



Exploiting Ethereum Rollback Semantics: Profit-Driven Attack Synthesis and Off-Chain Misinterpretation Testing

ISSTA 2026

Yixuan Liu, Xinlei Li, and Yi Li

Nanyang Technological University, Singapore

Background: Transactions and Contract Calls

A smart contract is a program deployed on a blockchain. Transactions trigger its execution, which can update persistent state or transfer assets.



An external transaction starts the root call; internal calls run within the same transaction.

? Status:

✓ Success

Overall transaction: success

Type	Trace Address	Action ?	From	To
✓ L call		Send	Multicall3	→ 0x13988F9f...9Ef766f8c
✓ L L staticcall		Transfer*	0x13988F9f...9Ef766f8c	→ Null: 0x000...001
! L L call		Transfer	0x13988F9f...9Ef766f8c	→ 0x13988F9f...9Ef766f8c

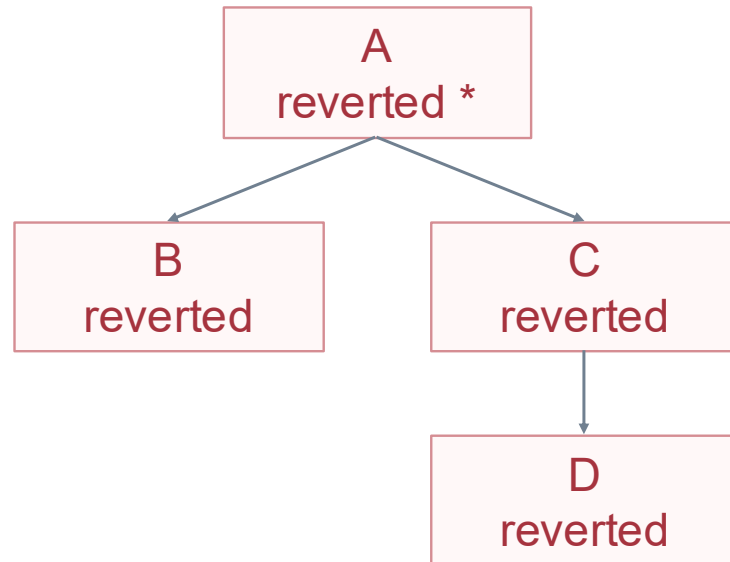
A successful transaction can contain failed internal calls.

Sources: RollGain §§2.1–2.3; Etherscan tx 0x535128...8811c. Two crops from the same transaction; all shown call values are 0 ETH.

Background: Rollback Across the Call Tree

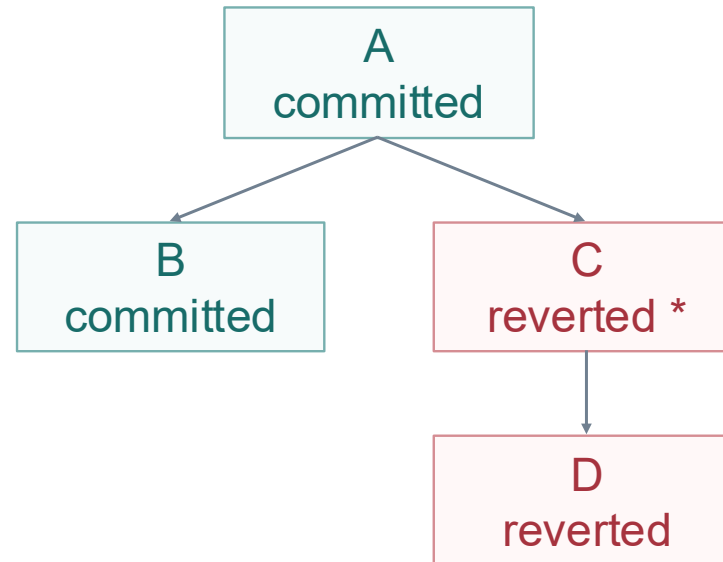
A call can complete successfully and still lose its effects when an ancestor reverts.

(a) Full rollback



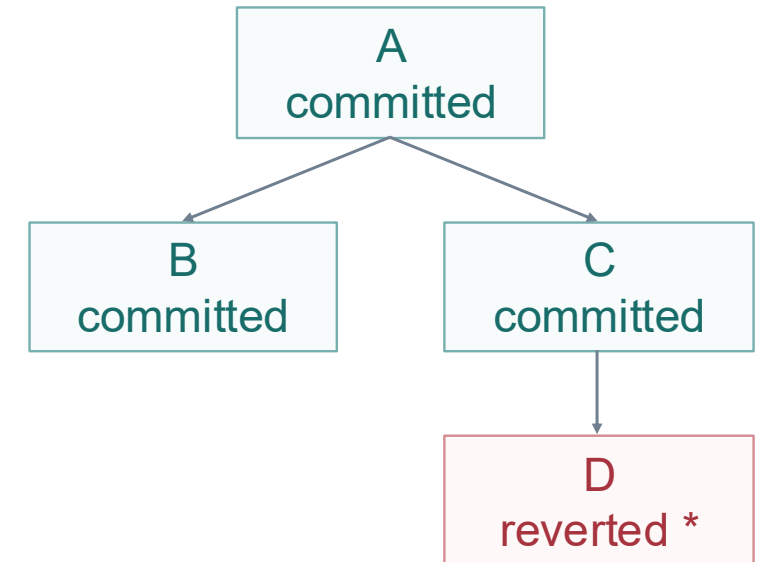
The root reverts; all call effects are discarded.

(b) Partial: C subtree



C reverts; D also loses its effects even without its own error.

(c) Partial: D only



D reverts; C handles the failure. A, B and C retain their effects.

Check the call and every ancestor before treating a transfer as committed.

Source: RollGain, §2.3 and Fig. 6. Same illustrative topology in all panels. * marks a call with an explicit error.

Threat Model: On-Chain and Off-Chain

Same ordinary privileges; different targets and failure conditions.

Deploy contracts · Invoke public functions · Submit transactions

Aspect	On-chain attack	Off-chain attack
Attacker controls	Function-call sequence within one transaction	Internal calls containing rollback
Objective	Positive gross asset gain	Incorrect record or follow-up action
Required weakness	Exploitable contract interaction	Consumer mishandles rollback

No assumed consensus control, ordering advantage or chain reorganization.

Source: RollGain, §2.5. Flash loans allowed; oracle manipulation and cross-transaction state control are not assumed.

Methodology: On-Chain Attack Model

Rollback filters executions by positive gross asset gain.

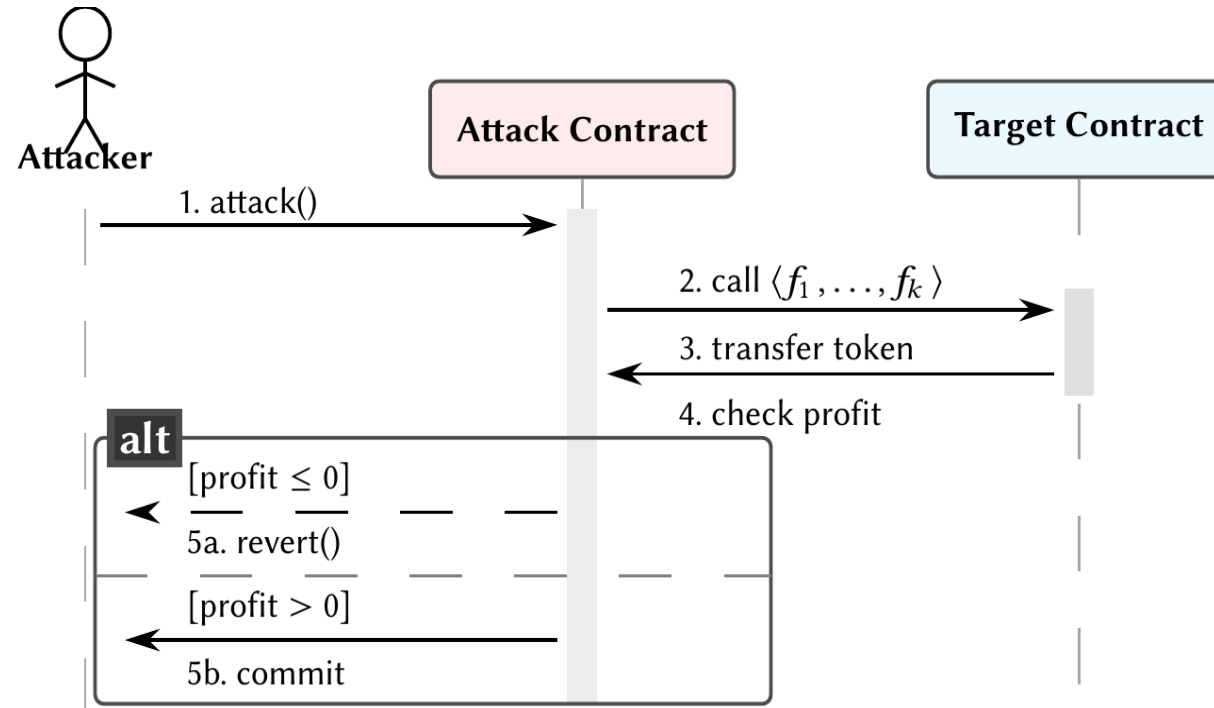


Fig. 1. On-chain rollback attack sequence.

Gross asset gain

$$G = \sum_t \delta_t \cdot p_t$$

δ_t Net balance change of asset t at the attacker address

p_t Price of asset t in units of the native token

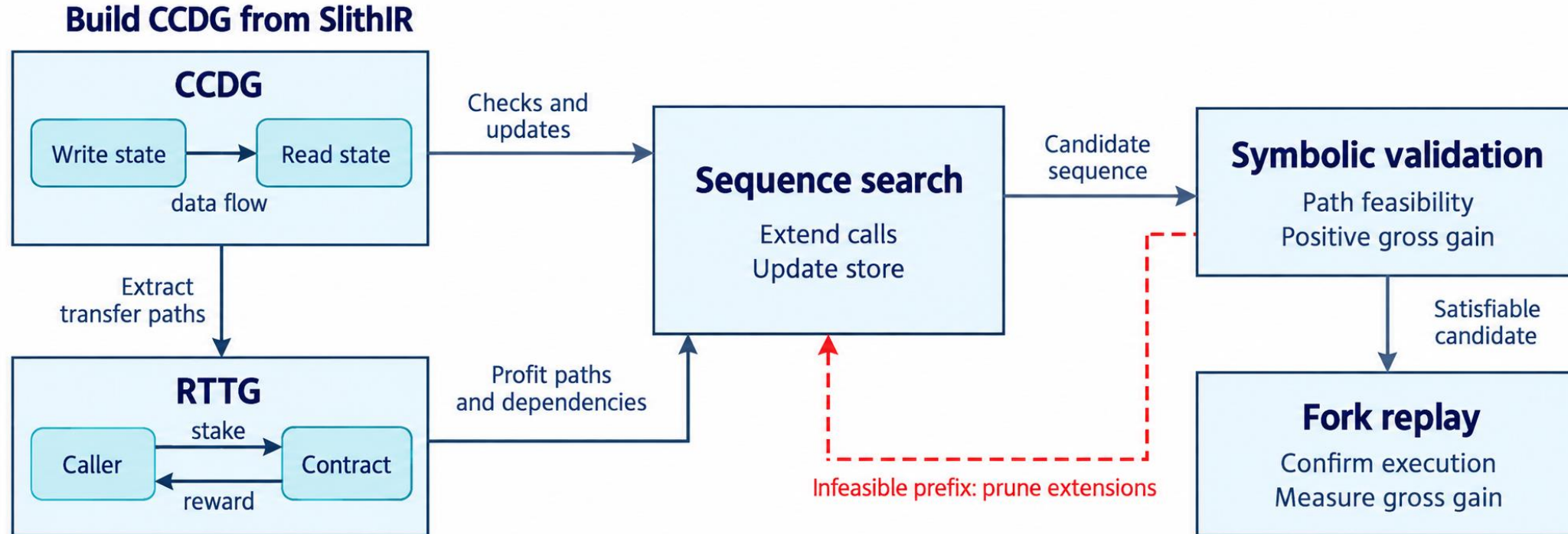
**Commit if $G > 0$.
Otherwise, revert.**

Gross gain excludes gas and fees; a passing sequence can still have a net loss.

Source: RollGain, §§2.5 and 3.5. δ_t measures the net balance change across the entire candidate sequence.

Methodology: RollGain Overview

Static models guide call ordering; validation returns infeasible prefixes to prune the search.



CCDG: Comprehensive Contract Dependency Graph

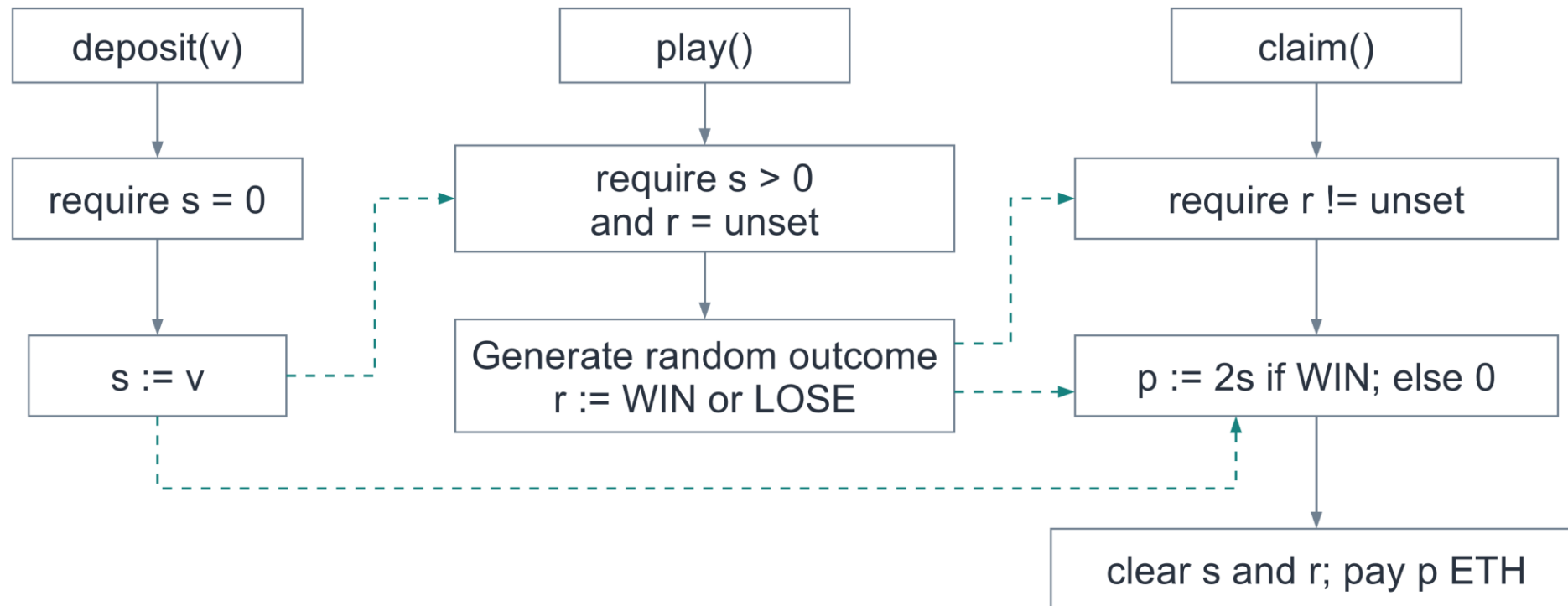
RTTG: Role-based Token Transfer Graph

Source: RollGain, Fig. 2, §3.3 and Algorithm 1. RTTG is derived from CCDG paths; dashed arrow shows pruning feedback.

Methodology: CCDG Construction

Comprehensive Contract Dependency Graph (CCDGG)

s: stake · r: unset / WIN / LOSE · Solid: control flow · Dashed: state dependency

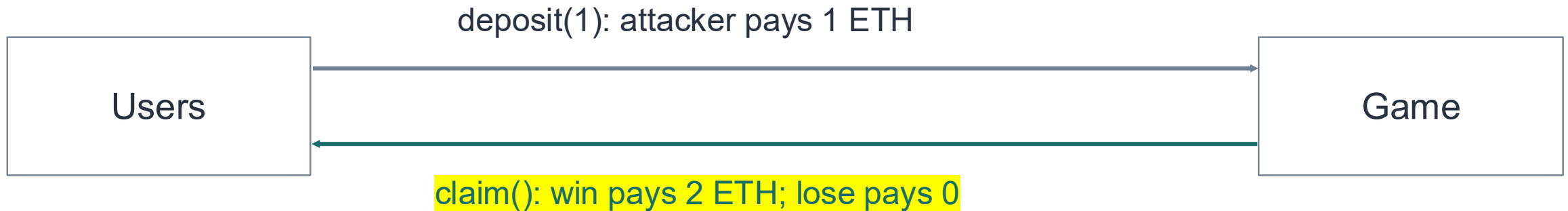


The result and payout are available before this transaction commits.

Illustrative reward game. Random outcome and payout occur in the same transaction. Method: RollGain, §3.3.1.

Methodology: RTTG Construction

Role-based Token Transfer Graph (RTTG)



Transfer	Amount	Execution condition	State dependencies
deposit(v)	v ETH	$s = 0$	{s}
claim() payout	2s ETH	$r = \text{win}$	{s, r}

play() changes the outcome without moving ETH. A losing claim pays nothing.

Same illustrative game. Game reserve ≥ 1 ETH before the 1 ETH stake; transfers must succeed. RollGain, §3.3.2.

Methodology: Candidate Sequence Search

Search chooses call order; the game outcome selects an execution branch.

Start: $s = 0$, $r = \text{unset}$. One candidate executes within one transaction.

Candidate sequence	Prerequisite check	Result
<code>play() → claim()</code>	play requires a stake	Invalid order
<code>deposit(1) → claim()</code>	claim requires a resolved outcome	Invalid order
<code>deposit(1) → play() → claim()</code>	Stake, then outcome, then payout	Executable; gain varies

Store: $(0, \text{unset}) \rightarrow (1, \text{unset}) \rightarrow (1, \text{win/lose}) \rightarrow (0, \text{unset})$

Validate the whole sequence: executability alone does not establish profit.

Illustrative state trace. Search can prune infeasible orders; profit validation checks each full execution path. RollGain, §§3.4–3.5.

Methodology: Profit Validation and Rollback

The attack contract checks gross gain after `claim()`, before committing.

`deposit(1) → play() → claim() → checkGain()`

Winning execution

Pay 1 ETH; receive 2 ETH

Gross gain = +1 ETH

$G > 0$: commit

Keep the payout and game state changes.

Losing execution

Pay 1 ETH; receive 0 ETH

Gross gain = -1 ETH

$G \leq 0$: revert the transaction

Undo the stake transfer and game updates.

Rollback discards the losing execution's effects; gas is still spent.

Illustrative atomic attempt. Retrying need not change the outcome; no guarantee of net profit. RollGain, §§3.1 and 3.5.

Evaluation: On-Chain Datasets

Profit exploitability under the threat model determines the positive evaluation set.

Stage / purpose	Dataset	Selection or reference
Candidate pool	472 contracts from Wei et al.	Deduplicated benchmark combining SWC, SmartBugs and other public sources
Recall / synthesis	43 profit-exploitable contracts	Two auditors independently label exploitability; resolve disagreements ($\kappa = 0.97$)
False-positive test	14 honeypot contracts	Separate negative evaluation set

Exclude phishing, denial of service and isolated defects without a profitable sequence.

Source: RollGain, §5. On-chain results are conditional on this threat-model-based selection.

Evaluation: On-Chain Effectiveness

Positive set: 43 profit-exploitable contracts · Negative set: 14 honeypots

Detection effectiveness

Tool	TP	FP	Prec.	Recall	F1
RollGain	41	0	100.0%	95.3%	0.98
Slither	32	11	74.4%	74.4%	0.74
Mythril	29	8	78.4%	67.4%	0.72
Oyente	26	12	68.4%	60.5%	0.64
Securify2	4	0	100.0%	9.3%	0.17
ConFuzzius	2	0	100.0%	4.7%	0.09

Ablation study

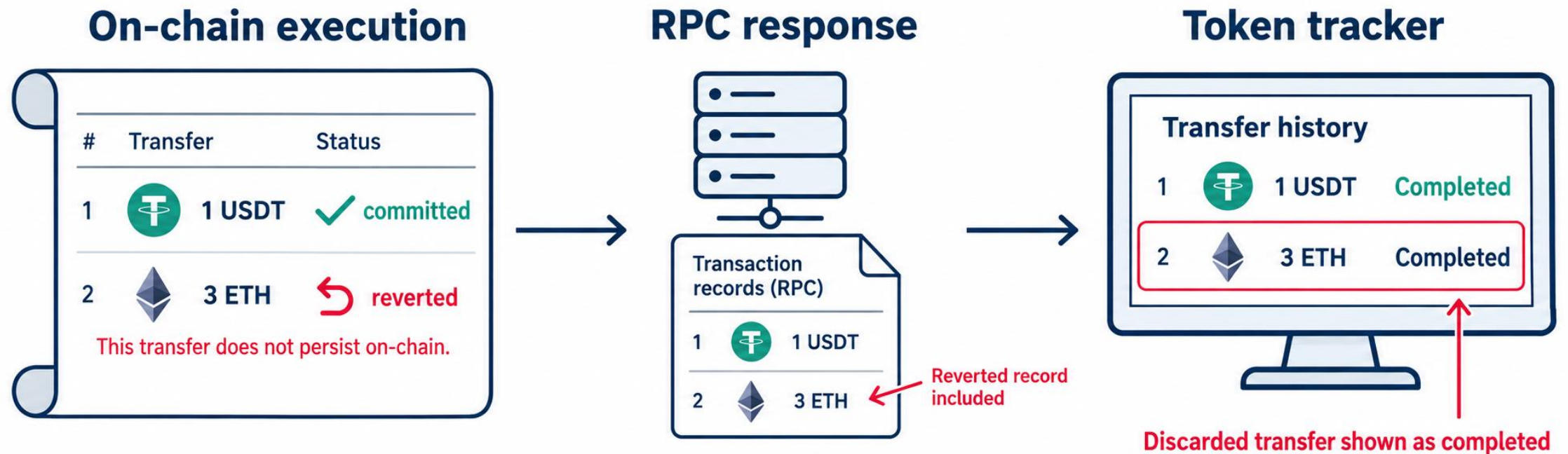
Config.	PoCs	Calls	Time
Full	38/43	180	54 min
w/o RTTG	32/43	561	92 min
Random	30/43	1026	113 min

38 of 41 detections yield fork-replayable PoCs (92.7%).

Source: RollGain, Table 2 and §5.3. PoC: proof of concept. Calls: validation calls; 10-minute limit per contract.

Motivation: Off-Chain Misinterpretation

Token trackers turn execution data into records of asset movement.

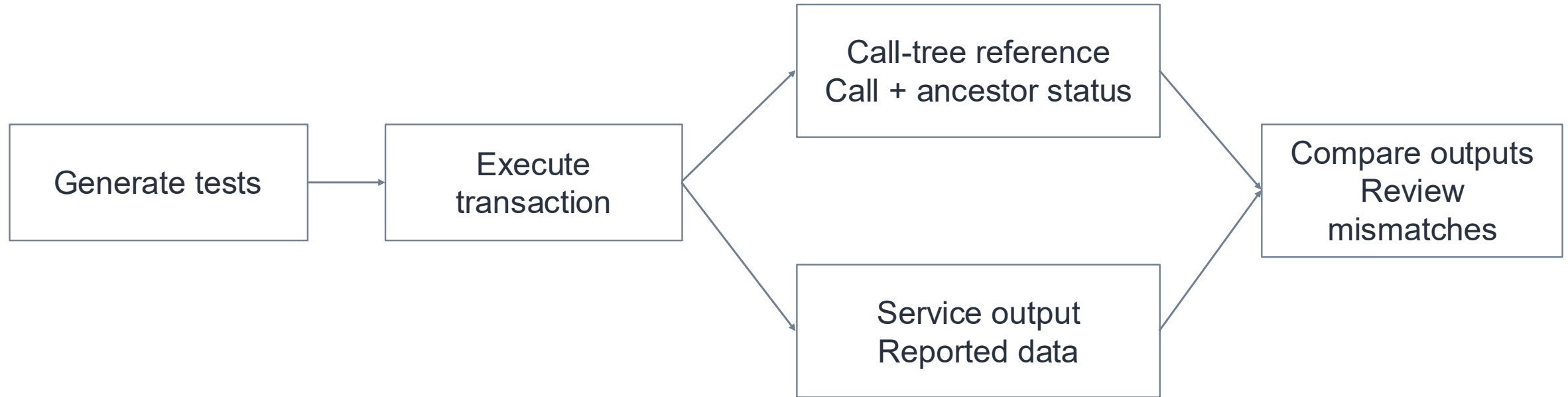


One real transfer and one discarded transfer are both displayed as completed.

Illustration of RollGain, Fig. 8 (not a service screenshot). The RPC provider patched the API after disclosure.

Methodology: Off-Chain Call-Trace Testing

Compare reported call statuses and transfer records with the call-tree reference.



Full rollback

Root failure; no call effects persist

Partial rollback

Root succeeds; some subcall effects revert

Source: RollGain, §§4.1–4.2, Fig. 5 and Table 1. Harnesses vary failure triggers, call depth and failure handling.

Evaluation: Off-Chain Analysis

Transaction study: 3.08B transactions · Service testing: rollback interpretation

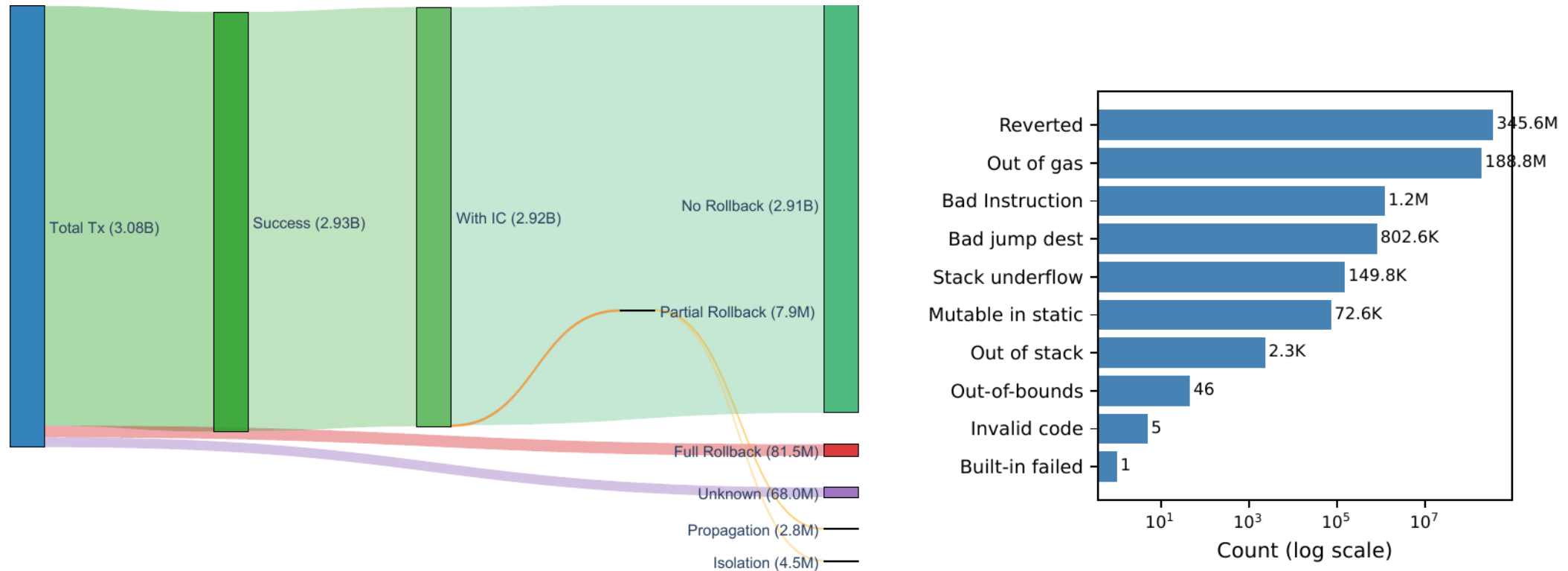


Fig. 7. Left: transaction rollback distribution. Right: internal call failure reasons.

20 vulnerabilities across 18 services: 18 confirmed, 16 fixed and 5 assigned CVEs.

Sources: XBlock-ETH + Dune, blocks 1–23,749,999; service selection: Alchemy, web search and GitHub (§§5.4–5.5).

Discussion: Mitigation

Protect contract interactions and interpret internal-call effects using the full call tree.

Contract-side controls

Apply checks-effects-interactions.

Constrain unsafe reentrant interleaving.

Use commit-reveal where appropriate.

Delay withdrawals when needed.

Consumer-side checks

Inspect receipt and internal-call trace.

Propagate rollback through each subtree.

Exclude reverted effects from balances.

Confirm execution before acting.

A successful subcall can still have its effects discarded by an ancestor.

Source: RollGain, §6. Mitigations depend on the contract and consumer; call-local success alone is insufficient.

Conclusion



Rollback is an exploitable execution semantic.

- 1 On-chain: synthesize sequences with a gross-gain guard.**
95.3% recall; 38 fork-replayable proof-of-concept exploits.
- 2 Off-chain: validate call outcomes and effects using the call tree.**
20 vulnerabilities across 18 services; 16 fixed and 5 CVEs.
- 3 Consider rollback semantics in both analysis and monitoring.**
Internal calls remain in traces even when their state effects are discarded.