OFFSIDE LABS

Jito Tip Router

Smart Contract Security Assessment

January 2025

Prepared for:

Jito Foundation

Prepared by:

Offside Labs

Yao Li

Ronny Xing

Contents

1 About Offside Labs

Offside Labs is a leading security research team, composed of top talented hackers from both academia and industry.

We possess a wide range of expertise in modern software systems, including, but not limited to, *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. We are also at the forefront of innovative areas like *cryptocurrencies* and *blockchain technologies*. Among our notable accomplishments are remote jailbreaks of devices such as the **iPhone** and **PlayStation 4**, and addressing critical vulnerabilities in the **Tron Network**.

Our team actively engages with and contributes to the security community. Having won and also co-organized *DEFCON CTF*, the most famous CTF competition in the Web2 era, we also triumphed in the **Paradigm CTF 2023** within the Web3 space. In addition, our efforts in responsibly disclosing numerous vulnerabilities to leading tech companies, such as *Apple*, *Google*, and *Microsoft*, have protected digital assets valued at over **$300 million**.

In the transition towards Web3, Offside Labs has achieved remarkable success. We have earned over **$9 million** in bug bounties, and **three** of our innovative techniques were recognized among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs

2 Executive Summary

Introduction

Offside Labs completed a security audit of *Jito Tip Router* smart contracts, starting on Dec 24, 2024, and concluding on Jan 10, 2024.

Project Overview

The Jito Tip Router NCN (Node Consensus Network) is a decentralized system designed to enhance MEV (Maximal Extractable Value) tip distribution for validators using the Jito-Solana client, moving away from the current centralized process managed by Jito Labs. It operates on Jito Restaking Programs, allowing operators to run a custom client that utilizes stakes from Vaults with Liquid Staking Tokens (LSTs) and Jito Tokens (JTO). Each epoch, the system determines the relative weights of vault assets, snapshots the stake of operators and vaults, aggregates results to reach consensus on a Meta Merkle Root for tip distributions, and uploads individual Merkle roots to validators. Additionally, it generates protocol fees that are distributed among the Jito DAO, operators, and vaults in JitoSOL, aiming to improve efficiency and transparency in the tip distribution process.

Audit Scope

The assessment scope contains mainly the smart contracts of the jito-tip-router program for the *Jito Tip Router* project.

The audit is based on the following specific branches and commit hashes of the codebase repositories:

- Jito Tip Router
  - Codebase: `https://github.com/jito-foundation/jito-tip-router`
  - Commit Hash: 443368abfdea0e35ec6dc16cc27ea86de1d31d9f

We listed the files we have audited below:

- Jito Tip Router
  - core/src/*.rs
  - program/src/*.rs
  - meta_merkle_tree/src/*.rs
  - tip_router_operator_cli/src/*.rs

Findings

The security audit revealed:

- 1 critical issues
- 2 high issues
- 8 medium issues
- 6 low issues
- 4 informational issues

Further details, including the nature of these issues and recommendations for their remediation, are detailed in the subsequent sections of this report.

3 Summary of Findings

ID	Title	Severity	Status
01	Operator Can Vote Multiple Times to Manipulate or Disrupt Consensus	Critical	Fixed
02	Early Snapshot of Future Epoch May Break Consensus	High	Fixed
03	CloseEpochAccount Instruction Not Check NCN for Accounts	High	Fixed
04	Inconsistent NCN Vault Count Could Cause DoS in initialize_weight_table Instruction	Medium	Fixed
05	Unverified Account Could Cause DoS in initialize_weight_table Instruction	Medium	Fixed
06	Redundant Verification Could Cause DoS in snapshot_vault_operator_delegation Instruction	Medium	Fixed
07	Submitting Invalid meta_merkle_root Should Be Prohibited	Medium	Fixed
08	Snapshot Delegation Instruction Not Ensure Vault's Updated Status	Medium	Fixed
09	Vaults and Operators in the Cooling Down Phase Should Also Be Valid	Medium	Fixed
10	Newly Initialized ncn_operator_state After epoch_snapshot Creation Could Preempt Operator Slot	Medium	Fixed
11	Inaccurate Tip Distributor Delegated Stake Accounting	Medium	Acknowledged
12	update_fee_config Lacks Validation for the New Fee Config	Low	Fixed
13	Leftover Dust in route_base_reward Instruction	Low	Fixed
14	Routing Process May Never Complete	Low	Fixed
15	Other Accounts Can Still Be Reinitialized During EpochState Closure	Low	Fixed
16	Leftover Dust in MerkleTree Generation	Low	Acknowledged
17	ClaimWithPayer Instruction Lacks Validation for tip_distribution_account	Low	Acknowledged
18	block_engine_fee_bps Not Checked in admin_set_config_fees Instruction	Informational	Fixed
19	Incorrect Account Writable Check	Informational	Fixed
20	Inconsistent Checks for Block Engine Fee	Informational	Fixed
21	Missing Check for External Program ID in Multiple Instructions	Informational	Fixed

4 Key Findings and Recommendations

4.1 Operator Can Vote Multiple Times to Manipulate or Disrupt Consensus

Severity: Critical	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `cast_vote` instruction, an operator can vote for a `meta_merkle_root` . However, the operator can vote multiple times before consensus is reached.

Currently, previous votes are not removed, allowing an operator to vote for the same ballot multiple times and accumulate multiple weights, which can break and manipulate consensus.

Additionally, an operator can vote for different ballots to fill all the ballot tally slots, potentially leading to a DoS attack. An attacker only needs to call `cast_vote` 256 times to permanently block voting in this epoch.

Impact

A malicious operator can vote multiple times to manipulate consensus or cause a DoS attack.

Recommendation

Revise the `increment_or_create_ballot_tally` function to use two iterations: first, check if there are existing tallies with the same `meta_merkle_root` ; if not, find an invalid slot to overwrite.

Mitigation Review Log

Fixed in PR-42.

4.2 Early Snapshot of Future Epoch May Break Consensus

Severity: High	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

The epoch parameter is introduced in snapshot related instructions to track the stake details of all operators. During the snapshot process, the counts of vaults and operators, along with the weights of mints, are recorded to prepare for the subsequent voting process.

However, the implementation does not impose adequate constraints on the epoch parameter. This allows anyone to take a snapshot for a future epoch using the current state. Since the state may significantly change by the time the future epoch arrives, the early snapshot could lead to inconsistencies and potentially break consensus.

For example, in the `InitializeEpochSnapshot` instruction, using the current `vault_count` and `operator_count` to initialize the future epoch snapshot may disrupt the consensus when that epoch is actually finalized.

```
    *epoch_snapshot_account = EpochSnapshot::new(
        ncn.key,
        ncn_epoch,
        epoch_snapshot_bump,
        current_slot,
        &ncn_fees,
        operator_count,
        vault_count,
    );
```

program/src/initialize_epoch_snapshot.rs#L109-L117

Impact

An attacker can take a epoch snapshot for a future epoch using the current state and then apply operators with the same count as in the current state. Once the operators are applied, the attacker can preempt all operator slots. This allows the attacker to manipulate the consensus process, leveraging operators with minor stakes to achieve consensus improperly.

The same issue also appears in the following instructions:

1. The `snapshot_vault_operator_delegation` instruction can write the current vault's delegation to the operator into the snapshot for future epochs.
2. The `realloc_operator_snapshot` instruction may write outdated `operator_fee_bps` to the `operator_snapshot_account` and incorrect `vault_operator_delegations` and `stake_weight` to the `epoch_snapshot_account`.

Recommendation

Add constraints to ensure that the snapshot process only allows epochs less than or equal to the current epoch.

Mitigation Review Log

Jito Team: 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-43

Offside Labs: Fixed.

4.3 CloseEpochAccount Instruction Not Check NCN for Accounts

Severity: High	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

The `CloseEpochAccount` instruction will close the `account_to_close` account and send the rent to the `account_payer` PDA.

Some `account_to_close` accounts, like `WeightTable` or `EpochSnapshot` , do not check for NCN.

Impact

An attacker could use config from a different NCN to close accounts related to the tip router NCN, and send the rent to the `account_payer` PDA for the NCN controlled by the attacker.

This may lead to tip router NCN related accounts being closed prematurely and their rent being stolen (permanently occupied).

Recommendation

Check NCN public key for `account_to_close` .

Mitigation Review Log

Fixed in PR-77

4.4 Inconsistent NCN Vault Count Could Cause DoS in initialize_weight_table Instruction

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `initialize_weight_table` instruction, the following check ensures that all valid vaults in NCN are registered in the `vault_registry` :

```rust
    if vault_count != vault_registry_count {
        msg!("Vault count does not match supported mint count");
        return Err(ProgramError::InvalidAccountData);
    }
```

program/src/initialize_weight_table.rs#L51-L54

However, in the restaking program, the `vault_count` of NCN does not align with the actual valid active vaults. When a new vault is registered to the NCN, the `vault_count` will be increased first in the `initialize_ncn_vault_ticket` IX. But the corresponding vault can only transition to a warmup state in the same epoch. Furthermore, for another scenario, a vault that becomes inactive before this program is deployed cannot be registered but is counted in the `vault_count` of NCN. These inconsistencies can lead to the above check failing, potentially causing a Denial of Service for the instruction.

Impact

The discrepancies between the `vault_count` of NCN and the actual valid vaults can cause the check in the `initialize_weight_table` instruction to fail, resulting in a DoS on that instruction.

Recommendation

Correct the tracking of `vault_count` in the restaking NCN at appropriate points in the code to maintain consistency.

Mitigation Review Log

Fixed in PR-52.

4.5 Unverified Account Could Cause DoS in initialize_weight_table Instruction

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

In the `initialize_weight_table` instruction, the following check ensures that all valid vaults in NCN are registered in the `vault_registry` :

```rust
    if vault_count != vault_registry_count {
        msg!("Vault count does not match supported mint count");
        return Err(ProgramError::InvalidAccountData);
    }
```

program/src/initialize_weight_table.rs#L51-L54

However, in the `register_vault` instruction, the `restaking_config` account is loaded directly without checking if it is a valid PDA derived from the JITO restaking program.

Impact

This oversight allows an attacker to provide a forged account with a manipulated `epoch_length` . This would allow an attacker to bypass the vault status checks in the `register_vault` instruction, thereby registering an invalid vault.

```rust
    if !vault_ncn_ticket
        .state
        .is_active_or_cooldown(slot, epoch_length)
    {
        msg!("Vault NCN ticket is not enabled");
        return Err(ProgramError::InvalidAccountData);
    }
```

program/src/register_vault.rs#L44-L50

This attack can eventually result in the check in the `initialize_weight_table` instruction failing, leading to a Denial of Service attack on that instruction.

Recommendation

Implement verification for the `restaking_config` account to ensure it is a valid PDA of the JITO restaking program.

Mitigation Review Log

Fixed in PR-52.

4.6 Redundant Verification Could Cause DoS in snapshot_vault_ operator _delegation Instruction

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `snapshot_vault_operator_delegation` instruction, the following check allows an uninitialized `vault_operator_delegation` with 0 stake weight:

```
    if !vault_operator_delegation.data_is_empty() {
        VaultOperatorDelegation::load(
            vault_program.key,
            vault_operator_delegation,
            vault,
            operator,
            false,
        )?;
    }
```

program/src/snapshot_vault_operator_delegation.rs#L58-L66

However, in lines program/src/snapshot_vault_operator_delegation.rs#L48-L54, the same verification is performed without checking for empty data.

Impact

The redundant verification may lead to a Denial of Service if an uninitialized `vault_operator_delegation` exists.

Recommendation

Remove the redundant verification to enhance efficiency and prevent potential DoS vulnerabilities.

Mitigation Review Log

Fixed in PR-43.

4.7 Submitting Invalid meta_merkle_root Should Be Prohibited

Severity: Medium	**Status: Fixed**
Target: Smart Contract	**Category: Data Validation**

Description

An evil operator can directly submit an all-zero `meta_merkle_root` to occupy the first (or the first uninitialized) index of `ballot_tallies`. This is because the `increment_or_create_ballot_tally` method only compares whether the ballots are equal without verifying their validity.

```
    for tally in self.ballot_tallies.iter_mut() {
        if tally.ballot.eq(ballot) {
            tally.increment_tally(stake_weights)?;
```

core/src/ballot_box.rs#L401-L403

Subsequent honest operators submitting the correct `meta_merkle_root` will overwrite the first invalid ballot.

Impact

Since the `ballot_index` of evil operator votes is also the winning ballot, the `winning_reward_stake_weight` (denominator) is less than the total stake weight (numerator) of all votes with the winning ballot during routing base rewards. This will cause a failure due to insufficient reward pool. The base reward routing process will never complete, resulting in DoS.

Recommendation

Forbidden to submit an invalid `meta_merkle_root` with all zero bytes.

Mitigation Review Log

Fixed in PR-42.

4.8 Snapshot Delegation Instruction Not Ensure Vault's Updated Status

Severity: Medium	**Status: Fixed**
Target: Smart Contract	**Category: Data Validation**

Description

The `snapshot_vault_operator_delegation` instruction does not check whether the vault account has been fully updated in the current epoch. This can lead to the `OperatorSnapshot::calculate_total_stake_weight` method using an outdated `vault_operator_delegation_account` to calculate the operator's `total_security` for that vault.

Impact

This can lead to un-updated state changes in the vault potentially disrupting consensus.

For example, a vault admin can cool down a delegation for an operator in the current epoch, but the operator's `total_security` will not change in the current and the next two epochs. After two epochs, the `snapshot_vault_operator_delegation` instruction of the tip router is called first. Since the vault is not fully updated at this time, the `cooling_down_amount` will be included in the `total_security`. Then, the `crank_vault_update_state_tracker` instruction of the vault program is called to deduct the `cooling_down_amount` and the `add_delegation` instruction is called to re-delegate this amount to another operator. This will result in the vault's stake weight proportion being doubled.

Recommendation

Use `Vault::check_update_state_ok` or `Vault::is_update_needed` function to ensure the vault is fully updated before calculating the stake weight for this vault.

Mitigation Review Log

Fixed in PR-41.

4.9 Vaults and Operators in the Cooling Down Phase Should Also Be Valid

Severity: Medium	**Status: Fixed**
Target: Smart Contract	**Category: Logic Error**

Description

In `ReallocOperatorSnapshot` and `SnapshotVaultOperatorDelegation` IXs, the current vault or operator is checked to determine if it is active for the NCN. Only when the status is active, the vault or operator is considered valid and recorded in the snapshot.

```
let ncn_operator_okay = ncn_operator_state_account
    .ncn_opt_in_state
    .is_active(current_slot, ncn_epoch_length);

let operator_ncn_okay = ncn_operator_state_account
    .operator_opt_in_state
    .is_active(current_slot, ncn_epoch_length);
```

program/src/realloc_operator_snapshot.rs#L84-L90

However JITO restaking program always enforces changes in the participation status of vaults/operators in the next or subsequent epoch. Before this, the vault and operator will be in a cooling down state. In this state, they should also be considered valid.

Impact

This disrupts the consensus of the current epoch. For example, a vault admin can first use `SnapshotVaultOperatorDelegation` to record the `total_security` for certain operators, then exit the NCN. The remaining operators will not be able to record their own `total_security` of the vault in the snapshot of the current epoch because their `is_active` status becomes false.

Recommendation

Use `SlotToggle::is_active_or_cooldown` instead of `is_active`.

Mitigation Review Log

Fixed in PR-53.

4.10 Newly Initialized ncn_operator_state After epoch_snapshot Creation Could Preempt Operator Slot

Severity: Medium | **Status: Fixed**

Target: Smart Contract | **Category: Logic Error**

Description

During the execution of the `InitializeEpochSnapshot` instruction, the `epoch_snapshot` is initialized using the current values of `vault_count` and `operator_count` as shown:

```
    *epoch_snapshot_account = EpochSnapshot::new(
        ncn.key,
        ncn_epoch,
        epoch_snapshot_bump,
        current_slot,
        &ncn_fees,
        operator_count,
        vault_count,
    );
```

program/src/snapshot_vault_operator_delegation.rs#L58-L66

However, if the `ncn_operator_admin` initializes a new `ncn_operator_state` after the `epoch_snapshot` initialization, this newly created pair of `operator` and `ncn_operator_state` can still be used to initialize an `operator_snapshot`. This results in the new operator being registered in the `epoch_snapshot`.

Since the new operator was not included in the initial `operator_count` used to create the `epoch_snapshot`, it forces preemption of another operator's slot.

Impact

If the `ncn_operator_admin` initializes an `ncn_operator_state` after the `epoch_snapshot` is already set up, it could cause unintended preemption and disrupts the vote process and reward distribution process.

Recommendation

Only allow operator registration with `ncn_operator_state.index < epoch_snapshot.operator_count`. This does not disrupt the consensus because newly added operators will be in a warmup state.

Mitigation Review Log

Fixed in PR-54.

4.11 Inaccurate Tip Distributor Delegated Stake Accounting

Severity: Medium	**Status: Acknowledged**
Target: Blockchain	**Category: Logic Error**

Description

The `jito-solana` and `tip-router-operator-cli` use the following code to calculate each delegation stake of the voter in the current epoch. This stake, referred to as `lamports_delegated`, is then used to calculate the reward distribution for each delegation:

- jito-solana: `tip-distributor/src/stake_meta_generator_workflow.rs#L315-L351`
- tip-router-operator-cli: `tip-router-operator-cli/src/stake_meta_generator.rs#L287-L322`

However, the `group_delegations_by_voter_pubkey` function directly uses the `Delegation.stake` as `lamports_delegated`, which is the total amount of delegated stake, not the effective stake. Depending on the relationship between the epoch and the `activation_epoch` and `deactivation_epoch`, this `Delegation.stake` may also include activating stake and withdrawable stake in addition to the effective stake.

Refer to the standard Solana validator staking impl `sdk/program/src/stake/state.rs#L661-L670` and documentation `https://docs.anza.xyz/consensus/stake-delegation-and-rewards#stake-warmup-cooldown-withdrawal`

Impact

In standard Solana validator staking, only effective stake can earn rewards or be exposed to slashing. Since the effective stake of some delegations may be overestimated when generating the tip distribution merkle tree, this can lead to unfair reward distribution.

More seriously, during the stake cool down phase, `Delegation.stake` also includes withdrawable stake. This allows a malicious delegator to immediately withdraw stake after the epoch ends, participate in other Defi to earn additional returns, and then re-stake before the current epoch ends. This way, they can enjoy Jito's reward distribution simultaneously.

This undermines the security and stability of staking and harms the economic model.

Recommendation

Only use effective skate when calculating rewards for each delegation.

Mitigation Review Log

Jito Team: Acknowledged, considering options.

4.12 update_fee_config Lacks Validation for the New Fee Config

Severity: Low	**Status: Fixed**
Target: Smart Contract	**Category: Data Validation**

Description

The `update_fee_config` method calls `self.check_fees_okay(current_epoch)` at the end to validate the fee config's validity. However, this check only verifies the `current_fees` and does not check the newly updated `updatable_fees` . Since `updatable_fees` will automatically take effect when the `activation_epoch` is reached, the new fee config lacks a validity check.

Recommendation

Use `next_epoch` , which is `current_epoch + 1` , as the parameter to call the `check_fees_okay` method to ensure the validity of the new config.

Mitigation Review Log

Fixed in PR-55.

4.13 Leftover Dust in route_base_reward Instruction

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Precision

Description

In the `route_ncn_fee_group_rewards` process of the `route_base_reward` instruction, rewards are distributed based on stake weight. However, the rewards are calculated using floor division, which may result in leftover dust that is not handled during the process:

```
    let precise_ncn_route_reward = precise_rewards_to_process
        .checked_mul(&precise_ncn_route_reward_stake_weight)
        .and_then(|x|
↪   x.checked_div(&precise_winning_reward_stake_weight))
        .ok_or(TipRouterError::ArithmeticOverflow)?;
```

core/src/base_reward_router.rs#L380-L383

Impact

The distribution process may lead to unallocated dust.

Recommendation

Route the leftover dust to the DAO group.

Mitigation Review Log

Fixed in PR-56.

4.14 Routing Process May Never Complete

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

Regardless of the status returned by `still_routing`, if the input `max_iterations` is 0, the `route_ncn_rewards` instruction will always reset the routing status to 0 each time it is called, thereby starting a new routing process.

The `route_base_rewards` instruction has the same issue, but resetting the routing status requires the `base_reward_receiver` to have a balance that has not been accounted for in the `reward_pool`. An attacker can transfer 1000 lamports to the `base_reward_receiver` and then call `route_base_rewards` to reset the routing status.

Impact

This is a nearly zero-cost griefing DoS attack that can cause the router to remain in the routing process indefinitely, preventing the distribution of rewards.

Recommendation

For the `route_base_rewards` and `route_ncn_rewards` instructions, it is recommended to add a minimum threshold for routing rewards in the `BaseRewardRouter::route_reward_pool` and `NcnRewardRouter::route_operator_rewards` methods. For example, 1e6 lamports. When the new unaccounted rewards are below this threshold, these rewards should be directly accounted as leftover to the default receiver, without starting a new routing cycle.

For the `NcnRewardRouter::route_reward_pool` method in the `route_ncn_rewards` instruction, it is recommended to check if `rewards_to_process` is 0 before starting the loop. If it is 0, skip the routing cycle directly.

Mitigation Review Log

Jito Team: Recommendation implemented in PR-57.

Offside Labs:

The mitigation implements two items from the recommendation:

1. It restricts the input `max_iterations` to a minimum of 1.
2. It skips the loop when `rewards_to_process` is 0.

Although the recommendation about adding a rewards minimum threshold isn't implemented, we believe the current restrictions are sufficient to defend against potential griefing DoS attacks. Therefore, we consider this issue resolved.

4.15 Other Accounts Can Still Be Reinitialized During EpochState Closure

Severity: Low	**Status: Fixed**
Target: Smart Contract	**Category: Logic Error**

Description

Since `EpochMarker` is only created when closing `EpochState`, it can not protect other preceding accounts (such as `WeightTable`) from being re-initialized after being closed.

This allows attackers to reinitialize these accounts to occupy the rent in `AccountPayer` .

Recommendation

Check if the current `EpochState` account is in the closure process each time an account is initialized.

Mitigation Review Log

Fixed in PR-77.

4.16 Leftover Dust in MerkleTree Generation

Severity: Low	**Status: Acknowledged**
Target: Smart Contract	**Category: Precision**

Description

In the `TreeNode::vec_from_stake_meta` process, rewards are distributed based on delegated amount. However, the rewards are calculated using floor division, which may result in leftover dust that is not handled during the process:

```
        let reward_amount = u64::try_from(
            (amount_delegated.checked_mul(remaining_total_rewards as u128))
                .ok_or(MerkleRootGeneratorError::CheckedMathError)?
                .checked_div(total_delegated)
                .ok_or(MerkleRootGeneratorError::CheckedMathError)?,
        )
        .map_err(|_| MerkleRootGeneratorError::CheckedMathError)?;
```

meta_merkle_tree/src/generated_merkle_tree.rs#L232-L238

Impact

The distribution process may lead to unallocated dust.

Recommendation

Recalculate and reallocate dust after the process.

Mitigation Review Log

Jito Team: Acknowledged, considering options.

4.17 ClaimWithPayer Instruction Lacks Validation for tip_ distribution_ account

Severity: Low	Status: Acknowledged
Target: Smart Contract	Category: Data Validation

Description

The `ClaimWithPayer` instruction can create a `ClaimStatus` for any `tip_distribution_account` . The rent for the `ClaimStatus` account needs to be locked for 10 epochs. An attacker can set the claim amount of the `tip_distribution_account` to the rent amount of the `tip_distribution_account` and directly withdraw the rent from the `tip_distribution_account` .

Impact

An attacker can force all lamports in the `claim_status_payer` to be locked for 10 epochs at nearly zero cost. During this time, the `ClaimWithPayer` instruction will be unusable. This forces the administrator to deposit more lamports to maintain normal functionality, but the attacker can continuously lock these additional balances. Although this does not cause actual loss, it results in a significant amount of SOL being occupied and locked.

Recommendation

Ensure the `merkle_root_upload_authority` of the `tip_distribution_account` is the `ncn_config` of the tip router NCN.

Mitigation Review Log

Jito Team: Won't fix, we have enough lamports for 10 epochs.

4.18 Informational and Undetermined Issues

block_engine_fee_bps Not Checked in admin_set_config_fees Instruction

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Logic Error

In the `admin_set_config_fees` instruction, the parameter `block_engine_fee_bps` is not validated against the `MAX_FEE_BPS` limit. This oversight could lead to unintended configurations that exceed acceptable fee limits.

Fixed in PR-58.

Incorrect Account Writable Check

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Data Validation

There are multiple incorrect account writable checks:

- program/src/distribute_base_ncn_reward_route.rs#L48-L48
- program/src/distribute_ncn_vault_rewards.rs#L36-L36
- program/src/admin_set_tie_breaker.rs#L23-L23
- program/src/initialize_operator_snapshot.rs#L43-L43

Fixed in PR-59.

Inconsistent Checks for Block Engine Fee

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Data Validation

In the `InitializeConfig` instruction, the first check program/src/initialize_ncn_config.rs#L58 requires `block_engine_fee_bps` to be less than `MAX_FEE_BPS` , but the internal check in `FeeConfig::new` core/src/fees.rs#L43 requires it to be less than or equal to.

Fixed in PR-58.

Missing Check for External Program ID in Multiple Instructions

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Data Validation

In several instructions, external program IDs are used to verify Program Derived Address (PDA) access. However, implementations vary:

1. Program ID is provided as pub key of account info and checked.
2. Program ID is provided as pub key of account info but not checked.
3. Program ID is provided as a constant.

The second implementation lacks checks for external program IDs, allowing an attacker to supply fraudulent accounts. The affected instructions include, but are not limited to:

- `initialize_ncn_config`
- `register_vault`
- `cast_vote`

- `set_merkle_root`
- `admin_*`

  Recommendation: Implement checks for external program IDs in all relevant instructions to enhance security and prevent the use of fake accounts.

  Fixed in PR-47.

5 Disclaimer

This audit report is provided for informational purposes only and is not intended to be used as investment advice. While we strive to thoroughly review and analyze the smart contracts in question, we must clarify that our services do not encompass an exhaustive security examination. Our audit aims to identify potential security vulnerabilities to the best of our ability, but it does not serve as a guarantee that the smart contracts are completely free from security risks.

We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster the security of their smart contracts.

It is important to note that the scope of our audit is limited to the areas outlined within our engagement and does not include every possible risk or vulnerability. Continuous security practices, including regular audits and monitoring, are essential for maintaining the security of smart contracts over time.

Please note: we are not liable for any security issues stemming from developer errors or misconfigurations at the time of contract deployment; we do not assume responsibility for any centralized governance risks within the project; we are not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By using this report, the client acknowledges the inherent limitations of the audit process and agrees that our firm shall not be held liable for any incidents that may occur subsequent to our engagement.

This report is considered null and void if the report (or any portion thereof) is altered in any manner.

OFFSIDE LABS

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs