certora

Security Assessment Report

Jito Tip Router

January 2025

Prepared for
Jito

certora

Table of content

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform	Comment
jito-tip-router	https://github.com/jito-foundation/jito-tip-router	443368a	Solana	First audit version
jito-tip-router	https://github.com/jito-foundation/jito-tip-router	ac76352	Solana	Second audit version; Operator Client added

Project Overview

This document describes the verification effort of Jito Tip Router using manual code review. The work was undertaken from December 23, 2024, to January 13, 2025.

The following contract list is included in our scope:

```
Unset
program/src/*
core/src/*
meta_merkle_tree/src/*
tip-router-operator-cli/src/*
```

The Certora team performed a manual audit of all the Solana contracts in the scope. During the manual audit process, the team discovered bugs in the Solana contracts code, as listed on the following page.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	6	6	6
High	1	1	1
Medium	2	2	1
Low	1	1	1
Informational	4	4	3
Total	**14**	**14**	**12**

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

Critical-Severity Issues

C-01 Ballot's weight for previous vote doesn't decrement on vote change, allowing attacker to vote multiple times

Severity: Critical	Impact: High	Likelihood: High
Files: core/src/ballot_box.rs	Category: Logical	Status: Fixed

Description: The system allows the operator to change their vote as long as consensus wasn't reached. The issue is that during a vote-change we don't decrement the weight from the ballot of the previous vote. This allows a single operator to vote multiple times, reaching a false consensus.

Impact: The funds from MEV tips would be stolen by a malicious operator

Recommendation: Decrement the weight of the ballot of the previous vote during a vote change.

Customer's response: Recommendation implemented here

Fix review: Looks good

C-02 Ballot Box can be DoS-ed by filling it up with fake ballots

Severity: Critical	Impact: High	Likelihood: High
Files: core/src/ballot_box.rs	Category: DoS	Status: Fixed

Description: The system allows an operator to change their vote as long as consensus wasn't reached. If the vote is for a new ballot that doesn't exist yet in the system - the system would create a new ballot and add it to the `ballot_tallies` votes.
A malicious operator can take advantage of that and fill up the `ballot_tallies` array with wrong ballots, preventing adding the correct ballot to the ballot box.

Consider the following scenario:

- Bob votes for a malicious Merkle root that would assign 50% of the funds to him and the rest would go to the eligible users
- Bob fills up the rest of the `ballot_tallies` array with invalid Merkle roots
- If the operators want to rescue any of the funds they would have to vote to the malicious markle root and give Bob 50% of the funds.

Impact: MEV tips wouldn't be distributed to the eligible users

Recommendation: When a vote is changed - delete the previous ballot if there are no other votes to it. This would ensure the number of ballots is never bigger than the number of operators that have voted.

Customer's response: Recommendation implemented here

Fix review: Looks good

C-03 Snapshot can be created for future epochs, assigning wrong stake weights

Severity: Critical	Impact: High	Likelihood: High
Files: snapshot_vault_operator_ delegation.rs initialize_weight_table.rs	Category: Lack of checks	Status: Fixed

Description: The snapshot process creates a snapshot of the stake-weights at the current epoch.
However, there's no check that prevents creating a snapshot for a future epoch, allowing an attacker to assign the values of the current epoch for future epochs.

Consider the following scenario:

- Bob has a vault with a stake of 1000 tokens
- Bob creates a snapshot for the next 100 epochs and snapshots their vault
- Bob withdraws the funds from the vault
- Bob now enjoys voting power and rewards for epochs where he doesn't have any funds
    - Additionally, new vaults can't be added to the snapshot of those future epochs (since the epoch snapshot contains a finite list of vaults)

Impact: Future epochs would contain the snapshot values of previous epochs, giving vaults and operators voting and reward power that doesn't match their stake at the relevant epoch.

Recommendation: Add a check to ensure we're not snapshotting (nor creating the weight table) future epochs.

Customer's response: Added a new account EpochState which is required for all other accounts in an epoch. It must be created for in its epoch, so that resolves this issue. PR Here

Fix review: Looks good, this seems to resolve the issue

C-04 Snapshotting can't be completed if empty vault-operator delegations exist

Severity: Critical	Impact: High	Likelihood: High
Files: snapshot_vault_operator_delegation.rs	Category: Code	Status: Fixed

Description: Snapshotting requires to snapshot every vault for every operator that exists. Even if there's no delegation account for that operator-vault we need to provide the empty account as a proof.
However, `VaultOperatorDelegation::load()` is being called in any case, meaning that for empty operator-vault delegations this check would always fail, preventing the completion of the snapshot process.

Impact: Snapshotting wouldn't be completed, preventing the distribution of the MEV tips.

Recommendation: Call `VaultOperatorDelegation::load()` only if the account isn't empty.

Customer's response: Recommendation implemented here

Fix review: Looks good

C-05 Reward distribution might be skipped to some operators during routing

Severity: Critical	Impact: High	Likelihood: High
Files: core/src/base_reward_router.rs	Category: Logical	Status: Fixed

Description: Due to CU limits the system allows doing routing in batches, completing only part of the routing in one instruction and completing the rest the next time the instruction is called.
If we didn't complete the routing in the current instruction, we save the state, which includes the last group index and the last vote index (as `starting_vote_index`).
The problem is that when we resume the routing we start from `starting_vote_index` not only for the last group but also for the next groups, effectively skipping any votes before `starting_vote_index` for the next groups.

Impact: Some operators would be deprived of their rewards

Recommendation: Reset `starting_vote_index` to zero after the first iteration.

Customer's response: Recommendation implemented here

Fix review: Looks good

C-06 Voting can be done after routing started, leading to a DoS for the router

Severity: Critical	Impact: High	Likelihood: High
Files: route_base_rewards.rs	Category: Timing	Status: Fixed

Description: Routing can start once consensus is reached, while voting is valid till `valid_slots_after_consensus` slots pass after consensus was reached.
This means that voting can still happen after routing has started. This would lead to an overdistribution of the rewards (since we initially thought there's less stake to the winning ballot, distributing more rewards than we should out of the ncn fee group rewards) that would cause an underflow that would revert the instruction.

Consider the following scenario:

- Consensus is reached
- An attacker sends a small amount of lamports to the base rewards receiver and starts routing, setting the max iteration so that we'll break right before the last vote
- If any more votes come in - any attempt to finalize the routing would lead to an underflow, preventing the distribution of rewards

Impact: Rewards can't be distributed

Recommendation: Don't start routing before voting is finalized.

Customer's response: Recommendation implemented here

Fix review: Looks good

High-Severity Issues

H-01 Base router resumes with wrong rewards, distributing low amounts for operators

Severity: High	Impact: Medium	Likelihood: High
Files: core/src/base_reward_router.rs	Category: Logical	Status: Fixed

Note: *this issue was found by the dev team during the audit*

Description: As part of routing base rewards, we route first to the ncn fee group and then from the ncn fee group to the operators.
When routing to the operators we read the rewards for the group by calling `ncn_fee_group_rewards()` and then distributing it to the operators.
In case that we've reached max iterations then we save the routing state and exit and continue the next time the instruction is called.
The issue is that if routing has stopped in the middle of an ncn fee group then the `ncn_fee_group_rewards()` value has changed (since every time we route the rewards to operators we decrease this value), meaning the next time we resume routing the operators would get less rewards than they should.

Impact: Operators would get less rewards than they should (funds would get stuck in the router till we do another round of routing)

Recommendation: Save the state of `ncn_fee_group_rewards()` for the next time we resume routing

Customer's response: Fixed in PR #35

Fix review: Looks good

Medium-Severity Issues

M-01 `restaking_config` isn't verified, allowing to supply a fake account

Severity: Medium	Impact: Low	Likelihood: High
Files: program/src/register_vault.rs	Category: Account verification	Status: Fixed

Description: The register-vault instruction receives `restaking_config` as one of the accounts, this is used to retrieve the epoch-length which is then passed to the function that checks if the vault's tickets are active.
However, the account isn't checked to be owned by the right program and have the right discriminator. This allows an attacker to send in a malicious account with a wrong epoch-length, bypassing the activity check.

Impact: An inactive vault can be registered to the vault registry

Recommendation: Verify the `restaking_config` account by calling `Config::load()`

Customer's response: Restaking config removed from register_vault in this change

Fix review: Issue resolved

M-02 Attacker can DoS reward distribution by starting routing again

Severity: Medium	Impact: Medium	Likelihood: Medium
Files: route_ncn_rewards.rs route_base_rewards.rs	Category: DoS	Status: Acknowledged

Description: Reward distribution can be done only when the router isn't 'still routing'.
The router starts routing upon any new reward balance (and for the ncn router, even if there's no new balance).
An attacker can take advantage of that and start routing by sending minimal amount of new balance, start routing (setting max iterations to 0, making the attacker's tx fee minimal) and prevent the distribution of rewards.
In order to continue the distribution, somebody would have to finish routing first, but then the attacker can start routing again. This can go on for a while, slowing down and making it difficult to distribute the rewards.

Impact: Rewards distribution would be DoS-ed

Recommendation: Require a minimum new balance in order to start routing, this would exact a cost from the attacker for every round.

Customer's response: Acknowledged, not addressing because mitigation would just require the permissionless cranker to start routing again which is acceptable. This is preferable to adding a minimum balance to routing.

Low-Severity Issues

L-01 Router does one less iteration than it should

Severity: Low	Impact: Very low	Likelihood: High
Files: core/src/base_reward_router.rs core/src/ncn_reward_router.rs	Category: Logical	Status: Fixed

Description: As part of routing the caller can set the maximum number of iterations that we should do via the `max_iterations` parameter.
The problem is that we actually do 1 less iteration than we should, we increase the iteration before the check, and then we quit once we've reached max iterations.
For example, if max iterations is set to 1 the iterations would be reached on the first iteration, and we'd quit before executing that iteration.

JavaScript

```
    iterations = iterations
        .checked_add(1)
        .ok_or(TipRouterError::ArithmeticOverflow)?;

    if iterations >= max_iterations {
        msg!(
            "Reached max iterations, saving state and exiting {}/{}",
            group_index,
            vote_index
        );
        self.save_routing_state(group_index, vote_index);
        return Ok(());
    }
```

Impact: The instruction would do 1 less iteration than it should.

Recommendation: Change the code so it would do one more iteration.

Customer's response: Recommendation implemented here

Fix review: Looks good

Informational-Severity Issues

I-01 CU consumption can be saved by providing hints

Severity: **Informational**	Impact:	Likelihood:
Files: core/src/base_reward_router.rs And more	Category: Optimization	Status: Confirmed

Description: There are multiple instances in the code where we try to find an element within an array.
This can be optimized if we allow the user to provide a 'hint' to the right index of the element, this way we don't have to go over the entire array to find the right element.

```JavaScript
        for route in self.ncn_fee_group_reward_routes.iter_mut() {
             if route.operator.eq(operator) {
                 let rewards = route.rewards(ncn_fee_group)?;
                 route.decrement_rewards(ncn_fee_group, rewards)?;
                 self.decrement_rewards_processed(rewards)?;

                 return Ok(rewards);
             }
         }
```

Recommendation: Allow the user to provide a hint to the index of the element we're searching for.

I-02 Inconsistency in the way the restaking program is treated

Severity: **Informational**	Impact:	Likelihood:
Files: Distribute_ncn_vault_rewards.rs Snapshot_vault_operator_delegation.rs And more	Category: Design	Status: Confirmed

Description: In some instructions (e.g. `snapshot_vault_operator_delegation`) the provided restaking and vault programs are checked to be the specific restaking program that's currently deployed, while in other instructions (e.g. `distribute_ncn_vault_rewards`) it isn't checked. This doesn't have any significant effect, since it's checked to match the provided ncn, but it's better to have a consistent policy regarding whether we allow other ncn-s to be created.

```JavaScript
    if restaking_program.key.ne(&jito_restaking_program::id()) {
        msg!("Incorrect restaking program ID");
        return Err(ProgramError::InvalidAccountData);
    }
```

Recommendation: Make the program consistent by either allowing a different restaking program in all instructions or checking it's the specific program in all of them.

Customer's response: Fixed in PR #47

Fix review: Fix confirmed

I-03 Redundant discriminator check in `admin_set_new_admin`

Severity: **Informational**	Impact:	Likelihood:
Files: admin_set_new_admin.rs	Category: Redundant code	Status: Confirmed

Description: At `admin_set_new_admin` there's a check to verify the account has the right discriminator, this check is redundant because the discriminator is checked inside `NcnConfig::load()`

```
JavaScript
    NcnConfig::load(program_id, ncn_account.key, config, true)?;
    Ncn::load(restaking_program.key, ncn_account, false)?;

    let mut config_data = config.try_borrow_mut_data()?;
    if config_data[0] != NcnConfig::DISCRIMINATOR {
        return Err(ProgramError::InvalidAccountData);
    }
```

Recommendation: Remove the redundant check

Customer's response: Fixed in PR #40

Fix review: Fix confirmed

I-04 Out of bounds check is off by one at epoch snapshot

Severity: **Informational**	Impact:	Likelihood:
Files: core/src/epoch_snapshot.rs	Category: Error emission	Status: Confirmed

Description: At `EpochSnapshot::insert_vault_operator_stake_weight()` we revert if the number of registered vaults is greater than max vaults.
However, this check is off by one, because it's called before we add the current vault, so also when the amount registered is exactly max vaults - we'd be out of bounds when we register the new vault.

```JavaScript
        if self.vault_operator_delegations_registered() > MAX_VAULTS as u64 {
            return Err(TipRouterError::TooManyVaultOperatorDelegations);
        }
```

Impact: A regular out-of-bounds error would be returned instead of the verbose custom error.

Recommendation: Replace `>` with `>=` so that we'd revert also when it's equal to max vaults.

Customer's response: Fixed in PR #36

Fix review: Fix confirmed

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.