Round Trips Can Lie: Policy-Laundering Attacks on Cross-Language Validation Adapters

Anonymous Author(s)

Abstract

A recent Go-to-Rust validation architecture first checks adapter round trips and then reuses the adapters to compare program outputs. Even exhaustive round-trip checks do not establish semantic alignment. Over a shared total bijective carrier domain, conjugating one endpoint's adapter by a carrier permutation preserves native, carrier-domain, and full cross-language round trips, yet can make a source `DENY` transport to the same target-native value as a target `ALLOW`. We call this a policy-laundering attack.

We present GlueRift, a finite checker that treats the workflow's target-native comparator as a first-class input. It separates policy soundness, which forbids unsafe induced equalities, from comparison adequacy, which requires declared matches to compare equal. Precision and faithfulness expose extra equality and exact realization. This two-sided criterion rejects both false agreement and vacuous always-different adapters. The checker uses closed adapter and observer languages, exhaustively decides supported finite scopes, classifies generated transformations, and returns concrete paths and witnesses; unsupported semantics are reported as unknown.

We mechanize the total finite core in Lean and evaluate four distinct law-preserving attacks, benign and vacuity regressions, and total composition. Two separate-process Go/Rust replays through a shared Protobuf carrier reproduce target-native false agreement: an enum decision reversal and a nested repeated-type field-role swap. All six round trips pass in both. A direct-relation baseline, given the same semantics, reaches exactly the same top-level verdicts and first witnesses. Thus our contribution is not a new lens law or a logically stronger relation checker; it is an operational attack on a concrete validation seam and a reproducible diagnostic realization.

CCS Concepts

• **Security and privacy → Formal security models**; • **Software and its engineering → Software verification and validation**.

Keywords

translation validation, interoperability, round trips, policy laundering, bidirectional transformations

ACM Reference Format:

Anonymous Author(s). 2026. Round Trips Can Lie: Policy-Laundering Attacks on Cross-Language Validation Adapters. In *Proceedings of Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '26).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Cross-language validation needs a way to compare values that do not share a native representation. This is especially difficult for opaque, library-defined values. A recent Go-to-Rust translation pipeline synthesizes Go and Rust adapters around a shared Protobuf carrier, checks source-side and full cross-language round trips, and then reuses those adapters for differential I/O comparison [11]. For a source function f, target function g, and input/output adapters A_I, A_O, the final test is $A_O(f(v)) = g(A_I(v))$ on observed test inputs. Round-trip success is therefore admission evidence for glue that later participates in the equivalence verdict.

The motivating workflow checks a Go-side round trip and a full Go-to-Rust-to-Go round trip on representative values. Our counterexample strengthens, rather than weakens, this admission criterion: the reference artifact checks both native laws, both carrier-domain laws, and both full transports exhaustively over declared finite domains. The attack therefore does not depend on asymmetric or sampled round-trip testing.

This evidence leaves an ambiguity. For a permutation $\sigma : C \to C$, the conjugated target maps $\sigma \circ e_T$ and $d_T \circ \sigma^{-1}$ insert a change and its inverse into every cycle. In this shared bijective core they cancel, so every round trip still passes. They need not cancel at the downstream comparison boundary. The twisted decoder may transport a source denial to the target's allowed constructor; if the target program also returns that allowed value, the validator reports equality between policy-opposite executions.

This is not a claim that round-trip ambiguity or automorphism invariance is new. Lens research supplies round-trip laws, consistency relations, quotients, contracts, and composition [5–7, 10, 12]. Cycle-consistency theory gives a closely related automorphism ambiguity [8], and language-interoperability research verifies glue conversion relative to an independently stated semantic relation [9]. Our question is more operational and narrower:

> Can glue admitted by all ordinary adapter round trips cause the *workflow's target-native comparison cut* in a cross-language validation workflow to report unsafe program agreement?

We answer yes with a theorem, finite generated attacks, and native replays. We then ask what additional evidence excludes the attack without accepting the opposite vacuity—an adapter that never equates anything. GlueRift uses endpoint-owned relations: Safe describes pairs that may safely compare equal, while Match describes pairs that must compare equal. Their two inclusions define soundness and adequacy. The validator comparator is explicit: carrier equality, source-native equality, and target-native equality are different relations and no bridge is assumed between them.

This paper makes six contributions:

(1) We exhibit a target-native policy-laundering attack in which six exhaustive round trips pass but the validator reports unsafe equality.
(2) We give a comparator-indexed formalization that separates soundness, adequacy, precision, and faithfulness over a declared validation scope.
(3) We characterize lawful twists and distinguish policy harm from transformations that merely break a requested law.

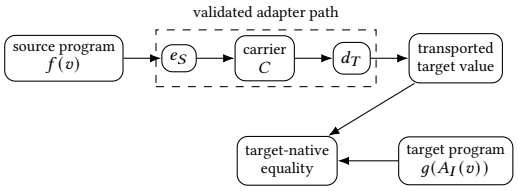

Figure 1: The target-native comparison seam. Round trips validate the adapter path; the final equality is between native target values.

(4) We implement a closed finite checker that generates structural attacks, reports concrete witnesses and adapter paths, and returns unknown rather than approximating unsupported semantics.
(5) We mechanize nine theorem and counterexample groups in Lean and provide two separate-process Go/Rust/Protobuf replays.
(6) We compare against an equally expressive direct-relation baseline and require verdict and first-witness parity.

We do not claim a vulnerability in the motivating authors' public implementation, which we did not execute. Our native experiments are an architecture-path-faithful reconstruction of the adapter and comparator topology. Nor do we claim arbitrary native-code verification, whole-program equivalence, attack prevalence, or completeness beyond the declared finite language.

2 The Validation Seam

2.1 Architecture and threat model

Let S, T, and C be finite source-native, target-native, and carrier domains. An adapter context contains four partial maps:

$$e_S : S \to \mathsf{Result}(C), \qquad d_S : C \to \mathsf{Result}(S),$$
$$e_T : T \to \mathsf{Result}(C), \qquad d_T : C \to \mathsf{Result}(T).$$

The motivating output comparison transports a source output through e_S and d_T, then compares the resulting target-native value with the target program's output. Figure 1 shows the seam.

The untrusted component is the adapter context: the four maps and any carrier transformation. The policy owner supplies finite comparison domains, a pair universe $U \subseteq S \times T$, the comparator, and endpoint semantics. The adversary may choose a well-typed adapter that passes all requested round trips, but cannot alter the comparator or the endpoint policy. This models an erroneous or adversarially misaligned generated adapter, not a compromised compiler or program.

In the output-laundering fixtures, the source and target program bodies and the input adapter are fixed; the input adapter is honest. The target program's permissive output is a pre-existing program disagreement. The adversarial output adapter does not create that disagreement: it causes the validator to misclassify it as equality.

The Core is deliberately total on every admitted comparison and round-trip input. Object-language `Result` is an ordinary sum whose branches may be permuted; meta-level conversion failure is not silently ignored. Effectful partial composition is outside our claims.

2.2 The six round trips

The request explicitly names six laws. Native round trips require

$$d_S(e_S(s)) = \mathsf{Ok}(s), \qquad d_T(e_T(t)) = \mathsf{Ok}(t).$$

Carrier-domain round trips, on policy-owned $K_S, K_T \subseteq C$, require

$$e_S(d_S(c)) = \mathsf{Ok}(c) \quad (c \in K_S), \quad e_T(d_T(c)) = \mathsf{Ok}(c) \quad (c \in K_T).$$

Full transports require the source-to-target-to-source and reverse cycles to be defined and return their starting values:

$$d_S(e_T(d_T(e_S(s)))) = \mathsf{Ok}(s),$$
$$d_T(e_S(d_S(e_T(t)))) = \mathsf{Ok}(t).$$

Composition above uses result binding; an intermediate error fails the law. A profile does not add laws implicitly.

These six laws are stronger than the sampled, Go-origin cycles in the motivating workflow. They are the reference artifact's admission request, not a claim about which laws that workflow implements.

2.3 A minimal laundering execution

Take two source decisions, `DENY` and `ALLOW`, two target decisions, `Blocked` and `Permitted`, and a two-value carrier. The aligned context maps denials to carrier 0 and allowances to carrier 1. Let σ swap 0 and 1, and twist only the target context:

$$e_T^\sigma = \sigma \circ e_T, \qquad d_T^\sigma = d_T \circ \sigma^{-1}.$$

Then $d_T^\sigma(e_S(\texttt{DENY})) = \texttt{Permitted}$. If the source program denies and the target program permits, the transported source output and target output are both `Permitted`. The ordinary comparator says equal even though the endpoint-owned policy says the pair is unsafe. Section 6 reproduces this execution in separate Go and Rust processes through Protobuf.

The finite A01 instance makes every set explicit. Its comparison universe is the four-pair product

$$U = \{\texttt{DENY}, \texttt{ALLOW}\} \times \{\texttt{Blocked}, \texttt{Permitted}\},$$

and endpoint roles define

$$\mathsf{Safe} = \mathsf{Match} = \{(\texttt{DENY}, \texttt{Blocked}), (\texttt{ALLOW}, \texttt{Permitted})\}.$$

The aligned context induces exactly this relation. The carrier swap instead induces

$$I_{A^\sigma}^T = \{(\texttt{DENY}, \texttt{Permitted}), (\texttt{ALLOW}, \texttt{Blocked})\},$$

so both induced pairs are unsafe while both required matches disappear.

3 Comparator-Indexed Security

3.1 Induced relations

Carrier equality is not the comparison in Figure 1. Accordingly, the comparator specification chooses one of three induced relations:

$$E_A^C(s,t) \iff \exists c.\ e_S(s) = \mathsf{Ok}(c) = e_T(t), \tag{1}$$
$$E_A^T(s,t) \iff e_S(s) \gg\!= d_T = \mathsf{Ok}(t), \tag{2}$$
$$E_A^S(s,t) \iff e_T(t) \gg\!= d_S = \mathsf{Ok}(s). \tag{3}$$

The selected relation is $I_A^\chi = E_A^\chi \cap U$. Comparator definedness is a positive obligation over all of U; partial results do not make implications vacuously true.

The three relations can differ despite all six round trips. In our V01 model, source and target encoders use disjoint carrier images, so $E_A^C = \varnothing$, while both cross-decoders collapse the images to the corresponding native values. All six laws hold, yet $E_A^T = E_A^S = \{(s_0, t_0), (s_1, t_1)\}$. Thus carrier-based checking would miss a native false agreement. GlueRift checks the selected native relation directly. It produces a separate exhaustive bridge result only when carrier summaries are reused for native claims.

3.2 Safety, required matches, and two vacuities

The policy owner declares

$$\mathsf{Match} \subseteq \mathsf{Safe} \subseteq U.$$

$\mathsf{Safe}(s, t)$ means that equating the pair is permitted by the security policy. $\mathsf{Match}(s, t)$ means that the validator is required to equate it. They induce four distinct checks:

$$\begin{aligned} \mathsf{Sound}_\chi(A) &\iff I_A^\chi \subseteq \mathsf{Safe}, \\ \mathsf{Adequate}_\chi(A) &\iff \mathsf{Match} \subseteq I_A^\chi, \\ \mathsf{Precise}_\chi(A) &\iff I_A^\chi \subseteq \mathsf{Match}, \\ \mathsf{Faithful}_\chi(A) &\iff I_A^\chi = \mathsf{Match}. \end{aligned}$$

Soundness prevents unsafe false agreement. Adequacy prevents an always-different adapter from appearing safe. Precision reports safe but undeclared extra equality. Faithfulness is exact comparison.

Coverage is not inferred from an adapter's successful image. For example, source-total coverage is

$$\forall s \in D_S^{cmp}.\ \exists t \in D_T^{cmp}.\ (s, t) \in \mathsf{Match},$$

and target-total is symmetric; bidirectional-total requires both. An empty Match under nonempty coverage makes the request invalid before adapter properties are evaluated. If the safety policy is unconstrained (no safety dimensions, hence $\mathsf{Safe} = U$), the checker reports the vacuity and withholds a security certificate even though the set inclusion for soundness is mathematically true.

Target non-amplification is one useful asymmetric safety instance. If policy levels form a checked finite preorder, it permits equality only when the target is no more permissive than the source. Its safe transformations need not form a subgroup; Section 4 keeps exact stabilizers separate from asymmetric safe sets.

3.3 Shape checks, bridges, and certification

The comparator choice also constrains the shape of a realizable Match. Because encoders and decoders are deterministic, E_A^T is functional from source values to target values. If the source full-transport law holds over the comparison source domain, its covered graph is also inverse-functional: two covered sources cannot transport to the same target and then both return through the full cycle. The source-native case is symmetric. Carrier equality becomes a partial bijection when the corresponding native and carrier laws hold on the admitted images. GlueRift checks these comparator-specific shape obligations before evaluating adequacy, so an impossible policy request is invalid rather than an adapter failure.

Carrier evidence moves to a native claim only through an explicit bridge. For example, the carrier-to-target bridge over U is

$$\forall (s, t) \in U. \quad E_A^C(s, t) \iff E_A^T(s, t).$$

It is checked exhaustively and reported as proved, disproved, or unknown. The target-native property itself never depends on this bridge: Equation (2) is evaluated directly. Moreover, bridge evidence belongs to one exact context. A bridge for A is not reused after transforming the context to A^σ.

Certificates reflect the distinction among logical results. A diagnostic request cannot certify. A policy-sound profile requires soundness; a policy-sound-adequate profile requires both inclusions; a faithful-exact profile additionally requires Safe = Match and the declared coverage. Any required law, definedness, coverage, anchor, or property failure blocks certification. An unconstrained safety policy always blocks security certification. These rules prevent a raw true implication from being presented as stronger evidence than the policy actually supplied.

4 Law-Preserving Twists

4.1 Preservation and laundering

Assume total mutually inverse maps $e_S : S \cong C : d_S$ and $e_T : T \cong C : d_T$ over one shared carrier domain C, and a carrier automorphism $\sigma : C \cong C$. Define $e_T^\sigma = \sigma \circ e_T$ and $d_T^\sigma = d_T \circ \sigma^{-1}$; the source maps are unchanged.

Theorem 1 (round-trip preservation). Under these assumptions, replacing the target maps by (e_T^σ, d_T^σ) preserves the source and target native round trips, the source and target carrier round trips, and both full cross-language transport round trips over S, T, and C.

Proof sketch. Source laws are unchanged. Target native and carrier laws follow from

$$\begin{aligned} d_T^\sigma \circ e_T^\sigma &= d_T \circ \sigma^{-1} \circ \sigma \circ e_T = d_T \circ e_T, \\ e_T^\sigma \circ d_T^\sigma &= \sigma \circ e_T \circ d_T \circ \sigma^{-1} = \mathrm{id}_C. \end{aligned}$$

The full transports require the endpoint carrier inverse laws, not only an adjacent syntactic cancellation:

$$\begin{aligned} RT_{STS}^\sigma &= d_S \circ \sigma \circ e_T \circ d_T \circ \sigma^{-1} \circ e_S \\ &= d_S \circ \sigma \circ \mathrm{id}_C \circ \sigma^{-1} \circ e_S = \mathrm{id}_S, \\ RT_{TST}^\sigma &= d_T \circ \sigma^{-1} \circ e_S \circ d_S \circ \sigma \circ e_T \\ &= d_T \circ \sigma^{-1} \circ \mathrm{id}_C \circ \sigma \circ e_T = \mathrm{id}_T. \end{aligned}$$

The Lean development proves these clean total-domain statements. □

The executable checker does not infer this theorem outside the clean core; for partial carrier domains it rechecks every request-scoped law exhaustively. Equation (2) nevertheless becomes

$$E_{A^\sigma}^T(s, t) \iff d_T(\sigma^{-1}(e_S(s))) = t,$$

so the target-native graph may change.

Corollary 2 (request-scoped policy laundering). Under Theorem 1, let $\mathcal{R}$ be a valid request whose required law set is contained in the six laws above. If the declared family admits the normalized automorphism $\sigma \in \Sigma_{\mathcal{F}}(A)$ with its inverse and action domain C, and there exists $(s, t) \in U$ such that $E_{A^\sigma}^\chi(s, t)$ and $\neg\mathsf{Safe}(s, t)$, then

$$\sigma \in \mathsf{HarmfulTrans}_{\chi,\mathcal{F}}(A, \mathcal{R}).$$

The algebra is familiar from automorphism ambiguity; the security content is the independently owned unsafe witness at the comparator specified for program agreement.

Table 1: Minimal Core boundary.

Layer	Supported Core forms
Types	Unit, Bool, bounded integer, bit-vector, sum, product, object Result
Adapters	Identity, compose, enum/field/result permutations, structural maps, complement, modular affine
Observers	Constructor/field role, finite policy map, tuple, case
Relations	Exact, checked finite preorder, finite table
Decision	Exhaustive over declared finite scope; unsupported is unknown
Composition	Total-success exact and finite-preorder judgments

4.2 Lawfulness is request-scoped

Not every transformation candidate is an attack. For the finite normalized candidate set $\Sigma_{\mathcal{F}}(A)$ admitted by a declared family and a request $\mathcal{R}$, GlueRift first defines

$$\begin{aligned}\sigma \in \mathsf{Lawful}_{\chi,\mathcal{F}}(A,\mathcal{R}) \iff\ & \mathsf{WT}(A^\sigma) \wedge \mathsf{Inverse}(\sigma) \\ & \wedge \mathsf{Conjugated}(A, A^\sigma) \\ & \wedge \mathsf{Defined}_\chi(A^\sigma, U) \\ & \wedge \bigwedge_{\ell \in \mathsf{Laws}(\mathcal{R})} \mathsf{Holds}_\ell(A^\sigma).\end{aligned}$$

It then partitions candidates into *lawful-safe*, *lawful-harmful*, and *law-breaking-or-inapplicable*. The last bucket prevents a sound transformation that breaks a carrier round trip from being reported as safe, and prevents a law-breaking candidate from being advertised as a laundering attack.

Formally, the two lawful buckets are

$$\begin{aligned}\mathsf{SafeTrans}_{\chi,\mathcal{F}} &= \{\sigma \in \mathsf{Lawful}_{\chi,\mathcal{F}}(A,\mathcal{R}) \mid \mathsf{Sound}_\chi(A^\sigma)\}, \\ \mathsf{HarmfulTrans}_{\chi,\mathcal{F}} &= \{\sigma \in \mathsf{Lawful}_{\chi,\mathcal{F}}(A,\mathcal{R}) \mid \neg\mathsf{Sound}_\chi(A^\sigma)\}, \\ \mathsf{InapplicableTrans}_{\chi,\mathcal{F}} &= \Sigma_{\mathcal{F}}(A) \setminus \mathsf{Lawful}_{\chi,\mathcal{F}}(A,\mathcal{R}).\end{aligned}$$

Thus "harmful" means a concrete unsafe equality under a candidate that still meets the validator's admission request, not merely a malformed conversion.

Only when a structural subfamily $G_{\mathcal{F}}(A) \subseteq \Sigma_{\mathcal{F}}(A)$ has separately established group laws do exact endpoint-label-preserving permutations form the ordinary stabilizer subgroup. We make no subgroup claim for asymmetric policy. Our T01 counterexample uses three positions with source levels (*deny*, *allow*, *allow*) and target levels (*deny*, *deny*, *allow*). Two carrier permutations are each lawful-safe, while their composition remains lawful under all six laws and becomes harmful only because it maps target *allow* to source *deny*.

5 GlueRift

5.1 Closed finite semantics

GlueRift is a reference checker, not a verifier for arbitrary Go or Rust. The Core type language contains unit, Boolean, bounded integers, bit-vectors, sums, products, and object-language `Result`. Adapters include identity and composition; enum, field, and result branch permutations; structural sum/product maps; bounded complement; and modular affine maps. Observers include constructor and field roles, finite policy maps, tuples, and cases. Relations are exact equality, validated finite-preorder non-amplification, or an explicitly enumerated finite table.

All types are enumerated canonically. One semantic kernel evaluates the four maps, three comparators, six laws, policy relations, coverage, four comparison properties, transformations, and witnesses. Anchor coverage conservatively includes every reachable constructor or field path read by any adapter map or active observer; only policy-owned explicit irrelevance can remove a path. A request outside the closed observer language returns unknown, as tested by V06.

The structural analyzer enumerates equal-signature enum variants, same-typed fields, compatible result branches, and their nested compositions. Bounded complement and modular affine maps are admitted declared scalar candidates, not claimed to be automatically or completely discovered. For every generated attack the evidence binds the aligned base context, normalized transformation and inverse, $\mathsf{All}(C)$, the four mechanically conjugated maps, transformed context, requested laws, classification, and first harmful witness.

5.2 Composition boundary

The Core composes only total-success observer judgments. Relative to domains D_X, D_Y and observers o_X, o_Y, write

$$\Gamma \vdash f : X \xrightarrow{R} Y$$

when, for every $x \in D_X$, evaluation returns some $y \in D_Y$ such that $f(x) = \mathsf{Ok}(y)$ and $R(o_X(x), o_Y(y))$. Exact obligations compose by transitivity. Non-amplification obligations compose only after the finite relation is exhaustively checked reflexive and transitive. Global observers are discharged directly unless their structure is syntactically derivable. C01 checks both exact and preorder composition. We do not claim effectful or partial-error composition.

5.3 Mechanization and baselines

The Lean 4 development contains nine checked groups: encoder injectivity from native round trips; twist preservation of native, carrier, and full transport laws; direct target-native laundering; soundness/adequacy vacuity and V01 divergence; comparator bridges; native relation shape; exact stabilizers and asymmetric non-closure; and total composition. The release runs an axiom audit and rejects `sorry` or `admit`. This mechanizes the stated total finite core; it does not formally verify the Rust checker or native backends.

BL2 exhaustively checks the same six round trips but receives no endpoint Safe or Match. BL4 receives the same normalized $(A, \chi, U, \mathsf{Safe}, \mathsf{Match})$, evaluator, totality, validity, coverage, and witness order as GlueRift. Therefore BL4 must produce the same common verdicts and first policy/property witness. This is intentional: a complete direct relation checker is logically sufficient, so BL4 is a semantic control realization rather than a performance baseline. The current release shares A02's semantic witness path and parity-checks C01's derivation with BL4. Its observed exclusive evidence is generated-twist provenance, context-bound carrier diagnostics, and native binding—not greater logical expressiveness.

6 Evaluation

We evaluate six questions: whether all-six-round-trip attacks exist at the workflow's target-native comparison cut; whether two-sided checks expose vacuity; whether the closed checker gives actionable diagnostics; whether total composition works; whether BL4

Table 2: Core attack and benign matrix. RT summarizes all six laws; S/A/P/F are soundness, adequacy, precision, and faithfulness.

Case	Shape	RT	S	A	P	F	Class/cert.
A01.base	enum aligned	P	P	P	P	P	cert.
A01	enum swap	P	D	D	D	D	harmful
A02.base	record aligned	P	P	P	P	P	cert.
A02	field swap	P	D	D	D	D	harmful
A03.base	scalar aligned	P	P	P	P	P	cert.
A03	complement	P	D	D	D	D	harmful
A05.base	result aligned	P	P	P	P	P	cert.
A05	result swap	P	D	D	D	D	harmful
H01	constructor rename	P	P	P	P	P	cert.
H02	correct reorder	P	P	P	P	P	cert.
H04.tna	conservative target	P	P	P	P	P	cert.
H04.exact	same candidate	P	D	P	D	D	none

parity holds; and whether native processes reproduce the reference relation. All results below are categorical exhaustive checks, not sampled measurements. P means proved exhaustive and D means disproved by a canonical witness.

6.1 Protocol and fixed oracles

The evaluation registry and categorical expected matrix were fixed by the artifact contract before implementation. It contains four aligned base/attack pairs, four benign modes, comparator and vacuity regressions, three asymmetric-transformation runs plus one law-breaking run, two composition modes, and two native bindings: 24 canonical checker runs in total. A test failure cannot be repaired by changing its expected category. Each attack base and candidate use identical types, comparator, U, policy, scope, requested laws, and profile; the only semantic change is the evidence-bound conjugation.

The checker enumerates values, pairs, dimensions, and witnesses in a canonical order. The canonical result owner mechanically generates every categorical result cell in Tables 2, 3, 4, and 6; the manuscript inputs those generated fragments directly. BL4 invokes the same evaluator and witness order, rather than an independent implementation that could diverge accidentally. Native replay reports bind the exact A01 or A02 candidate hash and are accepted only after exhaustive backend truth-table conformance. We ran the entire release procedure from a clean external output root and regenerated it for byte comparison against the checked release.

6.2 Finite attacks and benign adapters

Table 2 covers four distinct IR-level attack families. Each aligned base context is defined, passes all six laws, and is faithful. Each twisted candidate is mechanically bound to that base, remains defined, passes all six laws, and is classified lawful-harmful. BL2 accepts every attack because its entire criterion is round-trip success. Both GlueRift and BL4 disprove the four comparison properties and return the same first witness.

The benign rows guard against an attack-only checker. H01 accepts different constructor names with aligned roles; H02 accepts a physical field reorder with correct logical roles. H04 is intentionally asymmetric: a conservative target is faithful under non-amplification, but under exact semantics it remains adequate while soundness, precision, and faithfulness fail. This confirms that the relations measure different obligations rather than aliases for one verdict.

Table 3: Regression outcomes.

Case	Outcome	Defect isolated
V01.carrier	RT P, sound P	$E^C = \varnothing$
V01.target	RT P, sound D	E^T unsafe; bridge D
V02	invalid	empty Match coverage
V06	unknown	unsupported observer
V10	S/A P; P/F D	extra safe equality
Policy vacuity	no certificate	Safe = U warning
T01.σ_1	lawful-safe	asymmetric member
T01.σ_2	lawful-safe	asymmetric member
T01.composite	lawful-harmful	policy-only non-closure
T02.σ	law-breaking	sound but carrier RT D
C01	derivation P	total exact/preorder only

6.3 Comparator and vacuity regressions

Table 3 isolates common specification and classification errors. V01 proves all six laws in both runs. Carrier equality finds no induced pairs and is sound; target-native equality finds two pairs and exposes an unsafe one. Its carrier-to-target bridge is disproved, so carrier evidence is explanatory only. V02 rejects an empty required-match relation under nonempty coverage. V10 is sound and adequate but not precise or faithful, demonstrating extra safe equality. The policy-vacuity run satisfies the raw inclusion but is marked policy-unconstrained and receives no certificate.

T01 confirms that asymmetric safe transformations are not closed under composition: both generators are lawful-safe, while their composition still passes every requested law and becomes harmful only under the policy. T02 confirms the three-way partition: it is policy-sound but fails a requested target-carrier law and is therefore never called lawful-safe. Structural enumeration automatically generates A01, A02, and A05, including A02's nested path; A03 is explicitly recorded as an admitted declared scalar candidate.

6.4 Native target-native false agreement

We implement architecture-path-faithful native fixtures using a shared Protobuf schema, a Go source process, and a Rust target process. The processes execute with an empty-plus-whitelist environment, bound inputs and executables, read-only source enforcement, and network isolation. The native harness reads a content-addressed bundle emitted by the checker as its only semantic authority; the four map tables, complete target-native relation, six staged law tables, and canonical witness have zero mismatches.

Table 4 shows the decisive executions. E01's source returns `DENY`; the target returns `Permitted`; and the twisted output adapter transports the source result to `Permitted`. The target-native comparator therefore returns `EQUAL`. E02 is non-enum and nested: the source returns bounds $\{minimum : 0, maximum : 2\}$, while target and transported source both return $\{minimum : 2, maximum : 0\}$. Its witness identifies `output.policy.bounds.minimum`, expected role `minimum`, and transported role `source.maximum`. All six declared round-trip laws pass in both fixtures, and policy soundness is disproved.

Table 5 separates the topology we reproduce from features we deliberately omit. In particular, E02 uses finite integer fields rather than opaque library wrappers, so these runs support an operational

Table 4: Separate-process native replays.

Case	Source	Target	Transported	Result
E01	DENY	Permitted	Permitted	EQUAL
E02	min 0,max 2	min 2,max 0	min 2,max 0	EQUAL

Table 5: Motivating workflow versus native reconstruction.

Property	Motivating workflow	Reconstruction
Shared Protobuf carrier	yes	yes
Source output transported to target type	yes	yes
Final target-native equality	yes	yes
Separate Go/Rust execution	yes	yes
Adapter synthesis	LLM-generated	fixed finite IR
Opaque external-library values	central	not modeled
Round-trip coverage	sampled Go cycles	exhaustive six laws

Table 6: Observed evidence on A02; BL4 is the Direct-Relation control.

Output	BL4	GlueRift
Policy-soundness verdict	disproved	disproved
First endpoint witness	same	same
Nested semantic witness path	same	same
Generated harmful twist	no	enumerated
Normalized inverse/action domain	no	yes, $\mathsf{All}(C)$
Mechanical four-map conjugation	no	checked
Context-bound carrier bridge	no	checked
Native candidate binding	no	E02 bound

proof witness for the comparison path, not broad practicality for library adapters.

These executions establish the operational claim for our reconstruction, not prevalence and not a vulnerability in a named implementation. They also show why the comparator index matters: the observed equality is target-native, not an assumed carrier equality.

6.5 Baseline and reproducibility results

BL2 reports all six laws proved for every lawful attack and therefore admits them at the round-trip layer. BL4 is paired with every required attack, benign, composition, and regression run. Across all paired runs, common validity, coverage, policy, TNA, and four property verdicts match GlueRift, as does the first shared witness. Table 6 reports the concrete A02 evidence difference. It also records that the nested semantic witness path is shared, avoiding an advantage claim from a deliberately degraded baseline.

Starting from the source-only package, run these commands in order:

```
./artifact/bootstrap-tools
./artifact/reproduce
```

The first provisions checksum/version-pinned tools and caches; the second auto-selects source-only generation or checked-release verification, rebuilds every proof, checker fixture, baseline, and native binary, validates the acyclic evidence graph, and regenerates both the result owner and the TeX fragments input above. Verification mode additionally byte-compares the complete regenerated graph. The Lean audit covers L1–L9 without placeholders. Table 7 shows how the principal paper claims connect to mechanization and executable evidence.

Table 7: Claim-to-evidence map.

Paper claim	Lean	Checker evidence	Native
RT preservation	L2/L3	A01/02/03/05 six-law reports	E01/E02
Direct target equality	L4	A01/02/03/05 target reports	E01/E02
Comparator divergence	L5/L6	V01	–
Vacuity and extra equality	L5	V01.carrier, V02, policy-vacuity, V10	–
Lawful classification	L8	T01/T02	–
Total composition	L9	C01	–

7 Discussion and Limitations

What the result says. Round trips establish that conversions cancel along specified cycles. They do not identify which endpoint meanings carrier positions should represent, nor do they establish security of a different comparison cut through the cycle. Independently owned endpoint relations break that ambiguity. Soundness excludes unsafe equality; adequacy prevents the defense from succeeding by making comparison useless.

Trusted base. The policy owner must correctly declare finite domains, U, Safe, Match, observers, comparator, and requested laws. GlueRift checks well-formedness, coverage, supported observer syntax, and finite relation properties, but it does not infer endpoint meaning. The Rust evaluator, Lean compiler/kernel, native toolchains, Protobuf runtime, and canonicalization/evidence scripts are also trusted according to their roles. The Lean theorems reduce confidence in the formal spine; they do not verify the checker implementation.

Scope. Completeness is only for finite declared values, the closed adapter and observer IR, and the registered transformation family. Scalar templates are admitted candidates, not exhaustively discovered. Arbitrary callbacks, general list semantics, unbounded integers, effectful composition, and arbitrary Go/Rust adapters are unsupported. The checker reports unknown rather than approximating a missing observer. A passing result says only that the selected properties hold for the selected comparator and declared scope; it is not whole-program equivalence.

External validity. We study one recent architecture and two adapter-path-faithful native reconstructions. We neither measure prevalence nor search for natural vulnerabilities. The contribution survives that limitation because the claim is an implication counterexample: ordinary round-trip admission does not entail target-native policy soundness. Broader studies may establish how often this seam occurs, but are not evidence needed for the theorem.

Disclosure and ethics. The experiments do not target a deployed service, use credentials, or execute the motivating preprint's public implementation. They expose an architecture-level design weakness through a reconstruction. Because no public implementation was executed or named as affected, we classify the result as a counterexample to an abstract validation architecture, not an implementation vulnerability, and make no implementation-specific disclosure claim.

8 Related Work

Translation validation and interoperability glue. The motivating 2026 preprint synthesizes Protobuf-backed adapters, checks Go-side and full Go/Rust round trips, and reuses the adapters for I/O comparison on observed tests [11]. We attack that admission seam in a reconstruction; we do not dispute its broader translation results. Differential translation validators such as FLOURINE and RustAssure compare source and translated behavior through concrete fuzzing or symbolic execution [2, 4]. Our target is orthogonal: the adapter-mediated relation needed for such a comparison can itself turn a program disagreement into equality while satisfying its admission laws. GlueTest uses language interoperability to run source-language tests against translated code and explicitly notes that the interoperability layer can introduce inconsistencies [1]. Patterson et al. instead start from declared type convertibility and a semantic relation, then prove target-level glue sound relative to it [9]. That defense principle subsumes our top-level direct relation check. Our delta is the round-trip-preserving target-native attack, two-sided vacuity diagnostics, generated finite twists, and bound native replay.

Lenses and bidirectional programming. Classical lenses establish round-trip laws and compositional bidirectional transformations [5]. Symmetric lenses model consistency directly between two domains [7]; quotient lenses state well-behavedness modulo equivalence [6]; contract lenses attach predicates that support safe partial composition [12]. Monadic formulations study effectful composition [10], which our total Core intentionally does not claim. Optics and type equivalences connect reversible programs and canonical equivalences [3]. These works make explicit that semantic consistency and composition are not new contributions of GlueRift. BL4 operationalizes this strongest prior principle and is required to match our logical verdicts.

Cycle consistency and automorphisms. Moriakov et al. show that exact CycleGAN solutions are invariant under automorphisms and characterize the solution space as a principal homogeneous space [8]. Our conjugation theorem has the same algebraic shape. We do not claim the twist theorem as a new abstract result. We instantiate it at a cross-language program comparison boundary, define harm relative to endpoint-owned security semantics, and retain only transformations that remain applicable and lawful for the validator's explicit request.

9 Conclusion

Adapter round trips can all be correct while the comparison they enable is wrong. A carrier twist and its inverse cancel inside native, carrier, and full transport cycles, yet alter the target-native graph used to declare program agreement. GlueRift makes that graph explicit and checks it against both a safety relation and a required-match relation, thereby excluding unsafe false agreement without accepting vacuous non-comparison. Lean proofs, four finite attack families, a fair direct-relation baseline, and two Go/Rust/Protobuf replays support this bounded claim. The practical lesson is simple: round-trip admission is not semantic alignment evidence for a downstream comparator; that alignment requires independently owned endpoint obligations.

References

[1] Muhammad Salman Abid, Mrigank Pawagi, Sugam Adhikari, Xuyan Cheng, Ryed Badr, Md Wahiduzzaman, Vedant Rathi, Ronghui Qi, Choiyin Li, Lu Liu, Rohit Sai Naidu, Licheng Lin, Que Liu, Asif Zubayer Palak, Mehzabin Haque, Xinyu Chen, Darko Marinov, and Saikat Dutta. 2024. GlueTest: Testing Code Translation via Language Interoperability. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 612–617. doi:10.1109/ICSME58944.2024.00061

[2] Yubo Bai and Tapti Palit. 2025. RustAssure: Differential Symbolic Testing for LLM-Transpiled C-to-Rust Code. In *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE '25)*. IEEE. doi:10.1109/ASE63991.2025.00051

[3] Jacques Carette and Amr Sabry. 2019. Optics and Type Equivalences. In *Proceedings of the Eighth International Workshop on Bidirectional Transformations (Bx 2019)*. Workshop paper.

[4] Hasan Ferit Eniser, Hanliang Zhang, Cristina David, Meng Wang, Maria Christakis, Brandon Paulsen, Joey Dodds, and Daniel Kroening. 2024. Towards Translating Real-World Code with LLMs: A Study of Translating to Rust. arXiv:2405.11514 [cs.SE] doi:10.48550/arXiv.2405.11514 Preprint.

[5] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. 2007. Combinators for Bidirectional Tree Transformations: A Linguistic Approach to the View-Update Problem. *ACM Transactions on Programming Languages and Systems* 29, 3, Article 17 (2007). doi:10.1145/1232420.1232424

[6] J. Nathan Foster, Alexandre Pilkiewicz, and Benjamin C. Pierce. 2008. Quotient Lenses. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming (ICFP '08)*. ACM, 383–396. doi:10.1145/1411204.1411257

[7] Martin Hofmann, Benjamin C. Pierce, and Daniel Wagner. 2011. Symmetric Lenses. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '11)*. ACM, 371–384. doi:10.1145/1925844.1926428

[8] Nikita Moriakov, Jonas Adler, and Jonas Teuwen. 2020. Kernel of CycleGAN as a Principal Homogeneous Space. In *International Conference on Learning Representations (ICLR)*. arXiv:2001.09061

[9] Daniel Patterson, Noble Mushtak, Andrew Wagner, and Amal Ahmed. 2022. Semantic Soundness for Language Interoperability. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '22)*. ACM, 609–624. doi:10.1145/3519939.3523703

[10] Li yao Xia, Dominic Orchard, and Meng Wang. 2019. Composing Bidirectional Programs Monadically. In *Programming Languages and Systems*. Lecture Notes in Computer Science, Vol. 11423. Springer, 147–175. doi:10.1007/978-3-030-17184-1_6

[11] Hanliang Zhang, Arindam Sharma, Cristina David, Meng Wang, Brandon Paulsen, Daniel Kroening, Wenjia Ye, and Taro Sekiyama. 2026. Validated Code Translation for Projects with External Libraries. arXiv:2602.18534 [cs.SE] doi:10.48550/arXiv.2602.18534 Preprint.

[12] Hanliang Zhang, Wenhao Tang, Ruifeng Xie, Meng Wang, and Zhenjiang Hu. 2023. Contract Lenses: Reasoning about Bidirectional Programs via Calculation. *Journal of Functional Programming* 33 (2023), e10. doi:10.1017/S0956796823000059