Stake Deposit Interceptor Security Assessment & Formal Verification

Jito Foundation

December 2024

Prepared for
Jito Labs

certora Jito Foundation

Table of content

Project Summary **3**
- Project Scope 3
- Project Overview 3
- Findings Summary 4
- Severity Matrix 4

Detailed Findings **5**
- High-Severity Issues 5
  - H-01 Cooldown end time might overflow 5
- Medium-Severity Issues 6
  - M-01 Max fee bps is not applied when updating the deposit stake authority 6
  - M-02 Precision Loss in Fee Calculation Due to Multiple Integer Divisions 7
- Low-Severity Issues 8
  - L-01 When a receipt is closed/claimed with a transferred owner, the rent will not be paid to the former owner 8
  - L-02 Lack of validation might allow using a stake pool not allowed by the program 9
- Informational Severity Issues 9
  - I-01 process_claim_pool_tokens does not check that the vaults are distinct 9
  - I-02 Usage of saturating_sub in process_claim_pool_tokens 10
  - I-03 calculate_fee_amount may return fee amount greater than the total_amount 11

Formal Verification **12**
- General Assumptions and Simplifications 12
- Verification Notations 12
- Formal Verification Properties 13
- P-01. Fee is only paid by the redeemer 13
- P-2. Vault always has funds to cover its obligation 13
- P-3. Fee and deposit parameters of DepositReceipt cannot be modified once created 14
- P-4. Tickets can always be redeemed 15
- P-5. The initial_fee_bps must always be less than 10_000 15
- P-6. Rules on fee assessment 16

Disclaimer **18**

About Certora **18**

certora

Jito Foundation

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform	Comment
Stake-deposit-interceptor on **Exo Tech**	https://github.com/exo-tech-xyz/stake-deposit-interceptor	82e0c41	Solana	Audit version
Stake-deposit-interceptor on **Exo Tech**	https://github.com/exo-tech-xyz/stake-deposit-interceptor	b2c4075	Solana	Fix version
Stake-deposit-interceptor on **Jito Foundation**	https://github.com/jito-foundation/stake-deposit-interceptor	62ca755	Solana	2nd round of audit fixes on **Jito Foundation**

Project Overview

This document describes the verification effort of Stake Deposit Interceptor using manual code review and Certora Prover. The work was undertaken from November 24, 2024, to December 13, 2024.

The Certora Prover demonstrated that the implementation of the Solana contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solana contracts in the repo. During the verification process and the manual audit, the Certora team discovered bugs in the Solana contracts code, as listed on the following page.

We have verified the fixes in the commit hash of the fix version and reran the FV rules to verify that all rules still hold. On Dec. 24, 2024 s second round of audit fixes was provided, we reran all FV rules and verified the fixed of the informational-severity issues.

certora

Jito Foundation

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	0	0	0
High	1	1	1
Medium	2	2	2
Low	2	2	1
Informational	3	3	3
Total	**8**	**8**	**7**

Severity Matrix

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

Detailed Findings

High-Severity Issues

H-01 Cooldown end time might overflow

Severity: High	Impact: High	Likelihood: Medium
Files: processor.rs	Category:	Status: Fixed

Description:

There's a potential integer overflow vulnerability where converting a large u64 value (deposit_time + cool_down_seconds) to i64 could result in a negative number due to the lack of proper bounds checking. This is particularly concerning since cool_down_seconds can be arbitrarily large.

The code computes the expiration time of the cooldown period by adding deposit_time and cool_down_seconds. However, there's a critical issue in how this computation is performed:

```
Unset
let cool_down_end_time = u64::from(deposit_receipt.deposit_time)
    .checked_add(deposit_receipt.cool_down_seconds.into())
    .expect("overflow") as i64;
```

The code first adds two u64 values (which can be up to 2^64 - 1), then attempts to cast the result to i64 (which can only hold values up to 2^63 - 1). This cast is unsafe because if the sum

is greater than i64::MAX (2^63 - 1), the value will overflow and become negative when cast to i64.

Recommendation:

The fix is

```
Unset
            let cool_down_end_time: i64 = u64::from(deposit_receipt.deposit_time)
                .checked_add(deposit_receipt.cool_down_seconds.into())
                .expect("overflow").try_into().unwrap();
```

it uses try_into instead of as to ensure that conversion fails on overflow

Customer's response: resolved in commit

Fix Review: The issue has been resolved.

Medium-Severity Issues

M-01 Max fee bps is not applied when updating the deposit stake authority

Severity: Medium	Impact: Low	Likelihood: High
Files: processor.rs	Category: Logical Error	Status: Fixed

Description:

When calling process_init_stake_pool_deposit_stake_authority, there's a check whether initial_fee_bps > DepositReceipt::FEE_BPS_DENOMINATOR.

However, the same check is not applied when calling process_update_deposit_stake_authority

Recommendation:

Apply the same check when updating the deposit stake authority.

Customer's response: resolved in commit

Fix Review: The issue has been resolved.

M-02 Precision Loss in Fee Calculation Due to Multiple Integer Divisions

Severity: Medium	Impact: Low	Likelihood: High
Files: state.rs	Category:	Status: Fixed

Description:

The current fee calculation performs two sequential divisions: first dividing by cool_down_seconds and then by FEE_BPS_DENOMINATOR (10,000). This approach involves an intermediate integer division which can lead to precision loss due to truncation of fractional parts.

Recommendation:

Combine the two divisors (cool_down_seconds and FEE_BPS_DENOMINATOR) into a single denominator before performing the division.

Customer's response: resolved in commit

Fix Review: The issue has been resolved.

Low-Severity Issues

L-01 When a receipt is closed/claimed with a transferred owner, the rent will not be paid to the former owner.

Severity: Low	Impact: Low	Likelihood: Low
Files: processor.rs	Categories:	Status: Confirmed

Description:

As part of the system's design, it is possible to change the receipt owner by calling `process_change_deposit_receipt_owner`. However, no variable tracks the rent payer for the receipt.

Consequently, if the receipt is claimed and closed, the rent will go to the new owner, not the one who paid the rent and not the new owner.

This issue was uncovered by rule P-01 Fee is paid only by the redeemer.

Recommendation:

Ensure that when the receipt is claimed and closed

Customer's response: Acknowledged, will not be fixed

L-02 Lack of validation might allow using a stake pool not allowed by the program

Severity: Low	Impact:	Likelihood:
Files:	Categories:	Status: Fixed

Description:

When calling `process_deposit_stake`, there's no check whether the stake pool program is the one allowed by the authority.

Recommendation:

Add validation to ensure that the stake pool program is allowed by the authority.

Customer's response: resolved in commit

Fix Review: The issue has been resolved.

Informational Severity Issues

I-01 `process_claim_pool_tokens` does not check that the vaults are distinct.

Description: `process_claim_pool_tokens` transfers tokens from `vault_token_account_info` to `destination_token_acc_info` and `fee_token_account_info`. It is good practice to check that

there are no self-transfers. Since process_claim_pool_tokens does not check that these vaults are distinct, it leaves open a possibility of a self-transfer.

Recommendation: Add the following checks:

```
Unset
vault_token_account_info.key != destination_token_acc_info.key

vault_token_account_info.key != fee_token_account_info.key
```

Customer's response: resolved in commit

Fix Review: The issue has been resolved.

I-02 Usage of `saturating_sub` in `process_claim_pool_tokens`.

Description: In `process_claim_pool_tokens`, the amount that is transferred from `vault_token_account_info` to `destination_token_acc_info` is computed as below

```
Unset
      let amount =

            u64::from(deposit_receipt.lst_amount).saturating_sub(fee_amount);
```

In a scenario where the Clock exhibits unexpected behaviour, the fee_amount may be incorrectly computed to be a number bigger than `deposit_receipt.lst_amount`. In such a scenario, `saturating_sub` will not produce an error, but instead drain the vault with a bigger than expected amount.

Recommendation: Use `checked_sub` to protect against the above scenario.

Customer's response: resolved in commit

Fix Review: The issue has been resolved.

I-03 `calculate_fee_amount` may return fee amount greater than the total_amount

Description: In `calculate_fee_amount`, if the `current_timestamp` is supplied such that `current_timestamp` < `self.deposit_time` due to an unexpected Clock behavior, then `cool_down_time_left` > `cool_down_seconds`. This results in `fee_amount` > `self.lst_amount`.

Recommendation: Add a check to make sure `current_timestamp` > `self.deposit_time`.

Customer's response: resolved in commit

Fix Review: The issue has been resolved.

Formal Verification

General Assumptions and Simplifications

- We do not model allocation of new accounts, so we simplified those functions to be essentially no-ops.
- We model PDA computation by assuming it returns a nondeterministic Pubkey.
- We use checked_sub instead of saturating_sub as explained in I-02. Doing so allows the prover to handle arithmetic overflows more efficiently.

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue
Violated	A counter-example exists that violates one of the assertions of the rule.

Formal Verification Properties

P-01. Fee is only paid by the redeemer.	
Status: Verified after fix	

Rule Name	Status	Description	Link to rule report
rule_only_redeemer_pays_fee	Verified after fix	*This rule asserts that if process_claim_pool_tokens executes without any errors, then the owner must be a signer. The rule fails due to L-01.*	*Report* *Report after fix*

P-2. Vault always has funds to cover its obligation.	
Status: Verified	

Rule Name	Status	Description	Link to rule report
rule_vault_increases_on_deposit	Verified	*The rule asserts that after process_deposit_stake is called, the vault_amount increases by an amount equal to the lst_amount of the deposit_receipt.*	*Report*
rule_vault_decreases_on_redeem	Verified	*The rule asserts that after process_claim_pool_tokens is called, the vault_amount decreases by an amount equal to the lst_amount of the deposit_receipt.*	*Report*

P-3. Fee and deposit parameters of DepositReceipt cannot be modified once created.

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_process_change_deposit_receipt_owner_does_not_change_fees	Verified	*This rule asserts that the function process_change_deposit_receipt_owner does not change the fields lst_amount, deposit_time, cool_down_seconds and initial_fee_bps of the deposit_receipt.*	*Report*

P-4. Tickets can always be redeemed.

Status: Verified	

Rule Name	Status	Description	Link to rule report
rule_process_claim_pool_tokens_does_not_revert	Verified	*This rule asserts that when process_claim_pool_tokens is called with correct AccountInfos and if there is no overflow, then the function will not revert.*	*Report*
rule_process_claim_pool_tokens_pubkey_check_owner	Verified	*This rule asserts that if process_claim_pool_tokens executed successfully, then the deposit_receipt belonged to the owner, and other AccountInfos are as expected.*	*Report*

P-5. The initial_fee_bps must always be less than 10_000

Status: Verified after fix	

Rule Name	Status	Description	Link to rule report
rule_process_init_stake_pool_deposit_stake_authority_fee_bounds_check	Verified	*This rule asserts that the initial_fee_bps set by process_init_stake_pool_deposit_stake_authority is less than 10_000.*	*Report*
rule_process_update_deposit_stake_authority_fee_bounds_check	Verified after fix	*This rule asserts that the initial_fee_bps set by process_update_deposit_stake_authority is less than 10_000. The rule fails due to M-01.*	*Report* *Report after fix*

P-6. Rules on fee assessment

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_no_fee_after_cooldown	Verified	*This rule asserts that the calculate_fee_amount method will return zero after the cool_down_period.*	*Report*

rule_fee_precision_loss	Verified	*This rule asserts the loss in precision for calculate_fee_amount. See M-02 for detail. The rule generates an example where calculate_fee_amount computes fee which is 1 less than the fee computed according to the recommendation in M-02.*	*Report*

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

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.