> security_review · cantina_public_competition

Reserve Governor

Independent Security Assessment — Full Audit Report

PROTOCOL	PLATFORM	DATE
Reserve Governor	Cantina	May 2026

COMMIT	LINES REVIEWED
reserve-protocol/reserve-governor	2,742

HIGH	MEDIUM	LOW	INFORMATIONAL	TOTAL
1	1	7	3	12

AUDITORS

Prameya Singh Soni (blackburn)

MuscleFreak92

// Executive Summary

This report presents the findings of an independent security review of the Reserve Governor protocol, a hybrid optimistic/pessimistic on-chain governance system built on OpenZeppelin Governor v5. The audit was conducted as part of the Cantina public competition in May 2026.

The protocol introduces a dual-path proposal system: a fast optimistic path (with a veto period) and a standard pessimistic path (with a full community vote). Both paths share a single `TimelockControllerOptimistic` instance and are protected by a singleton `Guardian` contract that serves as the designated CANCELLER_ROLE holder.

A total of **12 findings** were identified across 9 core contracts and 2 library contracts, ranging from governance denial-of-service vulnerabilities to ERC4626 accounting issues and reward token edge cases. Two findings were submitted and validated by Cantina judges during the competition.

// Scope

CONTRACT	DESCRIPTION	LINES
StakingVault.sol	ERC4626 + ERC20Votes + dual delegation + multi-token rewards	572
ReserveOptimisticGovernor.sol	Hybrid governor — optimistic + standard proposals	525
ProposalLib.sol	Proposal creation and pessimistic transition logic (delegatecall)	233
Deployer.sol	Factory for full-system deployment	229
Guardian.sol	Singleton CANCELLER_ROLE holder for all timelocks	124
OptimisticSelectorRegistry.sol	(target, selector) allowlist for fast proposals	119
TimelockControllerOptimistic.sol	Shared timelock + bypass path for optimistic execution	99
VersionRegistry.sol	Versioned deployer registry	93

UnstakingManager.sol	Time-locked withdrawal manager	83
RewardTokenRegistry.sol	Governs which tokens may be used as rewards	58
ThrottleLib.sol	Per-proposer throttle (delegatecall)	57
Constants.sol + Versioned.sol	Shared role hashes and version mixin	41

// Findings Summary

ID	TITLE	SEVERITY	CONTRACT
H-01	Optimistic Proposer Aborts Confirmation Vote — Persistent Governance DoS	HIGH	Governor / ProposalLib
M-01	additionalGuardians Bypass Guardian Restrictions — Unrestricted Cancellation	MEDIUM	Deployer / Governor
L-01	Proposer Cancels Succeeded Optimistic Proposal — proposalId Permanently Locked	LOW	Governor
L-02	Veto Threshold Uses Floor Instead of Documented Ceil — Veto 1 Token Easier	LOW	Governor
L-03	Fee-on-Transfer Underlying Bricks Native Reward Accounting via Underflow	LOW	StakingVault
L-04	Missing _decimalsOffset() Override — ERC4626 First-Depositor Inflation Attack	LOW	StakingVault
L-05	No Upper Bound on vetoPeriod — Misconfiguration Freezes Governance 136 Years	LOW	Governor
L-06	Deflationary Reward Token Causes Permanent Vault DoS via _accrueRewards Underflow	LOW	StakingVault
L-07	Registry Deregistration Silently Discards Pending User Rewards	LOW	StakingVault
I-01	Governance Cannot Cancel Proposals via governor.cancel() from Within a Proposal	INFO	Governor

I-02	Zero-Supply Path Returns Canceled Without Setting proposalCore.canceled Flag	INFO	Governor
I-03	unregisterSelectors Race Window — In-Flight Optimistic Proposals Survive Removal	INFO	SelectorRegistry

// Detailed Findings

H-01 Optimistic Proposer Can Abort the Confirmation Vote and Re-Submit Indefinitely, Enabling Persistent Governance DoS

HIGH

When a vetoed optimistic proposal transitions to pessimistic governance, `ProposalLib.transitionToPessimistic()` sets the confirmation proposal's `proposer` to the original optimistic proposer. Since standard proposals can be canceled by their proposer during `Pending` state, the adversary can abort the community-triggered confirmation vote before voting begins — then re-submit the same calldata with a new description and repeat indefinitely.

```
// ProposalLib.sol — transitionToPessimistic()
ProposalData memory proposalData = ProposalData(
    newProposalId,
    governor.proposalProposer(proposalId),  // original optimistic proposer
    ...
);
```

```
// ReserveOptimisticGovernor.sol — _validateCancel()
return _isOptimistic(proposalId) ? s != ProposalState.Defeated : s ==
ProposalState.Pending;
//  standard proposal: proposer may cancel during Pending ← confirmation is standard
```

Impact: Any governance action routed through the optimistic path can be permanently prevented from reaching a community vote. Each cycle permanently consumes a confirmation proposalId. Revoking the attacker's role requires a slow governance proposal the attacker can disrupt with remaining throttle slots.

Recommendation: In `_validateCancel`, block proposer cancellation for proposals whose description starts with `CONFIRMATION_PREFIX`. Alternatively, assign the confirmation's proposer to `address(0)` in `transitionToPessimistic`.

CONTRACT	LIKELIHOOD	IMPACT	STATUS
Governor / ProposalLib	Low	High	Validated on Cantina

M-01 Inconsistent Access Control Allows additionalGuardians to Bypass Guardian Restrictions and Unconditionally Cancel Any Proposal

MEDIUM

The `Deployer` contract grants `CANCELLER_ROLE` on the timelock directly to every address in `additionalGuardians[]`. Because `_validateCancel()` returns `true` for any `CANCELLER_ROLE` holder unconditionally, these addresses can cancel any proposal — including standard proposals and active confirmation votes — without passing through the `Guardian` contract's access control logic.

```
// Deployer.sol
for (uint256 i = 0; i < baseParams.additionalGuardians.length; ++i) {
    _timelock.grantRole(CANCELLER_ROLE, baseParams.additionalGuardians[i]);
}

// ReserveOptimisticGovernor.sol
if (t.hasRole(CANCELLER_ROLE, caller)) {
    return true;  // no further checks — any CANCELLER_ROLE holder is unrestricted
}
```

Impact: additionalGuardians can censor standard governance proposals, cancel active confirmation votes, and complicate their own removal. This contradicts the Guardian NatDoc which declares it the sole CANCELLER_ROLE holder: *"Singleton contract to serve as CANCELLER_ROLE for all timelocks."*

Recommendation: Do not grant CANCELLER_ROLE directly to additionalGuardians. Instead, grant them DEFAULT_ADMIN_ROLE on the Guardian contract so all cancellations route through Guardian's validation logic.

CONTRACT	LIKELIHOOD	IMPACT	STATUS
Deployer / Governor	Medium	High	Validated on Cantina

L-01

Proposer Can Cancel a Succeeded Optimistic Proposal, Permanently Locking the proposalId

LOW

For optimistic proposals, `_validateCancel` allows the proposer to cancel in any state except `Defeated`. A proposer can therefore cancel their own proposal after it has `Succeeded` (veto period passed without threshold). The proposalId is permanently consumed and cannot be reused for identical calldata.

Recommendation: Restrict proposer cancellation of optimistic proposals to `Pending` state only (before the veto period starts).

CONTRACT	LIKELIHOOD	IMPACT
Governor	Low	Low

L-02

Veto Threshold Calculation Uses Floor+max(1) Instead of Documented ceil(), Making Veto 1 Token Easier Than Specified

LOW

The README specifies `vetoThreshold = ceil(vetoThresholdRatio × pastTotalSupply / 1e18)`. The implementation uses integer floor division then `Math.max(..., 1)` — not ceil. For supplies where the ratio produces a fractional result, the actual threshold is 1 token lower than documented.

```
// Code: floor + max(1)
uint256 vetoThresholdTok = (_vetoThreshold * pastSupply) / 1e18;
vetoThresholdTok = Math.max(vetoThresholdTok, 1);

// proposalThreshold correctly uses ceil:
return Math.mulDiv(proposalThresholdBps, token.getPastTotalSupply(clock() - 1),
10_000, Math.Rounding.Ceil);
```

Recommendation: Use `Math.mulDiv(..., Math.Rounding.Ceil)` to match the documented specification.

CONTRACT	LIKELIHOOD	IMPACT
Governor	Medium	Low

L-03

Fee-on-Transfer or Balance-Decreasing Underlying Permanently Bricks Native Reward Accounting via uint256 Underflow

LOW

`_deposit()` increments both `totalDeposited` and `nativeBalanceLastKnown` by `assets` before the actual token transfer. For a fee-on-transfer underlying, the vault receives `assets - fee` but both variables reflect the full `assets` amount. On the next `_accrueRewards()` call, `nativeBalanceLastKnown - totalDeposited` underflows, permanently bricking all native reward accounting. Recovery requires a UUPS upgrade.

Recommendation: Measure the actual received amount via balance snapshots before/after transfer, or check balance post-transfer and revert if fee is detected.

CONTRACT	LIKELIHOOD	IMPACT
StakingVault	Low	High

L-04

Missing _decimalsOffset() Override Enables First-Depositor Share-Inflation Attack (Dampened by Streaming)

LOW

StakingVault inherits OZ ERC4626Upgradeable but does not override `_decimalsOffset()`, which defaults to 0. With offset=0, only 1 virtual share and 1 virtual asset are added to the conversion math — providing negligible protection against the classic first-depositor inflation attack. An attacker deposits 1 wei, donates a large amount, and victim deposits round down to 0 shares. The native reward streaming mechanism dampens but does not eliminate the attack.

Recommendation: Override `_decimalsOffset()` to return 6 or 18 to add meaningful virtual share protection.

CONTRACT	LIKELIHOOD	IMPACT
StakingVault	Low	Medium

L-05

No Upper Bound on vetoPeriod or votingPeriod — Misconfiguration Can Freeze Governance for 136 Years

LOW

`vetoPeriod` and `votingPeriod` are stored as `uint32` with `type(uint32).max` = 4,294,967,295 seconds ≈ 136 years. No upper bound is enforced in `setOptimisticParams` or initialization. A misconfigured or malicious governance proposal setting either value to max would make all new proposals effectively unexecutable without cancellation.

Recommendation: Add a sanity upper bound (e.g., 30 days) on both parameters in the setter and initializer.

CONTRACT	LIKELIHOOD	IMPACT
ReserveOptimisticGovernor	Low	Medium

L-06

Deflationary or Negative-Rebasing Reward Token Permanently Bricks Vault via _accrueRewards Underflow

LOW

In `_accrueRewards(address)`, the line `balanceLastKnown - rewardInfo.balanceAccounted` computes a uint256 subtraction. For a deflationary or negative-rebasing reward token, the actual balance can fall below `balanceAccounted`, causing a permanent underflow panic on every subsequent vault interaction. Unlike the native reward case (L-03), recovery is possible via `removeRewardToken()`, but requires admin intervention before the panic state is hit.

Recommendation: Use checked arithmetic with a floor at 0, or validate reward token behavior at add time.

CONTRACT	LIKELIHOOD	IMPACT
StakingVault	Low	Medium

L-07 Registry-Side Reward Token Deregistration Silently Discards Pending User Rewards Without Event

LOW

When `RewardTokenRegistry` deregisters a token that a `StakingVault` still holds in its `rewardTokens` set, every subsequent `_accrueRewards` call silently discards the pending reward delta for that token. Unlike `removeRewardToken()` (which emits an event and has a NatDoc warning), this occurs without any vault-side action or notification — users with unclaimed reward index deltas permanently lose those rewards.

Recommendation: Emit an event or require explicit vault-side acknowledgment before registry deregistration affects vault reward accounting.

CONTRACT	LIKELIHOOD	IMPACT
StakingVault	Low	Medium

I-01 Governance Cannot Cancel Proposals via governor.cancel() from Within a Governance Proposal

INFO

The Governor holds CANCELLER_ROLE on the timelock. However, governance proposals execute through the timelock — the `msg.sender` for target calls is the timelock, not the governor. Since the timelock does not hold CANCELLER_ROLE on itself, `_validateCancel` denies cancellation. Governance can only cancel via `guardian.cancel()` or `timelock.cancel(operationId)` directly.

Recommendation: Document this behavior explicitly so governance designers are not surprised by it.

CONTRACT	IMPACT
Governor	Informational

I-02 Zero-Supply Path Returns Canceled State Without Setting proposalCore.canceled Flag

INFO

When `pastTotalSupply(snapshot) == 0`, `state()` returns `Canceled` — but `proposalCore.canceled` is never set. A subsequent call to `_cancel()` will revert because the proposal appears to be in a non-cancellable state bitmap. The throttle charge from the original proposal is permanently consumed.

Recommendation: Set `proposalCore.canceled = true` in the zero-supply path, or treat zero-supply as `Expired` to avoid state inconsistency.

CONTRACT	IMPACT
Governor	Informational

I-03 unregisterSelectors Race Window — In-Flight Optimistic Proposals Survive Selector Removal

INFO

The selector allowlist is checked only at `proposeOptimistic()` time. `executeBatchBypass()` does not re-validate selectors at execution. A proposal submitted with a valid selector can still execute after that selector is unregistered via a governance proposal. The Guardian must monitor and cancel such proposals independently.

Recommendation: Document this as an explicit invariant that Guardian operators must enforce, or add optional re-validation at execution time.

CONTRACT	IMPACT
OptimisticSelectorRegistry / Governor	Informational