PUBLIC SAMPLE — REDACTED · client identity removed; ▮▮ bars mark removed identifiers · methodology & findings intact

ARI

FULL-STACK SECURITY REVIEW

SECURITY REVIEW — STAKING

[redacted] · Track T4 — Validator-node staking & reward distributor

PROJECT	TRACK / TIER	DATE
[redacted]	T4 · Standard	12 May 2026
CHAINS	**CODE VERSION**	**CLASSIFICATION**
Ethereum	locked at kickoff (pending)	PUBLIC SAMPLE

Disclaimer. This report describes a time-boxed technical security review of the materials identified in the Scope of Review as they existed at the referenced commit/version during the review window. It is a point-in-time assessment: subsequent changes to code, configuration, infrastructure, or dependencies are not covered. No security review can identify all vulnerabilities. This report does not constitute a guarantee, warranty, or insurance of any kind, and it is not an endorsement of the project, its business model, or its tokens. It is not investment, legal, or financial advice. The review does not include formal verification, continuous monitoring, or exhaustive novel economic attack modeling unless expressly stated in scope. [redacted] may publish or share this report in full, unmodified form, including this disclaimer. Partial reproduction that omits findings, statuses, or this disclaimer is not authorized. Vari accepts no liability for decisions made by third parties on the basis of this report.

00 EXECUTIVE SUMMARY

Staking is an upgradeable single-token staking/reward distributor across validator “nodes”. Owner pushes rewards via distributeReward, split between a full-power accumulator and a time-weighted partial-power term for a stake’s first distribution. The core reward math is careful and internally consistent: per-stake divisions round down so dust stays in the contract (system stays over-collateralized), and gross reward is pulled on each distribution so principal, rewards and fees are jointly funded. No non-owner fund-theft path was found. The material issues are owner/validator operational hazards: an unbounded fee in createNode can permanently brick a node, a batch distribution reverts wholesale if any node is inactive, and setNodeValidator can desync validator accounting and lock a validator’s own principal.

CRITICAL	HIGH	MEDIUM	LOW	INFO
0	0	3	4	1

No critical or high-severity issues; findings are medium and below.

01 SCOPE OF REVIEW

Staking core + error library. Upgradeable (Ownable). [redacted]'s own code ([redacted]).

`TRACK :: T4 · TIER :: Standard · CHAINS :: Ethereum · CODE :: locked at kickoff (pending)`

Out of scope: third-party dependencies (base libraries, external protocols, bridge/oracle endpoints), off-chain keeper/relayer infrastructure, and key management, assumed correct unless expressly listed. Test, mock and deploy-script files were read for context but are not the subject of the findings.

02 METHODOLOGY

AI-assisted broad analysis over the full source followed by targeted human review of business logic, value-moving paths, roles/permissions and higher-risk areas; cross-layer reconciliation; and live-state verification of the deployed settings (see §05). The review is manual + AI-assisted + static; it does not include formal verification or runtime fuzzing unless stated. Severity reflects exploitability + impact, never effort: **Critical** (direct path to loss of funds / full compromise) · **High** (exploitable, realistic conditions, material impact) · **Medium** (limited impact or unusual preconditions) · **Low** (defense-in-depth / best practice) · **Info** (observation).

03 FINDINGS

ID	SEVERITY	LAYER	FINDING
PROTO-T4-01	MEDIUM	contracts	createNode does not bound fee; a node created with fee ≥ 100% is permanently bricked
PROTO-T4-02	MEDIUM	contracts	distributeReward batch reverts entirely if any single node is inactive
PROTO-T4-03	MEDIUM	contracts	setNodeValidator blindly zeroes validator accounting; can deactivate a live node and lock the new validator’s stake
PROTO-T4-04	LOW	contracts	No reentrancy guards; unstake transfers reward before finalizing state — safe only under the non-hooking-token assumption
PROTO-T4-05	LOW	contracts	Accumulated/withdrawn reward stored as uint96; a stake exceeding the cap locks reward and principal
PROTO-T4-06	LOW	contracts	Division-by-zero revert when a distribution shares a timestamp with the previous one
PROTO-T4-07	LOW	contracts	Implementation lacks constructor _disableInitializers(); no storage gap
PROTO-T4-08	INFO	contracts	Standard-ERC20 assumption unguarded; no pause/emergency lever

MEDIUM PROTO-T4-01 **createNode does not bound fee; a node created with fee ≥ 100% is permanently bricked**

LOCATION

`Staking.sol: createNode / distributeReward`

DESCRIPTION

createNode sets the node fee with no upper bound, unlike setFee which requires fee < DENOMINATOR. In distributeReward, feeAmount = reward×fee/DENOMINATOR then reward − feeAmount; if fee ≥ DENOMINATOR this underflows and reverts. setFee cannot rescue it because the new fee only takes effect once lastDistributionId advances, which requires a successful distribution — exactly what reverts. Deadlock.

IMPACT

A node misconfigured with fee ≥ 100% can never distribute rewards and cannot be repaired; its stakers earn nothing forever (principal remains withdrawable). Owner footgun with permanent effect for that node.

RECOMMENDATION

Add require(fee < DENOMINATOR) to createNode, matching setFee.

Status: OPEN · Confidence: high

MEDIUM PROTO-T4-02 **distributeReward batch reverts entirely if any single node is inactive**

LOCATION

`Staking.sol: distributeReward / _distributeReward`

DESCRIPTION

_distributeReward requires stakedByValidator ≥ validatorMinimalStake. A validator can unstake below the minimum at any time; if the owner submits a batch that includes such a node, the entire multi-node distribution reverts, delaying rewards for all other nodes in the batch.

IMPACT

Liveness/griefing: one validator (or a single bad node id) blocks reward distribution for every node in the batch until the owner detects and re-submits. No direct fund loss.

RECOMMENDATION

Skip inactive nodes instead of reverting (per-node try/catch semantics), or enforce off-chain that batches exclude inactive nodes and emit a signal.

Status: OPEN · Confidence: high

MEDIUM PROTO-T4-03

setNodeValidator blindly zeroes validator accounting; can deactivate a live node and lock the new validator’s stake

LOCATION

`Staking.sol: setNodeValidator / _unstake`

DESCRIPTION

setNodeValidator sets the new validator and zeroes stakedByValidator, guarded only by a code comment. Two hazards: (1) if the node was active via the old validator’s stake, zeroing immediately deactivates it though that stake still sits in totalStaked; (2) if the new validator already held ordinary stakes, unstake computes 0 − amount and reverts (uint96 underflow), so they cannot unstake their own positions until they first stake enough as validator.

IMPACT

Owner action can transiently lock a validator’s own principal and desync validator accounting, deactivating a node with real stake. Recoverable but confusing; funds temporarily stuck.

RECOMMENDATION

Do not silently zero the accounting: require no existing stakes, migrate explicitly, or recompute from the new validator’s actual stakes; replace the comment-only precondition with an enforced check.

Status: OPEN · Confidence: high

LOW PROTO-T4-04

No reentrancy guards; unstake transfers reward before finalizing state — safe only under the non-hooking-token assumption

LOCATION

`Staking.sol: _unstake / _withdrawReward / withdrawFee`

DESCRIPTION

None of the stake/unstake/withdraw paths use nonReentrant. _unstake withdraws reward (token transfer to caller) before finishing state updates; the current code is safe because it re-reads stake storage after the transfer and [redacted] is a standard ERC20. With a hook-bearing token this ordering becomes a cross-call hazard.

IMPACT

No exploit with the intended token; latent risk if token assumptions change or the contract is reused with a callback token.

RECOMMENDATION

Add ReentrancyGuardUpgardeable and mark stake/unstake/withdraw paths nonReentrant; treat the token as untrusted.

Status: OPEN · Confidence: medium

LOW PROTO-T4-05 **Accumulated/withdrawn reward stored as uint96; a stake exceeding the cap locks reward and principal**

LOCATION

`Staking.sol: _accumulatedRewardOf / rewardOf / _withdrawReward`

DESCRIPTION

Accumulated reward is cast to uint96 and withdrawn is uint96; if a single stake's lifetime reward exceeds ~7.9e28, the cast reverts, and because unstake calls the reward path, both reward withdrawal and principal unstake revert permanently for that stake.

IMPACT

For very large/long-lived positions, a hard cap with catastrophic effect if hit. Unlikely at realistic [redacted] magnitudes.

RECOMMENDATION

Track accumulated/withdrawn in uint256, or document and enforce a maximum that keeps cumulative reward below the cap.

Status: OPEN · Confidence: medium

LOW PROTO-T4-06 **Division-by-zero revert when a distribution shares a timestamp with the previous one**

LOCATION

`Staking.sol: _distributeReward`

DESCRIPTION

With fresh partial stakes present, maxTotalPowerXTime = stakedIn × (now − previous distribution timestamp); if two distributions land in the same block, this is zero and the partial-power division reverts.

IMPACT

Transient DoS of that distribution; owner must wait at least one second. No fund loss and no bad state persists.

RECOMMENDATION

Guard the same-timestamp case (require elapsed > 0, or short-circuit partial power to 0 when the denominator is 0).

Status: OPEN · Confidence: high

LOW PROTO-T4-07 **Implementation lacks constructor _disableInitializers(); no storage gap**

LOCATION

`Staking.sol: contract Staking / initialize`

DESCRIPTION

The upgradeable implementation has no constructor calling _disableInitializers(), so the logic contract itself can be initialized directly (no proxy impact, but best practice). No __gap is declared.

IMPACT

Low: implementation can be initialized; future inheritance changes could complicate storage layout.

RECOMMENDATION

Add a constructor calling _disableInitializers(); confirm the storage-layout policy.

Status: OPEN · Confidence: high

INFO PROTO-T4-08 **Standard-ERC20 assumption unguarded; no pause/emergency lever**

LOCATION

`Staking.sol: stakeFor / distributeReward / token`

DESCRIPTION

Accounting credits nominal amounts; a fee-on-transfer token would over-credit (insolvency) and a rebasing token would drift. The token is fixed at init ([redacted]) so this is currently safe but unenforced. There is also no pause to halt staking/distributions during an incident.

IMPACT

None with [redacted]; latent accounting breakage if reused with a non-standard token; limited incident response.

RECOMMENDATION

Document the standard-ERC20 requirement; consider a pause on stake/distribute paths.

Status: OPEN · Confidence: medium

04 WHAT'S SOLID

- Full-power per-stake rewards round down, so claims never exceed the distributed amount and dust remains as surplus (no insolvency from the full-power path).
- Partial-power (time-weighted) accounting is consistent: per-stake partial rewards sum to the distributed partial amount minus rounding dust, and a just-staked position earns ~0 partial reward (no JIT capture).
- distributeReward pulls gross reward (net + fee) up front so staker rewards and validator fees are both funded; principal held 1:1.
- State-before-transfer ordering on the reward path; unstake re-reads stake/node storage after the transfer, neutralizing the obvious single-function reentrancy double-withdraw.
- setFee enforces fee < DENOMINATOR and defers fee changes to the next distribution; SafeCast guards downcasts.

05 LIVE ON-CHAIN VERIFICATION

> **Pending deployed addresses.** Governance/config findings must reflect the deployed state, not the source. The items below are queued to be read live (via Etherscan V2 proxy `eth_call`) once [redacted] provides the deployed addresses + chains per system; each will be reported PASS / ACTION / VERIFY / INFO. Until then they are marked VERIFY.

- token address in the deployed proxy — confirm [redacted] (standard 18-dec, no FoT/rebasing/hooks).
- owner() and whether it is a timelock/multisig (owner controls createNode, distributeReward, setFee, setNodeValidator and upgrades).
- Upgrade authority / whether the implementation was left initializable.
- validatorMinimalStake and each live node's fee (no node with fee ≥ DENOMINATOR).
- Per node: validator, stakedByValidator vs minimum (active status), totalStaked, collectedFee; consistency after any setNodeValidator rotation.
- Contract token balance vs (stake principal + unclaimed rewards + collected fees) to confirm solvency.

06 ACTION POINTS

PRIORITY	ID	ACTION	STATUS
P1	PROTO-T4-01	createNode does not bound fee; a node created with fee ≥ 100% is permanently bricked	◦ OPEN
P1	PROTO-T4-02	distributeReward batch reverts entirely if any single node is inactive	◦ OPEN
P1	PROTO-T4-03	setNodeValidator blindly zeroes validator accounting; can deactivate a live node and lock the new validator’s stake	◦ OPEN
P2	PROTO-T4-04	No reentrancy guards; unstake transfers reward before finalizing state — safe only under the non-hooking-token assumption	◦ OPEN
P2	PROTO-T4-05	Accumulated/withdrawn reward stored as uint96; a stake exceeding the cap locks reward and principal	◦ OPEN
P2	PROTO-T4-06	Division-by-zero revert when a distribution shares a timestamp with the previous one	◦ OPEN
P2	PROTO-T4-07	Implementation lacks constructor _disableInitializers(); no storage gap	◦ OPEN
P2	PROTO-T4-08	Standard-ERC20 assumption unguarded; no pause/emergency lever	◦ OPEN

P0 = before deploy / now · P1 = soon · P2 = hardening. Status column is filled at the re-review round (report goes to v1.1: ◦ OPEN · ✔ FIXED · ~ ACKNOWLEDGED · ✗ NOT FIXED).

07 RESIDUAL RISK

After the recommended fixes, residual risk concentrates in the privileged roles (owner/manager/strategist/governor keys) and in operational posture — key custody, multisig thresholds, timelocks, and monitoring — which code alone cannot guarantee. A short economic note on the reward/liveness coupling accompanies the funds-at-risk tracks. This is a point-in-time review; re-verify after any change to code, configuration, or deployed settings.