Defense By Thesis

Security Audit Report

tLabs

Beta Signer Allowlist & Bridge Fee Rebates

Initial Report // September 16, 2025
Final Report // September 25, 2025

Team Members

Ahmad Jawid Jamiulahmadi // Senior Security Auditor
Mukesh Jaiswal // Senior Security Auditor

Table of Contents

About Thesis Defense

Defense is the security auditing arm of Thesis, Inc., the venture studio behind tBTC, Fold, Mezo, Acre, Taho, Etcher, and Embody. At Defense, we fight for the integrity and empowerment of the individual by strengthening the security of emerging technologies to promote a decentralized future and user freedom. Defense is the leading Bitcoin applied cryptography and security auditing firm. Our team of security auditors have carried out hundreds of security audits for decentralized systems across a number of ecosystems including Bitcoin, Ethereum + EVMs, Stacks, Cosmos SDK, NEAR and more. We offer our services within a variety of technologies including smart contracts, bridges, cryptography, node implementations, wallets and browser extensions, and dApps.

Defense will employ the Defense Audit Approach and Audit Process to the in scope service. In the event that certain processes and methodologies are not applicable to the in scope services, we will indicate as such in individual audit or design review SOWs. In addition, Thesis Defense provides clear guidance on successful Security Audit Preparation.

Section 1.0
Scope

Technical Scope

- **Repository:** https://github.com/threshold-network/tbtc-v2/tree/main/cross-chain/bob/contracts
- **Audit Commit:** dbefc8c0cbcd686b319a889b6589047cb4ed1f69
- **Verification Commit:** 4ad14049fab1a4b7f5a7b773fd53c4f66f2fdf70

Section 2.0
Executive Summary

Schedule

This security audit was conducted from September 8, 2025 to September 15, 2025 by 2 security auditors for a total of 12 person-days.

Overview

This audit covers two pull requests implementing the following changes:

Allowlist Contract Integration

The new `allowlist` smart contract replaces the `TokenStaking` smart contract as per TIP-092 and TIP-100. Staking is no longer required to operate nodes; instead, beta stakers are selected by the DAO and authorized via a DAO-managed `allowlist`. The contract integrates with the `WalletRegistry` smart contract and replaces previous `TokenStaking` calls.

Bridge Fee Rebate Model for T Stakers

A rebate mechanism was introduced for T stakers, granting fee rebates based on the amount of tokens staked. The model includes a cap enforced via a rolling window per token staked, scaling linearly with bridge usage, and includes an unstaking period.

Tests

A comprehensive test suite is implemented.

Project Documentation

We found comprehensive in the PR descriptions in and inline documentation in the code.

Section 3.0
Key Findings Table

Issues	Severity	Status
ISSUE #1 Staking Provider Can Be Disabled Without Two-Step Process	Medium	Fixed
ISSUE #2 Missing Zero-Address Validation in the `finalizeUnstaking` Function	Low	Fixed
ISSUE #3 Lack of Zero Address Validation in `initialize` and `addStakingProvider` Functions	None	Fixed
ISSUE #4 Missing Zero-Amount Validation in the `stake` Function	None	Fixed
ISSUE #5 Missing Zero-Weight Check in the `addStakingProvider` Function	None	Fixed
ISSUE #6 Cache Array Length in Loops for Gas Savings	None	Fixed
ISSUE #7 Use Custom Errors to Save Gas	None	Fixed
ISSUE #8 Unrestricted `seize` Function Allows Event Spam	None	Not Fixed

Severity definitions can be found in Appendix A

Section 4.0
Findings

We describe the security issues identified during the security audit, along with their potential impact. We also note areas for improvement and optimizations in accordance with best practices. This includes recommendations to mitigate or remediate the issues we identify, in addition to their status before and after the fix verification.

ISSUE#1

Staking Provider Can Be Disabled Without Two-Step Process

Medium Fixed

Location

contracts/Allowlist.sol#L109

contracts/Allowlist.sol#L138

Description

The decrease of the weight of a staking provider is a two step process. However, a staking provider can be directly disabled by the `approveAuthorizationDecrease` function without first calling the `requestWeightDecrease` function. This results from `pendingNewWeight` defaulting to 0, with no explicit flag to verify that a decrease request was made.

Impact

This allows the `WalletRegistry` to potentially reduce a staking provider's weight to 0, bypassing the intended two-step process.

Recommendation

We recommend adding a `decreasePending` flag to track valid requests, requiring it in the `approveAuthorizationDecrease` function, and resetting it after finalization to ensure that the two-step process is followed.

ISSUE#2

Missing Zero-Address Validation in the `finalizeUnstaking` Function

Low Fixed

Location

contracts/bridge/RebateStaking.sol#L324

Description

The `finalizeUnstaking(address receiver)` function allows the `receiver` to be the zero address. This can lead to token transfers to `address(0)`, resulting in permanent loss of tokens.

Impact

Potential irreversible fund loss if tokens are sent to `address(0)`.

Recommendation

We recommend adding a zero address check for the `receiver` to prevent potential permanent loss of funds.

ISSUE#3

Lack of Zero Address Validation in `initialize` and `addStakingProvider` Functions

None Fixed

Location

contracts/Allowlist.sol#L68

contracts/Allowlist.sol#L79

Description

The smart contract does not validate that the `stakingProvider` parameter in the `addStakingProvider` function and `_walletRegistry` parameter in the `initialize` function are non-zero addresses. This means governance could register the zero address as a staking provider and configure the wallet registry to the zero address. Both actions leave the system in an inconsistent state.

Impact

Setting the wallet registry to the zero address would render all registry-related functionality inoperable. Using the zero address as a provider results in non-functional providers that cannot be interacted with.

Recommendation

We recommend adding zero address checks for `stakingProvider` and `_walletRegistry`.

ISSUE#4

Missing Zero-Amount Validation in the `stake` Function

None Fixed

Location

contracts/bridge/RebateStaking.sol#L301

Description

The `stake(uint96 amount)` function allows the `amount` to be zero, which results in a no-op transfer and unnecessary state changes. This can waste gas and create misleading `Staked` events.

Impact

None.

Recommendation

We recommend adding a check to prevent zero-amount stakes.

ISSUE#5

Missing Zero-Weight Check in the `addStakingProvider` Function

None Fixed

Location

contracts/Allowlist.sol#L79

Description

In the case where `parameters.minimumAuthorization == 0` in the `WalletRegistry` smart contract, the function `addStakingProvider` accepts a `weight` of `0` without reverting. This results in a staking provider being added with zero `weight` which can later be re-added with a non-zero `weight`, bypassing the uniqueness check that relies on:

```
 if (info.weight != 0) {
revert StakingProviderAlreadyAdded();
}
```

Impact

When allowed by `WalletRegistry`, this can lead to duplicate additions and inconsistent event history, potentially confusing the monitoring logic.

Recommendation

We recommend adding a check to revert when `weight == 0`.

ISSUE#6

Cache Array Length in Loops for Gas Savings

None Fixed

Location

contracts/bridge/RebateStaking.sol#L216 contracts/bridge/RebateStaking.sol#L283

Description

Functions like `getRebateInRollingWindow` and `cancelRebate` iterate over `stakeInfo.rebates` using `i < stakeInfo.rebates.length` without caching the array length. This causes the array length to be read from storage on every iteration.

Impact

Unnecessary repeated storage reads increase gas costs, especially for users with large `rebates` arrays.

Recommendation

We recommend caching the array length in a local variable before the loop:

```
 uint256 n = stakeInfo.rebates.length;
for (uint256 i = stakeInfo.rollingWindowStartIndex; i < n; i++) {...}
```

This saves gas by avoiding repeated `SLOAD` operations.

ISSUE#7

Use Custom Errors to Save Gas

None Fixed

Location

contracts/bridge/RebateStaking.sol#L94

contracts/bridge/RebateStaking.sol#L314

Description

The smart contract uses `require` statements with revert strings (e.g., “Amount cannot be 0”, “Unstaking already started”). Revert strings consume additional gas because they must be stored in the contract bytecode and returned on failure. This results in higher gas costs for deployments and reverts, especially on frequently called functions.

Impact

None.

Recommendation

We recommend defining and using custom errors instead of string-based require statements.

ISSUE#8

Unrestricted `seize` Function Allows Event Spam

None Not Fixed

Location

contracts/Allowlist.sol#L184

Description

The `seize` function is an external function that emits a `MaliciousBehaviorIdentified` event with arbitrary user-supplied data. Anyone can call it and emit unbounded-length events.

Impact

This enables spam of on-chain logs, which may mislead off-chain monitoring systems or bloat event indexing.

Recommendation

We recommend restricting the `seize` function to the appropriate authorized caller(s).

Section 5.0
Appendix A

At Thesis Defense, we utilize the Immunefi Vulnerability Severity Classification System - v2.3.

Severity	Definition
Critical	• Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results • Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield • Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties • Permanent freezing of funds • Permanent freezing of NFTs • Unauthorized minting of NFTs • Predictable or manipulable RNG that results in abuse of the principal or NFT • Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content) • Protocol insolvency
High	• Theft of unclaimed yield • Theft of unclaimed royalties • Permanent freezing of unclaimed yield • Permanent freezing of unclaimed royalties • Temporary freezing of funds • Temporary freezing NFTs
Medium	• Smart contract unable to operate due to lack of token funds • Enabling/disabling notifications • Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol) • Theft of gas • Unbounded gas consumption
Low	• Contract fails to deliver promised returns, but doesn't lose value
None	• We make note of issues of no severity that reflect best practice recommendations or opportunities for optimization, including, but not limited to, gas optimization, the divergence from standard coding practices, code readability issues, the incorrect use of dependencies, insufficient test coverage, or the absence of documentation or code comments.

Section 6.0
Appendix B

Thesis Defense Disclaimer

Thesis Defense conducts its security audits and other services provided based on agreed-upon and specific scopes of work (SOWs) with our Customers. The analysis provided in our reports is based solely on the information available and the state of the systems at the time of review. While Thesis Defense strives to provide thorough and accurate analysis, our reports do not constitute a guarantee of the project's security and should not be interpreted as assurances of error-free or risk-free project operations. It is imperative to acknowledge that all technological evaluations are inherently subject to risks and uncertainties due to the emergent nature of cryptographic technologies.

Our reports are not intended to be utilized as financial, investment, legal, tax, or regulatory advice, nor should they be perceived as an endorsement of any particular technology or project. No third party should rely on these reports for the purpose of making investment decisions or consider them as a guarantee of project security.

Links to external websites and references to third-party information within our reports are provided solely for the user's convenience. Thesis Defense does not control, endorse, or assume responsibility for the content or privacy practices of any linked external sites. Users should exercise caution and independently verify any information obtained from third-party sources.

The contents of our reports, including methodologies, data analysis, and conclusions, are the proprietary intellectual property of Thesis Defense and are provided exclusively for the specified use of our Customers. Unauthorized disclosure, reproduction, or distribution of this material is strictly prohibited unless explicitly authorized by Thesis Defense. Thesis Defense does not assume any obligation to update the information contained within our reports post-publication, nor do we owe a duty to any third party by virtue of making these analyses available.