  authority_token:     SignedJWT     // Claims: sub, iat, exp, scope, action_type,
nonce.
  issuer_id:          String        // Must match sub in authority_token.
  issued_at:          ISO8601       // Must match iat in authority_token.
  expires_at:         ISO8601       // Gate evaluates: current_time < expires_at.
  nonce:              String        // Single-use. Minimum 128 bits entropy.
  gate_id:            String        // Identity of receiving gate.
  record_signature:   String        // RS256/ES256 over all preceding fields.
}
```

### A.3 Refusal Receipt Schema

```
RefusalReceipt {
  receipt_id:         UUID          // Unique identifier.
  record_ref:         UUID          // record_id of refused DecisionRecord.
  failed_check:       Enum          // SIGNATURE_INVALID|EXPIRED|NONCE_REPLAYED|SCOP
E_INADMISSIBLE
  refusal_reason:     String        // Structured reason code.
  refused_at:         ISO8601       // Timestamp of refusal.
  gate_id:            String        // Identity of refusing gate.
  refusal_signature:  String        // Signature over all preceding fields.
}
```

### A.4 Admission Receipt / A.5 Audit Trail Entry

```
AdmissionReceipt { receipt_id, record_ref, admitted_at, gate_id, admission_token
, admission_signature }
AuditEntry { entry_id, sequence(monotonic/no-gaps), entry_type(ADMISSION|REFUSAL
),
            record_ref, receipt_ref, preceding_hash(SHA-256 chain), written_at,
entry_signature }
```



## Appendix B — Standards Mapping Table

Attachment map, not a compliance claim.

CommitGate Component	ISO/IEC 42001 [1]	NIST AI RMF [2]	EU AI Act [3]
Commit boundary	Clause 8.1 — Operational planning	MANAGE 2.2 — Treatment plans implemented	Article 14 — Oversight built into system
Signed DecisionRecord	Clause 7.5 — Documented information	MAP 1.1 — Context documented	Annex IV — Technical documentation
Scope check	Clause 6.1 — Actions to address risk	MAP 5.1 — Impact identified	Article 9 — Risk management
Expiry check	Clause 8.4 — Lifecycle processes	MANAGE 2.4 — Treatments monitored	Article 15 — Robustness
Nonce / replay protection	Clause 8.4 — Lifecycle processes	MANAGE 2.4 — Treatments monitored	Article 15 — Cybersecurity
Refusal receipt	Clause 9.1 — Monitoring	MEASURE 2.6 — Measurement documented	Article 12 — Logging
Append-only audit trail	Clause 9.1 — Performance evaluation	MEASURE 2.8 — Documented evidence	Article 12 — Automatic logging
Proof surface	Clause 9.2 — Internal audit	MEASURE 4.1 — Approaches reviewed	Article 17 — Quality management
Gate-time audit write	Clause 9.1 — At time of operation	MANAGE 4.1 — Treatments verified	Article 12 — Events at time of occurrence
First-fail refusal semantics	Clause 8.1 — Operational controls	MANAGE 2.2 — Response before consequence	Article 14.4 — Oversight prevents undue influence

## Appendix C — Expanded Threat Model

---

### C.2 Threat 1 — Stale Authority

<b>Category</b>	Token lifecycle
<b>Attack vector</b>	Legitimately issued token reused after validity window closed.
<b>Preconditions</b>	Token legitimately issued; agent retained beyond intended use.
<b>Gate control</b>	Check 2 — Expiry: <code>current_time &gt;= expires_at</code> triggers refusal.
<b>Failure mode</b>	Historical authority produces ongoing effect.
<b>Residual risk</b>	Gate cannot judge appropriateness of freshness window.
<b>Assurance</b>	Confirm <code>refused_at &gt; expires_at</code> ; confirm no downstream mutation for refused <code>payload_hash</code> .

### C.3 Threat 2 — Scope Drift

<b>Category</b>	Scope violation
<b>Attack vector</b>	Agent submits <code>action_type</code> outside declared scope.
<b>Preconditions</b>	Agent has valid authority for some scope; action falls outside it.
<b>Gate control</b>	Check 4 — Scope admissibility triggers refusal.
<b>Failure mode</b>	Agent self-certifies admissibility.
<b>Residual risk</b>	Incorrect scope definition produces incorrect boundary.
<b>Assurance</b>	Confirm gate scope config; confirm <code>action_type</code> inadmissible; confirm no downstream effect.

### C.4 Threat 3 — Replay Attack

<b>Category</b>	Token reuse
<b>Attack vector</b>	Previously admitted record resubmitted.
<b>Preconditions</b>	Attacker has access to a prior admitted record or token.
<b>Gate control</b>	Check 3 — Nonce: previously registered nonce triggers refusal.
<b>Failure mode</b>	Single authorisation repeats indefinitely.
<b>Residual risk</b>	Does not prevent compromised issuer generating new tokens with fresh nonces.
<b>Assurance</b>	Confirm nonce in register; confirm <code>NONCE_REPLAYED</code> in receipt; confirm no mutation.

### C.5 Threat 4 — Silent Override

<b>Category</b>	Architectural bypass
-----------------	----------------------

<b>Attack vector</b>	Mutation reaches execution layer via ungated pathway.
<b>Preconditions</b>	Ungated pathway exists and is accessible.
<b>Gate control</b>	Architectural invariant. If any execution pathway exists outside the gate, the model's guarantees do not hold.
<b>Failure mode</b>	Gate controls nothing.
<b>Residual risk</b>	Gate cannot verify own exclusivity; structural audit required.
<b>Assurance</b>	Map all execution pathways; test direct mutation without admission token.

## C.6 Threat 5 — Advisory-Only Control

<b>Category</b>	Control degradation
<b>Attack vector</b>	Execution layer proceeds regardless of gate decision.
<b>Preconditions</b>	Execution layer not architecturally required to hold for gate admission.
<b>Gate control</b>	Model definition — advisory mode excluded.
<b>Failure mode</b>	Refusals are ignored.
<b>Residual risk</b>	Cannot be detected from inside the gate.
<b>Assurance</b>	Verify execution layer requires valid admission token; test without token.

## C.7 Threat 6 — Post-Hoc Audit Theatre

<b>Category</b>	Evidence integrity
<b>Attack vector</b>	Logging performed after execution or by execution layer.
<b>Preconditions</b>	Audit trail not written by gate at gate time.
<b>Gate control</b>	Gate-time audit write — before any downstream action.
<b>Failure mode</b>	Receipts can be fabricated retrospectively.
<b>Residual risk</b>	Trail integrity must be independently verified.
<b>Assurance</b>	Confirm trail write timestamps precede execution; verify preceding_hash chain.

## Appendix D — Proof Slice Example

---

### D.1 Purpose

A concrete worked example of the CommitGate proof surface: an unauthorised payment initiation.

### D.2 Scenario

An AI agent operating within a customer support pipeline submits a `PAYMENT_INITIATE` action. The agent's authority token was issued for scope `CUSTOMER_SUPPORT`. The gate's configuration does not admit `PAYMENT_INITIATE` under `CUSTOMER_SUPPORT`.

### D.3 Execution Trace

- Step 1.** Agent constructs DecisionRecord: `action_type: PAYMENT_INITIATE`, `scope: CUSTOMER_SUPPORT`, with fresh authority token.
- Step 2.** Agent signs the DecisionRecord and submits to CommitGate.
- Step 3.** Gate verifies `record_signature`. Check 1 passes.
- Step 4.** Gate checks `expires_at`. Token is within freshness window. Check 2 passes.
- Step 5.** Gate checks nonce against register. Nonce is new. Check 3 passes.
- Step 6.** Gate checks whether `PAYMENT_INITIATE` is admissible under `CUSTOMER_SUPPORT`. It is not. Check 4 fails.
- Step 7.** Gate issues refusal receipt: `failed_check = SCOPE_INADMISSIBLE`. Signed, written to append-only audit trail before any response is returned.
- Step 8.** No admission token issued. Execution layer receives no instruction. No payment instruction appears in downstream payment log.
- Step 9.** Agent receives refusal receipt. Session ends.

### D.4 Proof Surface Verification

An assurance operator applies the five conditions:

- **Condition 1 — Attempt recorded.** Audit trail contains DecisionRecord with `record_id`, `action_type: PAYMENT_INITIATE`, `scope: CUSTOMER_SUPPORT`.
- **Condition 2 — Denial observable and attributable.** Audit trail contains refusal receipt signed by the gate, naming `failed_check: SCOPE_INADMISSIBLE`.
- **Condition 3 — No downstream mutation.** Payment system transaction log shows no entry matching the refused `payload_hash` after `refused_at` timestamp.
- **Condition 4 — Signed receipt durable.** Refusal receipt retrieved from append-only trail. `refusal_signature` verifies against gate's public key.
- **Condition 5 — Sequence replayable.** Replay of audit trail produces the same sequence. No gap in `preceding_hash` chain.

**Result:** The payment was attempted, refused, recorded, and did not become consequence. All five proof surface conditions are satisfied. The boundary held.