International Journal of Innovative Research in Engineering
Volume 7, Issue 4 (July - August 2026), PP: 160-170.
https://www.doi.org/10.59256/ijire.20260704018
www.theijire.com
ISSN No: 2582-8746

Three Measurement Hazards in Analyzer-in-the-Loop Repair of Smart Contracts

Ian Staley
Independent Researcher, United States.

How to cite this paper:
Ian Staley' "ThreeMeasurementHazardsin Analyzer-in-the-LoopRepairofSmart Contracts", IJIRE-V7I4-160-170.

Abstract: *Pipelines that pair a large language model with a static analyzer, feeding findings back as repair instructions, appear throughout recent smart contract repair research. They rest on a rarely examined assumption: that the analyzer output serving as the oracle faithfully records what the analyzer found. I report three ways that assumption fails, identified during a four-contract instrument-validation exercise preceding a planned repair study. First, Mythril v0.24.8 can exit without reaching the analysis phase while returning exit status zero, empty standard error, and a findings array byte-identical to that of a genuinely clean scan; the failure is reported in a sibling JSON field that finding-extraction code has no reason to read. Second, 12 of 23 Slither findings in my validation set fell outside the high, medium, and low impact bands, so an unfiltered count measures a composite whose components may not behave alike under repair. Third, keying finding identity on source location breaks across repair rounds. On the one contract carried through three rounds, location-based keying inflated resolved findings from 7 to 12 and introduced findings from 2 to 7. The underlying instability is established in the warning-tracking literature; my contribution is its consequence for repair metrics, where it biases both transition counts upward and can confound comparison between methods producing differently sized .patches. I separately report an executed exploit showing a specification-level authorization defect that produced no high or medium impact finding. I propose calibration procedures for each hazard and release the harness, contracts, and raw analyzer output at doi:10.5281/zenodo.21586404.*

Key Words: *smart contract security; static analysis; large language models; automated program repair; measurement validity; research reproducibility.*

I.INTRODUCTION

The pattern is familiar. A model generates or modifies Solidity code, a static analyzer scans the result, the findings are serialized back into a prompt, and the model is asked to fix them. The loop runs for some number of rounds and stops. Variations appear throughout the recent literature on smart contract repair [1], [2], [3], and the same shape is straightforward to embed in a continuous integration pipeline that gates on analyzer output.

The design is attractive for a well-argued reason. Research on model self-correction is split, and the split runs along a clear line. Attempts at intrinsic self-correction, where the model critiques its own work unaided, generally fail to improve and sometimes degrade output [4], [5], while approaches that supply an external signal succeed [6], [7], [8], [9]. A static analyzer is an unusually good external signal: automatic, repeatable, and producing located, typed findings rather than impressionistic ones.

What that argument takes for granted is that the analyzer output faithfully represents what the analyzer found. Measurement validity is the standard frame for this concern in empirical software engineering [10], [11], [12], and the concern is not abstract: the scientific software literature has documented for decades that computational instruments can produce wrong answers quietly [13], [14].

I did not set out to write this paper. The hazards below were found while validating a measurement harness for a planned study of repair convergence and regression, in the interval between building the instrument and collecting data. Each would have corrupted that study.

I address four questions.

- RQ1. Can an analyzer execution failure be recorded as a zero-finding result that a findings-only repair pipeline cannot distinguish from a clean contract?
- RQ2. What construct is represented by an unfiltered repair-resolution count when analyzer output spans multiple impact categories?
- RQ3. How does source-location finding identity distort cross-round repair metrics, and in what direction?
- RQ4. Can a consequential specification-level defect fall entirely outside the analyzer oracle that a repair loop is scored against?

The claims here are existence claims. My evidence base, described in Section III, is four hand-authored

contracts and one repair sequence, which cannot support any statement about how often these hazards arise. It can support the claim that each occurs, that each mechanism follows from documented tool behavior or from the arithmetic of the measurement rather than from anything peculiar to my files, and that in specified archival conditions each is difficult to detect after the fact.

I am equally explicit about what is not new. The line-number instability behind the third hazard is the founding motivation of the literature on tracking analysis warnings across versions [15], [16], and compiler-resolution failures in the analyzer behind the first are reported in its issue tracker [17], [18]. My contribution is the measurement consequence in each case, and Sections IV and VI state the boundary precisely.

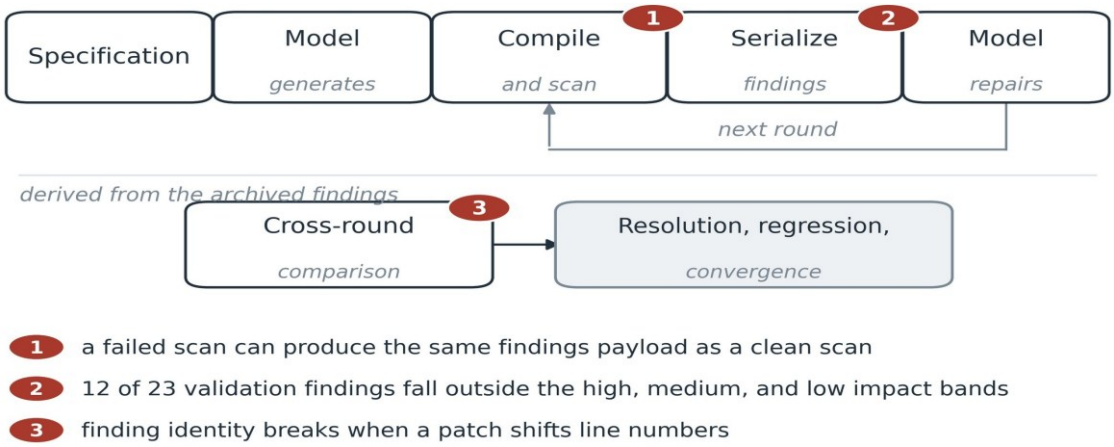

Figure 1. The analyzer-in-the-loop repair pattern, with the three hazards marked where each enters the measurement. Hazards one and two affect what a single round records; hazard three affects the comparison between rounds.

Figure 1 locates the three hazards within the repair pattern, distinguishing those that affect what a single round records from the one that affects the comparison between rounds.

The contributions are: a characterization of three measurement hazards with mechanism, trigger, and detectability for each; a calibration procedure combining positive and negative controls that converts undetectable analyzer failure into a hard stop; an executed demonstration of the gap between the analyzer oracle and a specification-level defect; and a released harness implementing the recommendations.

II.BACKGROUND

Measurement validity in empirical software engineering. Construct validity concerns whether a measure captures the concept it is meant to capture, and internal validity concerns whether observed relationships are attributable to the treatment rather than to artifacts of the instrument [10]. Community reporting standards make disclosure of measurement decisions an explicit expectation [11], and surveys of practice find the treatment of validity threats frequently perfunctory [12]. My three hazards are, respectively, an instrumentation failure, a construct-validity problem, and an internal-validity confound.

Silent failure in computational instruments. Hatton's experiments on independently implemented scientific codes found substantial disagreement between implementations of the same computation with no runtime indication of trouble [13], and the argument that computational results require disclosure of the producing code follows directly [14]. Static analysis tools are subject to the same class of problem, and implementation and configuration differences change their output materially [19].

Static analysis of Solidity. Slither lifts contracts into an intermediate representation and runs a large detector library, classifying output by impact and confidence [20]. Mythril performs symbolic execution over EVM bytecode [21]. Comparative work characterizes how these tools differ in coverage and error behavior [22], [23]. Benchmark infrastructure including SmartBugs and its successor supports reproducible execution and supplies annotated datasets [24], [25]. A recent evaluation over 788 contract files and 10,394 vulnerabilities found that existing tools fail to detect around half of the benchmark vulnerabilities and that precision does not exceed ten percent [26]. It is therefore established that these analyzers miss a great deal. Less attention has gone to what happens when their output is used as an experimental instrument rather than a development aid.

Repair of smart contracts. ContractTinker integrates static analysis into a chain-of-thought repair process [1]; ACFix mines access-control patterns to guide repair [2]; SCPatcher combines retrieval-augmented generation with a knowledge graph and uses an analyzer as a binary gate on whether new issues appeared [3]. Bobadilla and colleagues evaluated whether automated fixes mitigate exploits, finding wide variation across tools [27]. Each relies at some point on analyzer output as evidence about defects, regressions, or patch quality.

Warning identity across versions. Spacco and colleagues introduced tracking of defect warnings across versions [15], and Avgustinov and colleagues developed matching robust to source movement, observing explicitly that a purely location-based approach treats a violation as different once an edit shifts its line, and matching instead on the enclosing program element [16]. Section VI positions my claim against this work rather than around it.

Warning triage. Industrial experience establishes that warning volume must be filtered to be useful, that developers respond to perceived actionability, and that analyzers producing low-value output get disabled rather than tolerated [28], [29], [30]. My second hazard is the measurement-side consequence of the same distinction.

Repair evaluation conventions. The distinction between a plausible patch, which satisfies the available oracle, and a correct patch, which fixes the underlying defect, is long established, as is oracle overfitting [31], [32]. Section VII restates that problem for an analyzer oracle.

III.METHODS: SETTING, PROCEDURE, AND CONFIGURATION

The hazards were identified while validating a harness built for a planned study of multi-round repair. That study generates Solidity contracts from specifications in the receivables and trade-finance domain, subjects each to several rounds of analyzer-driven repair, and measures how many findings are resolved, how many appear, and where the process stops improving. The harness compiles with solc, scans with Slither and Mythril, normalizes findings into a common record, and computes the metrics.

Contract selection and authorship. Four specifications were selected from the planned twelve to span the structural range of the corpus: a true-sale receivable transfer (state transitions and role logic), a milestone escrow (value custody and external calls), a fee waterfall (arithmetic distribution and multiple transfers), and a double-financing guard (registry and hashing). I wrote all four contracts against those specifications, targeting Solidity 0.8.24, and range from 21 to 34 statements. No defects were deliberately seeded. The contracts were written to exercise the pipeline, not to represent model output, and they are neither sampled nor random.

Repair procedure. One contract, the milestone escrow, was carried through three repair rounds. In each round the current source was compiled and scanned, the resulting findings were serialized deterministically into a feedback prompt, and a revised contract was produced. Because the study's generation stage requires model access that was unavailable in the validation environment, I performed the revisions manually against the serialized findings, applying the reported issues without introducing unrelated changes. This substitution is a limitation: a model would plausibly produce different patch sizes and different regressions, and the observed magnitudes in Section VI should be read as an existence demonstration rather than an estimate of what a model would produce. All four rounds compiled successfully. The sequence stopped at three rounds by the round count fixed for the planned study, not by a convergence criterion. I note that the validation runs reported here were performed before that protocol document was finalized, so I describe the round count as pre-specified rather than pre-registered; the document is included in the deposit for transparency about what was fixed and when.

Finding normalization and metrics. Each finding is normalized to a record of analyzer, detector, impact, contract, function, line, and message. For contract c at round r with finding multiset F(c, r), findings resolved between rounds is the size of F(c, r-1) minus F(c, r), and findings introduced is the size of F(c, r) minus F(c, r-1), both as multiset differences. Section VI compares these quantities under two definitions of finding identity.

Configuration. Table 1 records the execution environment. All results in this paper were produced under it.

Component	Version or value
Slither	0.11.5
Mythril	v0.24.8
solc	0.8.24+commit.e11b9ed9.Linux.g++
solc binary SHA-256	fb03a29a…37c30d5f (full hash in the deposit)
crytic-compile	0.3.11
Python	3.12.3
Platform	Linux 6.18.5 x86_64, glibc 2.39
Slither invocation	slither <file> --json - (default detector set, none disabled)
Mythril invocation, condition A	myth analyze <file> --solv 0.8.24 -o json
Mythril invocation, condition B	myth analyze <file> -o json (identical minus the flag)
Mythril output mode	json (not jsonv2)
Compiler intended, both conditions	Solidity 0.8.24; B used the listed local binary, A terminated before compiler invocation
Trigger condition for A	solc 0.8.24 absent from the solcx cache and solc-bin.ethereum.org unresolvable
EVM semantics	solc 0.8.24 default target, Shanghai

Table 1. Execution environment and exact invocations. Conditions A and B differ only in the presence of the compiler-selection flag, giving a single-factor comparison. Reproducing A requires that the requested compiler be absent from the local solcx cache, since a cached binary satisfies the flag without a network fetch.

Use of AI tools in this study. A large language model, Anthropic Claude, was used in preparing this work, and I state where because the distinction between what a model produced and what a tool measured is exactly the kind ofprovenance this paper argues should be recorded. The model was used to draft and edit prose, to implement portions of the measurement harness and the analysis and plotting scripts, and to assist with locating and checking bibliographic records. It was not used to generate any of the contracts analyzed here, which I wrote, and it performed no part of the measurement: every finding count, exit status, and metric reported was produced by the tools and configuration in Table

1, is reproducible fromthe deposited raw output, and I verified each one. Where the model was used to write code, that code is in the deposit and its behavior is checked by the reported results rather than taken on trust. I take full responsibility for the content.

IV.RESULTS: HAZARD ONE, AN EXECUTION FAILURE RECORDED AS ZERO FINDINGS

This is an instrumentation failure: the measuring device did not run and the experiment continued.

Mythril reported zero findings on every contract in the validation set. This was initially read as substantive, since the contracts are small and Mythril searches a narrower class of properties than Slither. Suspicion arose because one contract carried a pattern that Slither reported under its reentrancy-eth detector at high impact and medium confidence, and a symbolic executor returning nothing where a static analyzer reports a high-impact reentrancy pattern is surprising. I constructed a positive control containing a reentrancy in a withdrawal function performing an external call before zeroing the balance, and an unrestricted selfdestruct callable by any address. Mythril reported zero findings on the control. Slither reported both.

The cause was not analytical. Mythril's compiler-selection flag causes it to fetch a solc binary from a remote host, and where that host is unreachable the process raises a connection error during setup, before any analysis. Removing the flag so that Mythril resolves solc locally produced four findings on the control, including the unrestricted selfdestruct at high severity.

I initially described this as the tool suppressing its traceback and emitting an empty document. That description is wrong, and correcting it sharpens the hazard. Table 2 compares three conditions: the failing run, the identical command with only the compiler-selection flag removed, and a genuinely clean contract under that same corrected invocation.

Observation	A. failed run	B. same, no flag	C. clean contract
Process exit code	0	1	0
Bytes on standard error	0	0	0
JSON field: success	false	true	true
JSON field: error	5,231-char traceback	null	null
JSON field: issues	[ ]	4 findings	[ ]
Analysis actually performed	no	yes	yes

Table 2. Three conditions on the positive control and a clean contract. A and B differ only in the compiler-selection flag. Exit status does not report whether the run succeeded: the failed run A exits 0 while the successful analysis B exits 1, because in these runs the exit code tracked the presence of findings rather than successful completion of analysis. A and C are indistinguishable on exit code, on standard error, and on the findings array alike.

Two points follow immediately. The tool does not hide the failure: the success field is false and the full traceback is present in the error field. And the findings array is byte-identical between the failed run and the clean contract, so extraction code keyed on that array cannot tell them apart; my own parser read the issues array and ignored the success field, which is why I misdiagnosed the behavior.

A third point I did not anticipate. Exit-status and standard-error checks alone do not detect the failure, and exit status here is worse than uninformative, because it is inverted. The failed run exits 0 while the successful analysis exits 1. In these runs the exit code tracked the presence of findings rather than successful completion of analysis, a convention that suits a security tool gating a build. I report this as observed behavior in the version tested rather than as a documented contract of the interface. A harness treating exit 0 as success would therefore accept the run that analyzed nothing and flag the run that worked. I report this because exit status is the first additional check a reader is likely to propose, and adopting it naively in this configuration would make matters worse rather than better.

The hazard is therefore not tool misbehavior but a mismatch between where a tool reports failure and where a measurement pipeline looks. That reframing makes it more general rather than less: any analyzer whose structured output separates status from payload is exposed, and the exposure grows with the degree to which the consuming code is written against the payload alone.

Detectability. The failure is recoverable from the raw JSON, which retains `success` and `error`. It is not recoverable from normalized finding records, from aggregate repair metrics, or from process-level signals. A study depositing only normalized findings would archive a failed run as a clean contract with no residue. This is why my mitigation pairs a calibration control with schema validation, and why I recommend depositing raw analyzer output rather than only normalized records.

Scope. I demonstrate this for one analyzer at one version under one trigger. I have not surveyed other tools, other versions, or the jsonv2 output mode, and the behavior may differ in any of them. References [17] and [18] report compiler-resolution failures in this tool; both describe visible errors in interactive use and neither, so far as I can determine, characterizes the structured-output case.

V.RESULTS: HAZARD TWO, IMPACT COMPOSITION AND WHAT A FINDING COUNT MEASURES

This is a construct-validity problem: the measure may not capture the concept it is reported as capturing.

Across the validation contracts, Slither emitted 23 findings under its default configuration. Eleven fell in the high, medium, or low impact bands; the remaining twelve, slightly over half, were classified as optimization or informational. Table 3 and Figure 2 give the composition.

Contract	High	Med.	Low	Optim.	Info.	Total
SPEC-02 true-sale	0	0	1	1	0	2
SPEC-04 escrow	2	0	4	2	1	9
SPEC-06 waterfall	0	0	4	7	0	11
SPEC-08 double-financing	0	0	0	1	0	1
TOTAL	**2**	**0**	**9**	**11**	**1**	**23**

Table 3. Impact composition of Slither findings across the four validation contracts. The optimization findings are dominated by immutable-variable recommendations and the informational finding is a low-level-call report.

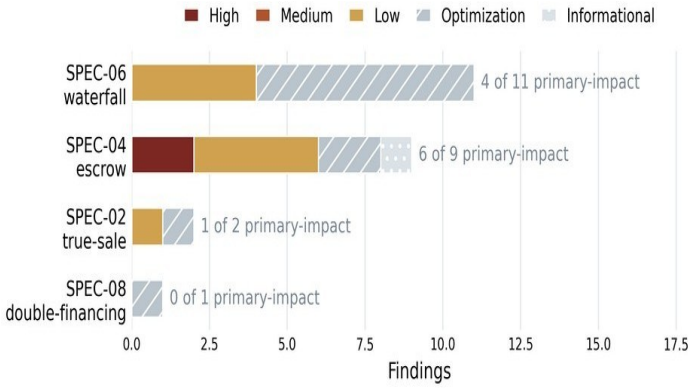

Figure 2. Impact composition per contract, ordered by total findings, with the primary-impact share annotated. Two of four contracts carry no finding above low impact.

Terminology matters here and I was previously imprecise. Slither classifies detector output by impact and confidence, and the optimization and informational categories are not synonymous with cosmetic. A low-level-call report is classified informational yet concerns behavior with direct security consequences. I therefore distinguish the primary impact bands, meaning high, medium, and low, from the non-primary bands, and avoid characterizing the latter as style issues.

The measurement concern is that an unfiltered finding count aggregates categories that may not behave alike under repair. Optimization findings such as immutable-variable recommendations admit localized mechanical edits, whereas findings requiring semantic change to authorization or control flow do not, which gives reason to expect the categories to differ in repair difficulty. If they do differ, an unfiltered resolution rate is a composite whose value depends on corpus composition, a quantity that may vary across corpora and is not always reported.

I state clearly that my evidence does not establish that difference, and that the one measurement available to me runs the other way. Stratifying the escrow repair sequence by band, primary-impact findings resolved at 5 of 6, or 0.83, and non-primary findings at 2 of 3, or 0.67. The counts are far too small to support any inference, and the resolved non-primary findings were immutable-variable recommendations while the persisting one was a low-level-call report, so the split may reflect which specific detectors were present rather than any property of the bands. I report it because it is the only relevant data I have and because it does not support the intuition. The claim that the categories differ in repair difficulty remains a plausible mechanism awaiting an impact-stratified study on a corpus large enough to estimate it. The recommendation is therefore procedural rather than substantive. Pre-register which impact bands enter the primary measure, report composition across all bands, and treat corpus composition as a reported characteristic rather than an incidental detail. I adopt the primary bands for the planned study and note that under that filter two of four validation contracts have no findings at all. The resulting sparsity should be treated explicitly in any later statistical model.

VI.RESULTS: HAZARD THREE, FINDING IDENTITY ACROSS REPAIR ROUNDS

This is an internal-validity confound, and the hazard where the boundary of my contribution needs the most care.

Measuring repair across rounds requires deciding when a finding at round two is the same finding as one at round one. The natural choice keys on the tool, the detector, and the source location, since that is what the tool reports and it identifies a finding uniquely within a single scan.

It does not survive edits. A patch that inserts or removes a line shifts the location of every finding below it. Under a location-based key an unchanged finding appears at one location in the earlier round and another in the later

one; the set difference records it as resolved, because the old key is absent, and as newly introduced, because the new key is present.

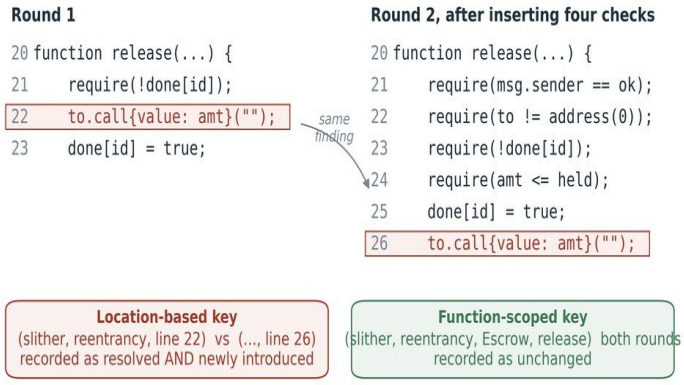

Figure 3. Why location-based identity double-counts. Four inserted checks push an unchanged finding from line 22 to line 26. The location key differs across rounds; the function-scoped key does not.

Figure 3 illustrates the mechanism on a concrete edit.

This instability is not my discovery. It is the founding motivation of the literature on tracking analysis warnings across versions. Spacco and colleagues posed the tracking problem two decades ago [15], and Avgustinov and colleagues developed matching robust to source movement, noting explicitly that a purely location-based approach would treat a violation as different once an edit shifts its line, and matching on the enclosing program element instead [16]. My mitigation is substantially their approach, and a reader familiar with that work should not credit me with rediscovering it.

My claim concerns consequence in a setting that work did not address. In repair measurement, the quantities corrupted by location-based identity are the two the study exists to report. Table 4 quantifies this on the escrow sequence, scoring the same three transitions twice with no difference except the identity key.

Round transition	Resolved, function	Introduced, function	Resolved, location	Introduced, location
0 to 1	7	1	9	3
1 to 2	0	1	3	4
2 to 3	0	0	0	0
TOTAL	**7**	**2**	**12**	**7**

Table 4. The same repair sequence scored under two identity schemes. Location-based keying raises resolved findings from 7 to 12 and introduced findings from 2 to 7, with no change to the contracts or the analyzer output.

The direction of the bias requires precision, and my earlier characterization of it was wrong. Both transition counts are biased upward. Higher resolution flatters the repair process and higher regression penalizes it, so the effect on any composite assessment depends on how the two are weighted. The defensible claim is narrower and still serious: location-based identity positively biases both transition counts, exaggerating apparent repair activity, and because the magnitude depends on the number and placement of inserted or removed lines it is expected to vary with patch size. Since patch size is a property of the treatment rather than an independent nuisance variable, this confounds comparisons between repair methods that produce patches of different size. I did not measure the patch-size relationship, which would require a corpus with varying edit magnitudes, and I state it as a mechanism rather than a result.

My mitigation follows the established approach. I key findings on tool, detector, contract, and enclosing function, discarding the line number from identity while retaining it as prompt metadata, and add multiset semantics so that two firings of the same detector in one function count as two and a reduction to one is recorded as a single resolution. This preserves multiplicity that set-based matching discards, which matters because resolution here is a count rather than a per-warning binary state.

The limitations are broader than a single undercount, and I enumerate them because the scheme should not be mistaken for semantic identity. Function-scoped matching fails or degrades when a function is renamed, when a finding moves between functions, when a function is split or merged, when inheritance changes the enclosing element, when findings are scoped to a contract, variable, or modifier rather than a function, and when detector names or output schemas change between analyzer versions.

The most consequential limitation is substitution blindness. If one finding of a given detector class in a function is fixed while a distinct finding of the same class appears in the same function, the multiset count is unchanged and the method records no transition at all. The scheme is therefore biased toward under-reporting churn in exactly the cases where a repair has replaced one defect with another of the same kind. This is the mirror image of the location-

based failure and it is not eliminated by my mitigation, only traded for.

For studies where these cases are likely to be common, stronger matching is available: syntactic context hashing, detector plus normalized source snippet, AST-node fingerprints, or the established warning-tracking algorithms. A sensitivity analysis across two or more schemes would be preferable to committing to one, and I recommend it. My scheme is the minimum that removes the location confound, not the best available matching.

A related implementation defect is worth recording because it is easy to reproduce and independent of the identity scheme. My first parser resolved the contract and function for each finding by reading the immediately reported element. For findings scoped to a state variable rather than a function this placed the enclosing function name in the contract field and left the function field unresolved, assigning incorrect keys to a subset of findings. In my Slither parser, correct resolution required walking the reported element's parent chain; the specifics are particular to that analyzer's output schema. The defect produced plausible output and was found only by inspecting keys directly.

VII.RESULTS: A DEFECT OUTSIDE THE ORACLE

The three hazards above concern the fidelity of analyzer output as a record of what the analyzer found. This section concerns the relationship between what the analyzer finds and what is wrong, which is a different and more familiar problem. I report one case because its form is stark and because I can demonstrate it rather than assert it.

The true-sale transfer specification requires that only the current owner of a receivable, or an address they have authorized, may change its ownership. The contract implements registration, sale, and reclaim, and neither the sale nor the reclaim function performs any authorization check.

I verified the consequence by execution rather than inspection. Compiling the contract with solc 0.8.24 and deploying it to a local EVM, an account that had never interacted with a registered receivable called the sale function naming itself as recipient; the ownership record transferred to that account and the irrevocable true-sale flag was set. The same unauthorized account then called reclaim and reverted ownership to the originator, stripping the recorded funder. The test is deterministic and included in the deposit.

Slither reported no high or medium impact finding on this contract. The findings emitted were a timestamp-dependence report and an immutable-variable recommendation. Mythril, once configured correctly, reported nothing. I am careful about what the exploit establishes. The contract custodies no ether and no token; what the unauthorized caller reassigns is the owner field of a storage record. Whether that constitutes economic loss depends on what the record entitles its holder to, which this contract does not implement, so I describe the defect as an unauthorized ownership reassignment and an unambiguous violation of the specification's authorization invariant, and avoid economic severity language the executable behavior does not support.

That analyzers miss defects at scale is established [22], [23], [26], and I claim no novelty for the general observation. One nuance cuts against a natural reading of my case: the most thorough recent evaluation found access control among the categories these tools handle relatively better [26]. This is therefore not a systematically hard category defeating the tools. It is the plain absence of a check the specification requires, with no anomalous construct, no dangerous call, and no known-bad idiom for a detector to match. The contract is unremarkable except that it does not do what it was supposed to do. Detecting this requires knowing the intended authorization policy, which is why dedicated access-control inference exists as a separate research thread [33].

The relevance to repair measurement is direct. A repair loop driven by these findings would have addressed timestamp dependence and variable immutability on a contract that reassigns ownership to any caller, and would have reported a high resolution rate for doing so. Any resolution or convergence measure built on analyzer output describes movement within the set of findings the tools can see, and this study does not establish the relationship between that set and the set of defects that matter.

VIII.DISCUSSION: CALIBRATION PROCEDURE AND RELEASED HARNESS

I frame the mitigations as instrument calibration rather than as a single check, since the first hazard shows that a threshold on output volume is not by itself informative.

Positive control. Before each collection run, scan a contract containing known defects and require, for each analyzer: successful compilation; an exit status interpreted according to the tool's documented semantics, rather than assuming that zero means successful analysis; a response conforming to the expected schema with non-null analysis metadata and, where the format provides a status field, a success indication; zero unhandled execution errors; and detection of at least one named detector or vulnerability class, specified by identifier rather than by count. Record tool and compiler hashes with the result. A count threshold alone is insufficient because a partially executing tool, or one whose detector set has changed between versions, can exceed a threshold on unrelated findings.

Negative control. Scan a contract believed free of the target defect classes and require that the named detectors do not fire. This guards the opposite failure, a configuration that emits findings indiscriminately, which a positive control alone cannot distinguish from correct operation. Negativecontrols are harder to construct than positive ones, because whether a detector fires can depend on code shape rather than on the presence of a defect; the control should therefore be minimal, versioned, and either independently reviewed or drawn from an established annotated benchmark.

Placement. Run both controls after environment initialization and, for long collection jobs, periodically or once per batch, since a tool can begin failing partway through a run for the same environmental reasons that cause it

to fail at the start. Abort or quarantine on failure rather than continuing.

Archival. Deposit raw analyzer output, not only normalized findings. The first hazard is recoverable from raw JSON and unrecoverable from normalized records, so the archival layer determines whether an independent party can audit for it.

On the positive control itself, one correction. I previously described an unprotected selfdestruct as a textbook contract-deletion defect. Since EIP-6780, SELFDESTRUCT generally transfers the account balance without deleting code or storage unless invoked in the same transaction as contract creation, so its consequences depend on the active fork. It remains deprecated and both analyzers still flag it, so it functions adequately as a detector control, but it should be described as an unrestricted balance-transferring selfdestruct rather than as contract deletion. Practitioners who prefer a less fork-sensitive control may substitute unrestricted ownership change, arbitrary delegatecall, or an intentionally detectable reentrancy.

Provenance. The positive control is an adaptation, not an invention. Annotated benchmark datasets of contracts with tagged vulnerabilities already exist for evaluating detection performance [24], [25]; I repurpose the idea from measuring what a tool can find to verifying that it ran. Industrial platforms already disable analyzers on evidence of misbehavior [28]; I apply that logic at data collection rather than at developer experience. What I add is the placement as a precondition whose failure halts collection, the pairing with a negative control, and the observation that this defends against a failure mode otherwise undetectable from normalized records.

Released materials. I release the harness incorporating these checks, together with the four contracts, the positive and negative controls, the specifications, the prompt templates, the normalization and scoring code, the exploit test from Section VII, and the raw analyzer output for every observation, including the failing Mythril run alongside the corrected one. Generation and analysis are separated so the analysis reproduces from archived output without model access, which is a requirement for artifact reuse [34] and also the property that lets a third party audit for the first hazard in deposited data. The deposit

[35] includes a standalone script that regenerates Table 4 from the archived Slither output with no analyzer, compiler, or model installed. Reporting practice for studies involving language models is an active area [36], and I suggest the disclosures recommended here belong alongside the model and prompt disclosures proposed there.

Table 5 summarizes the three hazards against the dimensions a reader needs in order to judge whether their own pipeline is exposed.

	Hazard 1	**Hazard 2**	**Hazard 3**
Validity type	Instrumentation	Construct	Internal
Mechanism	Failure reported outside the payload read	Aggregation across unlike categories	Line shift breaks location key
Trigger	Unreachable compiler host	Default detector configuration	Any patch inserting or removing lines
Observed	Exit 0, empty stderr, issues identical	12 of 23 findings non-primary	7 to 12 resolved, 2 to 7 introduced
Visible in raw output	Yes, success field	Yes, impact labels	Yes, successive locations
Visible in normalized records	No	Depends on retained fields	Only if both rounds retained
Visible in aggregate metrics	No	No	No
Bias direction	Toward apparent cleanliness	Unknown, depends on composition	Both counts biased upward
Mitigation	Positive and negative controls	Pre-specified impact filter	Function-scoped multiset identity
Residual limitation	One tool, one version tested	Difference in difficulty untested	Substitution blindness

Table 5. The three hazards compared. Visibility rows state the archival layer at which each becomes detectable, which determines what a deposit must contain for an independent party to audit it.

IX.THREATS TO VALIDITY

Construct validity. I treat observed instances as evidence of hazard existence. A single reproducible demonstration establishes that a failure mode occurs; it establishes nothing about severity or prevalence, and I make no such claim. The Section VII defect characterization rests on an executed test against a stated invariant, which removes the reliance on my own inspection but not the fact that I wrote both the specification and the contract.

Internal validity. The first hazard was triggered by a specific network condition interacting with a specific flag in a specific version. I argue the exposure generalizes because it follows from the separation of status and payload in structured output, but that is reasoning from one case rather than a survey. The repair sequence in Section VI was produced by me rather than a model, so the patch sizes, and hence the magnitude of the location-based inflation, are not what a model would necessarily produce.

External validity. Four hand-authored contracts, two analyzers, one language, one domain, one repair sequence. The identity hazard follows from the arithmetic of multiset difference and should transfer to any comparable setup. The

composition hazard depends on detector configuration and corpus, and my 12-of-23 figure characterizes my validation set, not Slither in general. The first hazard is specific to tools with the relevant output structure.

Conclusion validity. No statistical inference is drawn anywhere in this paper, and none would be supportable from this evidence base. The impact-stratified figures in Section V involve single-digit counts and are reported as a counter-observation to an intuition, not as an estimate.

Detectability claims. Statements that a hazard leaves no trace are relative to an archival layer, and I specify it in each case. The first hazard is undetectable from normalized finding records and from process-level signals but is recoverable from raw analyzer JSON. The second is visible in raw output, which retains impact labels and detector identities, and is a concern about interpretation rather than about missing information. The third is reconstructible from archived successive scans, since source locations are retained, but is invisible in the aggregate repair metrics that studies typically report. None of the three is undetectable from a complete reproducibility package, which is the argument for depositing one.

Novelty. Two of the three hazards rest on phenomena documented elsewhere [15], [16], [17], [18]. Readers should evaluate the contribution as the measurement consequences, the detectability analysis, and the calibration procedure, not as the discovery of the mechanisms.

X.CONCLUSION

Analyzer-in-the-loop repair is a sound design resting on a measurement assumption that deserves examination. Three mechanisms make the assumption unsafe. A tool process can exit successfully without reaching the analysis phase, reporting the failure in a field that finding-extraction code does not read, while standard error stays empty and exit status reports something else entirely. A default configuration can populate half the findings with categories whose behavior under repair may differ from findings in the high, medium, and low impact bands, making an unfiltered count a composite of unknown mixture. And location-based finding identity registers unchanged findings as both resolved and introduced whenever a patch shifts lines, biasing both transition counts upward in a manner that can confound comparisons between methods producing differently sized patches.

None of these requires an adversary, and two rest on phenomena already documented in adjacent literature. Their occurrence during a routine harness validation shows that they arise under ordinary, non-adversarial development conditions. It does not establish how often they arise, and the present evidence cannot.

The recommendation is procedural. Calibrate the instrument before collecting with it, using controls thatcheck status and named detectors rather than output volume, and archive raw analyzer output so that an independent party can verify the instrument was working. For studies using analyzer output as an oracle, I suggest reporting the impact filter applied, the finding identity rule, and the evidence that each analyzer executed. These are three additions to a methods section, and without them a reader cannot tell whether a reported resolution rate measures repair or measures the instrument.

Declarations

Funding. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Author contributions. I am solely responsible for the study design, implementation, analysis, and manuscript.

Ethics. Not applicable. This study involved no human participants and no personal data.

CRediT author statement. Ian Staley: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review and editing, Visualization.

Data Availability Statement

The harness, the four validation contracts, the positive and negative controls, the specifications, the prompt templates, the normalization and scoring code, the Section VII exploit test, and the complete raw analyzer output for every observation reported here, including the failing and corrected Mythril runs, are deposited in a public Zenodo record [35], DOI 10.5281/zenodo.21586404. Code is released under the MIT license and data, meaning the contracts, raw analyzer output, and specifications, under CC BY 4.0. The deposit is structured so that the analysis path executes from archived output without model access, and includes a script that reproduces Table 4 from the archived Slither output alone.

Declaration Of Generative Ai And Ai-Assisted Technologies In The Manuscript Preparation Process

During the preparation of this work I used Anthropic Claude in order to draft and edit prose, to implement portions of the measurement harness and analysis scripts, and to assist with bibliographic verification. After using this tool I reviewed and edited the content as needed and take full responsibility for the content of the published article. All tool executions, measurements, and numerical results reported here were produced by the software and configuration described in Table 1, and I verified each of them independently.

Conflicts Of Interest

I declare no conflicts of interest.

References

1. C. Wang, J. Zhang, J. Gao, L. Xia, Z. Guan, and Z. Chen, "ContractTinker: LLM-empowered vulnerability repair for real-world smart contracts," in *Proc. 39th IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, Sacramento, CA, USA, 2024, pp. 2350–2353, doi: 10.1145/3691620.3695349.
2. L. Zhang et al., "ACFix: Guiding LLMs with mined common RBAC practices for context-aware repair of access control vulnerabilities in smart contracts," *IEEE Trans. Softw. Eng.*, vol. 51, no. 9, pp. 2512–2532, Sep. 2025, doi: 10.1109/TSE.2025.3590108.
3. X. Li, S. Ye, W. Li, and Z. Li, "SCPatcher: Automated smart contract code repair via retrieval-augmented generation and knowledge graph," in *Proc. 34th ACM Int. Conf. Foundations of Software Engineering (FSE Companion)*, 2026, pp. 1212–1216, doi: 10.1145/3803437.3805558.
4. J. Huang et al., "Large language models cannot self-correct reasoning yet," in *Proc. 12th Int. Conf. Learning Representations (ICLR)*, Vienna, Austria, 2024. [Online]. Available: https://openreview.net/forum? id=IkmD3fKBPQ (accessed Jul. 25, 2026).
5. T. X. Olausson, J. P. Inala, C. Wang, J. Gao, and A. Solar-Lezama, "Is self-repair a silver bullet for code generation?," in *Proc. 12th Int. Conf. Learning Representations (ICLR)*, Vienna, Austria, 2024. [Online]. Available: https://openreview.net/forum?id=y0GJXRungR (accessed Jul. 25, 2026).
6. A. Madaan et al., "Self-Refine: Iterative refinement with self-feedback," in *Advances in Neural Information Processing Systems 36 (NeurIPS)*, New Orleans, LA, USA, 2023, pp. 46534–46594. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07 232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html (accessed Jul. 25, 2026).
7. N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems 36 (NeurIPS)*, New Orleans, LA, USA, 2023, pp. 8634–8652. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b87 8bb782e6954cd888628510e90-Abstract-Conference.html (accessed Jul. 25, 2026).
8. X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," in *Proc. 12th Int. Conf. Learning Representations (ICLR)*, Vienna, Austria, 2024. [Online]. Available: https://openreview.net/forum?id=KuPixIqPiq (accessed Jul. 25, 2026).
9. Z. Gou et al., "CRITIC: Large language models can self-correct with tool-interactive critiquing," in *Proc. 12th Int. Conf. Learning Representations (ICLR)*, Vienna, Austria, 2024. [Online]. Available: https://openreview.net/forum?id=Sx038qxjek (accessed Jul. 25, 2026).
10. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg, Germany: Springer, 2012, doi: 10.1007/978-3-642-29044-2.
11. P. Ralph et al., "Empirical standards for software engineering research," arXiv:2010.03525, 2020, doi: 10.48550/arXiv.2010.03525.
12. R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research: An initial survey," in *Proc. 22nd Int. Conf. Software Engineering and Knowledge Engineering (SEKE)*, Redwood City, CA, USA, 2010, pp. 374–379.
13. L. Hatton, "The T-experiments: Errors in scientific software," *IEEE Computational Science and Engineering*, vol. 4, no. 2, pp. 27–38, Apr.–Jun. 1997, doi: 10.1109/99.609829.
14. D. C. Ince, L. Hatton, and J. Graham-Cumming, "The case for open computer programs," *Nature*, vol. 482, no. 7386, pp. 485–488, Feb. 2012, doi: 10.1038/nature10836.
15. J. Spacco, D. Hovemeyer, and W. Pugh, "Tracking defect warnings across versions," in *Proc. Int. Workshop Mining Software Repositories (MSR)*, Shanghai, China, 2006, pp. 133–136, doi: 10.1145/1137983.1138014.
16. P. Avgustinov et al., "Tracking static analysis violations over time to capture developer characteristics," in *Proc. IEEE/ACM 37th Int. Conf. Software Engineering (ICSE)*, vol. 1, Florence, Italy, 2015, pp. 437–447, doi: 10.1109/ICSE.2015.62.
17. ConsenSys Diligence, "mythril.mythril CRITICAL exception in mythril.py," mythril issue 939, GitHub. [Online]. Available: https://github.com/ConsenSysDiligence/mythril/issues/939 (accessed Jul. 25, 2026).
18. ConsenSys Diligence, "--solv throws various errors when not using the docker image," mythril issue 1424, GitHub. [Online]. Available: https://github.com/ConsenSysDiligence/mythril/issues/1424 (accessed Jul. 25, 2026).
19. T. Muske and P. Bokil, "On implementational variations in static analysis tools," in *Proc. IEEE 22nd Int. Conf. Software Analysis, Evolution, and Reengineering (SANER)*, Montréal, QC, Canada, 2015, pp. 512–515, doi: 10.1109/SANER.2015.7081867.
20. J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proc. IEEE/ACM 2nd Int. Workshop Emerging Trends in Software Engineering for Blockchain (WETSEB)*, Montreal, QC, Canada, 2019, pp. 8–15, doi: 10.1109/WETSEB.2019.00008.
21. ConsenSys Diligence, "Mythril: Security analysis tool for EVM bytecode," version 0.24.8, 2026. [Online]. Available: https://github.com/ConsenSysDiligence/mythril (accessed Jul. 25, 2026).
22. T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts," in *Proc. ACM/IEEE 42nd Int. Conf. Software Engineering (ICSE)*, Seoul, South Korea, 2020, pp. 530–541, doi: 10.1145/3377811.3380364.
23. F. Salzano, C. K. Antenucci, S. Scalabrino, G. Rosa, R. Oliveto, and R. Pareschi, "An empirical analysis of vulnerability detection tools for Solidity smart contracts using line level manually annotated vulnerabilities," *Empirical Softw. Eng.*, vol. 31, Art. no. 143, 2026, doi: 10.1007/s10664-026-10867-7.
24. J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "SmartBugs: A framework to analyze Solidity smart contracts," in *Proc. 35th IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, 2020, pp. 1349–1352, doi: 10.1145/3324884.3415298.
25. M. di Angelo, T. Durieux, J. F. Ferreira, and G. Salzer, "SmartBugs 2.0: An execution framework for weakness detection in Ethereum smart contracts," in *Proc. 38th IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, 2023, pp. 2102–2105, doi: 10.1109/ASE56229.2023.00060.
26. K. Li et al., "Static application security testing (SAST) tools for smart contracts: How far are we?," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, Art. no. 65, pp. 1447–1470, Jul. 2024, doi: 10.1145/3660772.
27. S. Bobadilla, M. Jin, and M. Monperrus, "Do automated fixes truly mitigate smart contract exploits?," *IEEE Trans. Softw. Eng.*, vol. 52, no. 1, pp. 100–115, 2026, doi: 10.1109/TSE.2025.3618123.
28. C. Sadowski, J. van Gogh, C. Jaspan, E. Söderberg, and C. Winter, "Tricorder: Building a program analysis ecosystem," in *Proc.*

IEEE/ACM 37th Int. Conf. Software Engineering (ICSE), vol. 1, Florence, Italy, 2015, pp. 598–608, doi: 10.1109/ICSE.2015.76.
29. C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, "Lessons from building static analysis tools at Google," *Commun. ACM*, vol. 61, no. 4, pp. 58–66, Apr. 2018, doi: 10.1145/3188720.
30. B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in *Proc. 35th Int. Conf. Software Engineering (ICSE)*, San Francisco, CA, USA, 2013, pp. 672–681, doi: 10.1109/ICSE.2013.6606613.
31. E. K. Smith, E. T. Barr, C. Le Goues, and Y. Brun, "Is the cure worse than the disease? Overfitting in automated program repair," in *Proc. 10th Joint Meeting Foundations of Software Engineering (ESEC/FSE)*, Bergamo, Italy, 2015, pp. 532–543, doi: 10.1145/2786805.2786825.
32. Z. Qi, F. Long, S. Achour, and M. Rinard, "An analysis of patch plausibility and correctness for generate-and-validate patch generation systems," in *Proc. Int. Symp. Software Testing and Analysis (ISSTA)*, Baltimore, MD, USA, 2015, pp. 24–36, doi: 10.1145/2771783.2771791.
33. A. Ghaleb, J. Rubin, and K. Pattabiraman, "AChecker: Statically detecting smart contract access control vulnerabilities," in *Proc. IEEE/ACM 45th Int. Conf. Software Engineering (ICSE)*, Melbourne, Australia, 2023, pp. 945–956, doi: 10.1109/ICSE48619.2023.00087.
34. Association for Computing Machinery, "Artifact review and badging, version 1.1," Aug. 2020. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-and-badging-current (accessed Jul. 25, 2026).
35. I. Staley, "Replication package: Three measurement hazards in analyzer-in-the-loop repair of smart contracts," Zenodo, Jul. 2026, doi: 10.5281/zenodo.21586404.
36. S. Baltes et al., "Guidelines for empirical studies in software engineering involving large language models," arXiv:2508.15503, 2025, doi: 10.48550/arXiv.2508.15503.