Jito Restaking

Security Assessment

October 25th, 2024 — Prepared by OtterSec

Nicola Vella nick0ve@osec.io

Robert Chen notdeghost@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Jito Labs engaged OtterSec to assess the `restaking` program. This assessment was conducted between September 20th and October 10th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 8 findings throughout this audit engagement.

In particular, we identified a critical vulnerability, where the burn instruction allows users to bypass withdrawal ticket checks, enabling unauthorized withdrawal of unstaked amounts (OS-JRS-ADV-00). Additionally, we highlighted another issue concerning the desynchronization of the VRT token amount stored in the Staker Withdrawal Ticket vault account by sending an additional VRT token directly to the token account (OS-JRS-ADV-02). Furthermore, the permissionless design of burning the withdrawal tickets allows callers to set an unfair minimum amount out (OS-JRS-ADV-03).

We also provided recommendations to pass the token program ID from the input `AccountInfo` (OS-JRS-SUG-00) and suggested specific modifications to enhance code quality and readability (OS-JRS-SUG-02).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/jito-foundation/restaking. This audit was performed against commit f04242f.

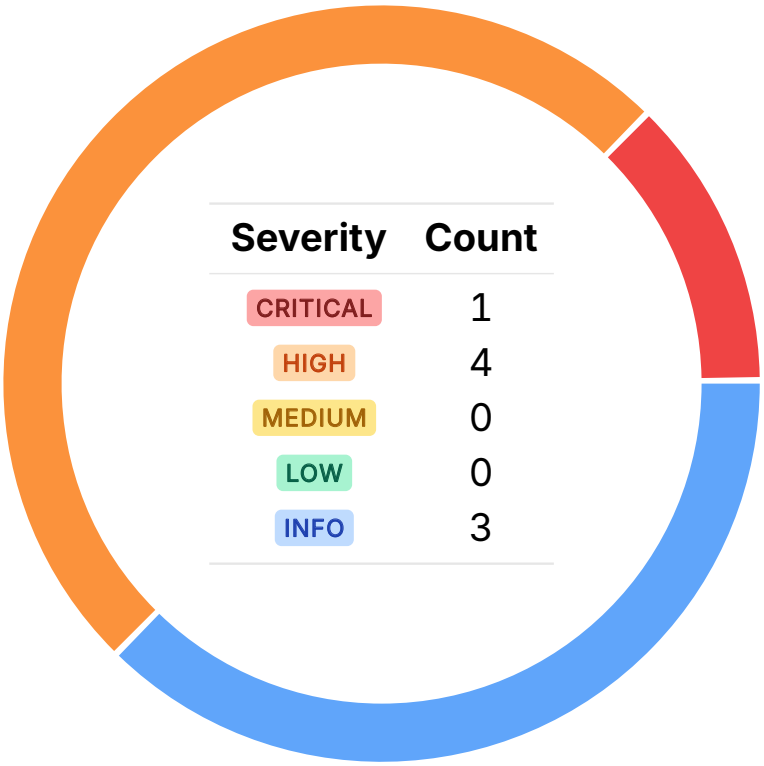
A brief description of the program is as follows:

Name	Description
restaking	A restaking platform for Solana and SVM environments.

03 — Findings

Overall, we reported 8 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	1
HIGH	4
MEDIUM	0
LOW	0
INFO	3

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-JRS-ADV-00	CRITICAL	RESOLVED	The `burn` instruction allows users to bypass withdrawal ticket checks, enabling unauthorized withdrawal of unstaked amounts.
OS-JRS-ADV-01	HIGH	RESOLVED	When a vault undergoes multiple slashes, it may reduce the deposited token balance but leave `VRT` token supply unchanged. Subsequent depositors will face share dilution, potentially contributing tokens without receiving adequate `VRT` shares.
OS-JRS-ADV-02	HIGH	RESOLVED	In `process_burn_withdrawal_ticket`, an attacker may desynchronize the `VRT` token amount stored in the `VaultStakerWithdrawalTicket` account by sending an additional `VRT` token directly to the token account.
OS-JRS-ADV-03	HIGH	RESOLVED	The permissionless design of `process_burn_withdrawal_ticket` allows callers to set an unfair `min_amount_out` and potentially skip the necessary `CrankVaultUpdateStateTracker` call, risking the `staker`'s rewards.
OS-JRS-ADV-04	HIGH	RESOLVED	There is a potential for share inflation in the vault, where sending tokens and invoking `UpdateVaultBalance` may allow an attacker to unfairly benefit from the first deposit, effectively stealing it.

Unauthorized Withdrawal of Unstaked Amount CRITICAL OS-JRS-ADV-00

Description

The `burn` instruction may be utilized to withdraw the unstaked amounts intended for withdrawal tickets. In the current implementation, since the function only validates that the `amount_in` is less than the `VRT` supply, any user with `VRT` tokens may call the `burn` instruction to initiate a withdrawal, regardless of whether they have a legitimate withdrawal ticket. This allows a user to bypass the standard withdrawal process by directly burning tokens and accessing funds

Remediation

Remove the burn instruction from the withdrawal process.

Patch

Resolved in PR#137.

Slashing-Induced Share Dilution HIGH OS-JRS-ADV-01

Description

The vulnerability arises when the vault's underlying tokens have been completely slashed, resulting in a balance of zero deposited tokens but still having outstanding `VRT` tokens in circulation. In such a scenario, the current implementation of `calculate_vrt_mint_amount` may result in an unfair outcome for new depositors. If `tokens_deposited` is zero due to slashing, but there are still outstanding `VRT` tokens, any new depositor will encounter the initial check: `if self.tokens_deposited() == 0`.

>_ *restaking/vault_core/src/vault.rs* RUST

```rust
/// Calculate the amount of VRT tokens to mint based on the amount of tokens deposited in
    ↪ thevault.
/// If no tokens have been deposited, the amount is equal to the amount passed in.
/// Otherwise, the amount is calculated as the pro-rata share of the total VRT supply.
pub fn calculate_vrt_mint_amount(&self, amount: u64) -> Result<u64, VaultError> {
    if self.tokens_deposited() == 0 {
        return Ok(amount);
    }
    amount
        .checked_mul(self.vrt_supply())
        .and_then(|x| x.checked_div(self.tokens_deposited()))
        .ok_or(VaultError::VaultOverflow)
}
```

This check then returns the amount deposited as the minted `VRT` without considering the existing `VRT` supply. In effect, this first depositor is assigned an amount of `VRT` equal to their deposited tokens, but the minted `VRT` does not accurately reflect their share of ownership because of the outstanding `VRT` tokens that others still hold.

Thus, the first depositor effectively donates their deposited tokens to prior `VRT` holders without receiving a fair share of `VRT`. Instead of gaining proportional ownership, their assets unfairly inflate the value of pre-existing `VRT` tokens. If the vault undergoes multiple slashes, it will progressively reduce `tokens_deposited` while outstanding `VRT` tokens remain the same, deflating the value of the `VRT` token. Consequently, there may be a risk of overflow in the share calculations due to needing to mint too many shares.

Remediation

Remove the slashing instruction.

Patch

Resolved in PR#141.

DOS Due to Withdrawal Ticket Desynchronization HIGH OS-JRS-ADV-02

Description

There is a potential Denial of Service (DoS) attack in `vault_program::process_burn_withdrawal_ticket` that may occur if an attacker manipulates the state of the system by directly sending `VRT` tokens to the `vault_staker_withdrawal_ticket_token_account`. This will result in inconsistencies between the amount of `VRT` tokens recorded in the token account and the amount recorded in the `VaultStakerWithdrawalTicket` account.

The direct transfer will increase the balance of tokens in the token account without updating `VaultStakerWithdrawalTicket`, creating a desynchronization between these two values since `vault_staker_withdrawal_ticket.vrt_amount()` would still reflect the original amount of tokens expected by the withdrawal process.

>_ *vault_program/src/burn_withdrawal_ticket.rs* RUST

```rust
pub fn process_burn_withdrawal_ticket(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    min_amount_out: u64,
) -> ProgramResult {
    [...]
    // close token account
    invoke_signed(
        &close_account(
            &spl_token::id(),
            vault_staker_withdrawal_ticket_token_account.key,
            staker.key,
            vault_staker_withdrawal_ticket_info.key,
            &[],
        )?,
        &[
            vault_staker_withdrawal_ticket_token_account.clone(),
            staker.clone(),
            vault_staker_withdrawal_ticket_info.clone(),
        ],
        &[&seed_slices],
    )?;
    close_program_account(program_id, vault_staker_withdrawal_ticket_info, staker)?;
    [...]
}
```

`close_account` checks that the amount stored in the token account is zero before allowing the account to be closed. If the amounts are out of sync, `close_account` operation may fail, as it verifies that the account's balance is zero. This effectively prevents the `staker` from closing their withdrawal ticket and claiming tokens, resulting in a denial-of-service scenario.

Remediation

Ensure the program always verifies the actual balance of the token account directly via `SPL` Token Program methods before allowing operations that rely on the amount of tokens.

Patch

Resolved in PR#140.

Lack of Proper Permission Checks and Safeguards HIGH OS-JRS-ADV-03

Description

The vulnerability in `vault_program::process_burn_withdrawal_ticket` arises from the lack of permission checks and safeguards around the parameters passed into the function, specifically the `min_amount_out` value and the potential omission of the `CrankVaultUpdateStateTracker` instruction. The `min_amount_out` parameter specifies the minimum amount of tokens the caller expects to receive after burning their withdrawal ticket. A malicious user may pass an artificially low value for `min_amount_out`.

>_ *vault_program/src/burn_withdrawal_ticket.rs* RUST

```rust
pub fn process_burn_withdrawal_ticket(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    min_amount_out: u64,
) -> ProgramResult {
    let (required_accounts, optional_accounts) = accounts.split_at(11);
    let [config, vault_info, vault_token_account, vrt_mint, staker, staker_token_account,
        ↪  vault_staker_withdrawal_ticket_info, vault_staker_withdrawal_ticket_token_account,
        ↪  vault_fee_token_account, token_program, system_program] =
        required_accounts
    else {
        return Err(ProgramError::NotEnoughAccountKeys);
    };

    Config::load(program_id, config, false)?;
    let config_data = config.data.borrow();
    [...]
}
```

If the fee deducted from the burn process is high, the `staker` may end up receiving much less than they anticipated. The `staker` may be unfairly penalized if the function proceeds to burn the withdrawal ticket with this unfairly low expectation. Also, if the `CrankVaultUpdateStateTracker` instruction is not invoked, the `staker` may miss out on rewards that they are entitled to because the state of the vault was not updated.

Remediation

Introduce checks to validate the `min_amount_out` parameter, ensuring it falls within a reasonable range based on the current state of the vault and the user's holdings. Also, enforce the requirement to call the `CrankVaultUpdateStateTracker` instruction within the same transaction.

Patch

Resolved in PR#142 and PR#138.

Vault Share Inflation Risk HIGH OS-JRS-ADV-04

Description

The vulnerability concerns the vault's mechanism, particularly the updating of balance and share minting. By sending tokens to the vault and invoking the `UpdateVaultBalance`, a user may manipulate the effective share value associated with `VRT` tokens. If this action occurs after a vault has been slashed (when the total tokens deposited are significantly reduced), the ratio of `VRT` tokens to deposited tokens becomes skewed. This allows the user to receive disproportionately more VRT tokens relative to their deposit, inflating their share of the vault without a corresponding increase in the overall asset value.

The primary risk involves the first depositor in a vault. If the first depositor's share value is not adequately protected against slashing, they can effectively lose the value of their initial deposit if later depositors take advantage of this inflation mechanism.

Remediation

Enforce a minimum deposit amount to ensure that only significant deposits are made, thereby reducing the incentive for malicious actors to exploit the system through minimal deposits. The rounding amount may be refunded in `mint_with_fee`.

Patch

Resolved in PR#150.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-JRS-SUG-00	Utilization of an unsafe pattern by directly passing `&spl_token::id()` as the `token_program_id` when interacting with the `SPL` token program.
OS-JRS-SUG-01	There is a possibility of over-withdrawing if a crank process is not complete within an epoch, causing discrepancies in the accounting, and delays in cranks result in improper updates since the cooldown is applied before the epoch's withdrawal accounting is finalized.
OS-JRS-SUG-02	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.

Unsafe SPL Token ID Handling OS-JRS-SUG-00

Description

In the current code, when calling `SPL` token instructions, the program passes `&spl_token::id()` directly as the token program ID. This utilizes the static program ID of the `SPL` token program (`spl_token::id()`), which assumes that this ID will always be the correct one in utilization. However, in some contexts, there may be custom deployments of the `SPL` token program or alternative token programs that implement similar functionality.

Remediation

Pass the token program ID from the input `AccountInfo`. This ensures that the token program ID is checked dynamically from the input accounts rather than relying on a hardcoded ID.

Risk of Over-withdrawing OS-JRS-SUG-01

Description

The current `VaultUpdateStateTracker` flow results in over-withdrawing if the crank does not finish in an epoch. However, if the crank is delayed, the actual state of the vault may change, and as a result, insufficient assets may be available to fulfill withdrawal requests. When a crank executes but does not finish in the expected time frame, the withdrawal requests that are processed may not account for the most recent state of the vault, resulting in improper accounting in `vault_operator_delegation.update`, since the update happens after cooldown is called for the current epoch.

Remediation

Ensure that the state of the vault is immediately reconciled after cooldown operations are processed to accurately reflect the current available assets.

Patch

Resolved in PR#145 and PR#163.

Code Maturity OS-JRS-SUG-02

Description

1. The `process_withdrawal_asset` and `process_initialize_vault_with_mint` instructions are unutilized and should be removed.
2. There is a potential inconsistency in the value of discriminators utilized in the `vault_core` accounts, where there is a jump between values, specifically from seven in `VaultStakerWithdrawalTicket` to nine in `VaultUpdateStateTracker`.
3. `WithdrawalAllocationMethod::Greedy` is biased against the first few operators. The withdrawal selection process always starts from the same point (zero index) in `check_and_update_index`, which creates a pattern of bias where only a subset of operators benefit from the withdrawal mechanism. Utilize a dynamic starting point based on the current epoch and the number of operators (`epoch % operators.length`). This will ensure every operator has an equal opportunity to be selected for withdrawals over time, regardless of their position in the list.

>_ vault_core/src/vault_update_state_tracker.rs RUST

```rust
pub fn check_and_update_index(&mut self, index: u64) -> Result<(), VaultError> {
    if self.last_updated_index() == u64::MAX {
        if index != 0 {
            msg!("VaultUpdateStateTracker incorrect index");
            return Err(VaultError::VaultUpdateIncorrectIndex);
        }
    }[...]
    self.last_updated_index = PodU64::from(index);
    Ok(())
}
```

Remediation

Implement the above-mentioned suggestions.

Patch

1. Issue #2 resolved in PR#139.
2. Issue #3 resolved in PR#144.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.