Bitpanda

Audit of the StakedVision and VisionTokenMigrator contracts

May 12, 2025

Common Prefix

Overview

Introduction

Common Prefix was commissioned to perform a security audit of Bitpanda's `StakedVision.sol` and `VisionTokenMigrator.sol` smart contracts.

The code is of high quality, with clear structure and well-commented functions that explain the expected behavior. It is also accompanied by an extensive test suite.

We recommend adding further test coverage for edge cases, particularly scenarios involving users who maintain stakes across different reward cycles.

Description of the contracts

The `VisionTokenMigrator` contract facilitates the migration from the original single-chain Pantos and BEST tokens to the new multi-chain Vision token on Ethereum. The migration process is initiated by an address with the `CRITICAL_OPS` role, which sends an amount of Vision tokens to the contract to fund the migration.

Once initialized, users can invoke the `migrate` function to exchange their entire balances of Pantos and BEST tokens for Vision tokens. Prior to doing so, users must grant the `VisionTokenMigrator` contract the necessary approvals to transfer their Pantos and BEST tokens.

The exchange rates from Pantos and BEST to Vision tokens are fixed and hardcoded within the contract.

The `StakedVision` contract enables users to stake Vision tokens and earn rewards in the form of additional Vision tokens. These rewards are distributed over cycles. Each cycle is initiated by an address with the `CRITICAL_OPS` role, which is responsible for setting the total reward amount for the cycle, transferring those rewards to the contract, and optionally defining a maximum yield per second. Rewards are distributed linearly over the duration of each cycle.

When a user stakes tokens, they receive ERC-4626-compliant share tokens proportional to the amount staked. These shares represent the user's stake and accrue rewards over time.

In addition to starting new cycles, the `CRITICAL_OPS` address can update the per-second reward cap and modify the cooldown duration at any time. When a non-zero `cooldownDuration` is set, users cannot immediately redeem their staked tokens and rewards. Instead, they must first call `cooldownShares` or `cooldownAssets`, specifying the amount they wish to withdraw. This request is recorded, and only after the cooldown period has elapsed can the user finalize the withdrawal. During the cooldown, no additional rewards are earned on the specified assets.

The `CRITICAL_OPS` address is expected to act in good faith and not arbitrarily or indefinitely extend the cooldown duration after users have deposited. However, even if the cooldown duration is increased, it will not affect users who have already initiated a withdrawal, as the cooldown timestamp is locked at the time of their request.

Disclaimer

Note that this audit does not give any warranties on the bug-free status of the given smart contracts, i.e. the evaluation result does not guarantee the nonexistence of any further findings of security issues. This audit report is intended to be used for discussion purposes only. Functional correctness should not rely on human inspection but be verified through thorough testing. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the project.

Findings Severity Breakdown

The findings are classified under the following severity categories according to the impact and the likelihood of an attack.

Level	**Description**
Critical	Logical errors or implementation bugs that are easily exploited and may lead to any kind of loss of funds
High	Logical errors or implementation bugs that are likely to be exploited and may have disadvantageous economic impact or contract failure
Medium	Issues that may break the intended contract logic or lead to DoS attacks
Low	Issues harder to exploit (exploitable with low probability), issues that lead to poor contract performance, clumsy logic or seriously error-prone implementation
Informational	Advisory comments and recommendations that could help make the codebase clearer, more readable and easier to maintain

Findings

Critical

No critical issues found.

High

HIGH-1	Rewards can be collected without prior staking due to zero supply edge case
Contract(s)	`StakedVision.sol`
Status	**Rejected**

Description

The `StakedVision` contract mints ERC4626-compliant shares to users based on the ratio of deposited tokens to the total assets and total supply of shares. Under normal conditions, the number of shares minted equals:

```
    shares = (assets * totalSupply) / totalAssets
```

However, when `totalSupply` is zero—such as at the beginning of a cycle or after all shares have been redeemed—the calculation deviates. In this case, the deposited token amount is directly converted to an equal number of shares, regardless of any accumulated rewards.

This behavior becomes problematic when combined with the reward distribution mechanism. Specifically, if rewards have accumulated before any user has deposited—meaning `totalAssets` has increased independently—a user can exploit this by:

1. Waiting until some rewards dx accumulate (e.g., after time `t` with `bpsYieldCapPerSecond == 0` and no existing staked tokens).

2. Making a minimal deposit x, receiving x shares since `totalSupply == 0`.

3. Immediately redeeming those x shares and receiving both the original x tokens and the full accumulated rewards dx.

Recommendation

We suggest introducing a safeguard in the reward distribution logic to ensure rewards are only distributed if `totalSupply > 0`. This guarantees that rewards are only accumulated when users have actually staked tokens.

Alleviation

The team has decided not to address this issue, as the described behavior is considered intentional. Specifically, when no stakes have occurred during a given cycle, the first staker is entitled to receive all accumulated rewards up to that point, regardless of the duration of his stake.

Moreover, the cycle design will be carefully constructed to make such scenarios rare, with only minimal rewards likely to accumulate before the first stake.

The team's official response was:

"No action has been taken for this issue as we think this is the expected behaviour.

In the provided example, rewards are no longer proportional to the staked period. However, this is considered fair and is the expected behaviour.

Furthermore, internal processes supporting how cycles are triggered make this situation pretty unlikely."

Medium

MEDIUM-1	Missing external pause controls in `StakedVision` contract
Contract(s)	`StakedVision.sol`
Status	**Resolved**

Description

The `StakedVision` contract inherits from [OpenZeppelin's `Pausable`](#) contract and defines a `PAUSER` role intended to control the pausing and unpausing of the contract. While the role is set, the contract does not expose any external `pause` or `unpause` functions. Additionally, the inherited `Pausable` contract only provides internal `_pause` and `_unpause` methods.

As a result, despite having a role dedicated to pausing, there is currently no way for the `PAUSER` to invoke these actions externally.

Recommendation

We recommend renaming the `StakedVision` contract's `_pause()` and `_unpause()` functions to `pause()` and `unpause()`, and declaring them as `external`. Additionally, the `whenNotPaused` and `whenPaused` modifiers on these functions can be removed, as the internal `Pausable` methods they invoke already enforce these checks.

Alleviation

The team addressed the issue at commit `c4e61ea46f9f0deeff5423732784e67b079a4bc6` by implementing externally callable `pause` and `unpause` functions, accessible only to designated roles.

Low

LOW-1	Subsequent requests override existing cooldown end
Contract(s)	`StakedVision.sol`
Status	**Rejected**

Description

When a non-zero cooldown duration is set, users must initiate a withdrawal by calling either `cooldownShares()` or `cooldownAssets()`. Upon calling, the specified amount is locked, and a `cooldownEnd` timestamp is set, after which the user can finalize the withdrawal.

However, if the user calls either function again before the previously initiated cooldown period ends, the newly requested assets are correctly added to the locked amount, but the `cooldownEnd` timestamp is reset for all locked assets—including those already partway through their cooldown. This behavior effectively delays the withdrawal of previously cooled-down assets, penalizing users for making additional cooldown requests during the waiting period.

Recommendation

We suggest either tracking separate cooldown periods for each amount of locked assets, allowing users to claim funds as their respective cooldowns expire or, if the current design is intentional, explicitly document this behavior in the contract comments and documentation to avoid confusion.

Alleviation

The team has informed us that this is by design and is considered expected behavior.

Informational/Suggestions

INFO-1	Redundant `CooldownDurationUpdated` event emission
Contract(s)	`IStakedVision.sol`
Status	**Resolved**

Description

The address with the `CRITICAL_OPS` role can call `updateCooldownDuration()` to modify the cooldown duration. Currently, each time this function is invoked, a `CooldownDurationUpdated` event is emitted—regardless of whether the new duration is actually different from the previous one.

Although the event includes both the old and new values (allowing observers to detect whether a change has occurred), emitting the event unnecessarily when no actual change has taken place introduces avoidable overhead and may create noise in event logs.

Recommendation

We suggest adding a conditional check to ensure that the new cooldown duration is different from the current one before proceeding with the update and emitting the event.

Alleviation

The issue was resolved at commit `74bc931d4f92ef5716ae4724be2b9860f064958b` by adding an additional check to ensure that a message is emitted only when the cooldown duration is updated to a new value.

INFO-2	Improvements on custom errors and events
Contract(s)	`IStakedVision.sol`
Status	**Resolved**

Description

The custom error `SurplusNotWithdraw` is defined in the contract but never emitted. Based on the comments, it appears to be a remnant of an older implementation of `createRewardsCycle`, no longer relevant.

Additionally, the `RewardsCycleCreated` event currently emits only the end time and total reward amount for each cycle. However, it omits the `bpsYieldCapPerSecond`, which is important for users and off-chain systems to fully understand the parameters of a given reward cycle.

Furthermore, the `CooldownStarted` event does not include the cooldown end time, which is the point after which the user is allowed to withdraw their assets and rewards.

Recommendation

We suggest removing the unused `SurplusNotWithdraw` error to simplify the codebase, extending the `RewardCycleCreated` event to include the `bpsYieldCapPerSecond` parameter and updating the `CooldownStarted` event to include the cooldown end timestamp to improve transparency and off-chain usability.

Alleviation

The team implemented the suggested improvements at commit `7d95128f76676be43398439541b5b6aa6087a48e`.

INFO-3	Missing input validation
Contract(s)	`VisionTokenMigrator.sol`
Status	**Resolved**

Description

In the constructor of the `VisionTokenMigrator` contract, a sanity check is performed to ensure that the token addresses (Pantos, BEST, Vision) are not set to the zero address. However, similar checks are not applied to the `CRITICAL_OPS` and `DEFAULT_ADMIN` role addresses.

Failing to validate these role addresses could result in accidentally assigning critical roles to the zero address. This would make the contract functionally unusable and necessitate a redeployment to fix the error.

Recommendation

We suggest adding explicit checks in the constructor to ensure that both `CRITICAL_OPS` and `DEFAULT_ADMIN` addresses are not the zero address.

Alleviation

The issue was resolved at commit `c7afa6f3af547448a2de3055c5303dcc80b33079` by adding the suggested sanity checks.

INFO-4	Lack of mechanism to withdraw unused Vision tokens
Contract(s)	`VisionTokenMigrator.sol`
Status	**Rejected**

Description

The address assigned the `CRITICAL_OPS` role is responsible for initiating the migration process and transferring a large amount of Vision tokens to the `VisionTokenMigrator` contract. These tokens are then distributed to users in exchange for their Pantos and BEST tokens.

To ensure full coverage of the circulating supply, it is expected that a generous amount of Vision tokens will be deposited. However, if some users never migrate their tokens, any unused Vision tokens will remain indefinitely locked within the contract, with no current mechanism to recover them.

Recommendation

We suggest implementing a function allowing the `CRITICAL_OPS` address to withdraw unclaimed Vision tokens from the contract after the migration period has ended or under predefined conditions (e.g., after a certain time or after pausing the migration).

Alleviation

The team has deliberately chosen not to implement a withdraw function, in order to avoid taking ownership of the unclaimed Vision tokens.

INFO-5	Prevent zero-duration reward cycles
Contract(s)	`StakedVision.sol`
Status	**Resolved**

Description

In the `createRewardsCycle()` function, the following condition is used to validate the end time of a new rewards cycle:

```
if(rewardsCycleEndTimestamp < block.timestamp) revert CycleEndTimestampInThePast();
```

However, using a strictly less-than (<) comparison allows the `rewardsCycleEndTimestamp` to equal the current `block.timestamp`, which results in a cycle with zero duration. In such a case, no rewards would be distributed, potentially leading to confusion or wasted gas.

Recommendation

We suggest updating the condition to use a less-than-or-equal (<=) comparison to ensure that each new reward cycle has a strictly positive duration.

Alleviation

The team fixed the issue at commit `c7afa6f3af547448a2de3055c5303dcc80b33079` by tightening the check to ensure that the cycle's end time is strictly in the future.

INFO-6	Inaccurate or missing comments
Contract(s)	`StakedVision.sol`, `VisionTokenMigrator.sol`, `IStakedVision.sol`
Status	**Resolved**

Description

- `IStakedVision.sol`:
    - The comment associated with the `NotEnoughRewardFunds()` error incorrectly states that it is triggered when users attempt to withdraw rewards. In reality, this error is emitted when the `CRITICAL_OPS` address attempts to set a reward amount for a new cycle that exceeds the available funds in the contract.

- `VisionTokenMigartor.sol`:

- In `_previewPantosMigration()`:

```
 // and equal, we can simplifying the formula to:

 // (amount * PAN_TO_VISION_EXCHANGE_RATE * DECIMALS_DIFFERENCE_SCALING)

 // EXCHANGE_RATE_SCALING_FACTOR //Common prefix: the simplified formula is
                                 // (amount*PAN_TO_VISION_EXCHANGE_RATE)
```

- In `_previewBestMigration()`:

```
// @param amount The amount of Pantos tokens //Common prefix: BEST tokens
-----------------------------------------------

// and equal, we can simplifying the formula to:

 // (amount * BEST_TO_VISION_EXCHANGE_RATE * DECIMALS_DIFFERENCE_SCALING)

 // EXCHANGE_RATE_SCALING_FACTOR //Common prefix: the simplified formula is
                                 // (amount*PAN_TO_VISION_EXCHANGE_RATE)
```

- `StakedVision.sol`:
  The core ERC-4626 functions (`deposit`, `mint`, `withdraw`, `redeem`) reference the ERC-4626 base contract for documentation, but the inherited contract does not contain any comments or explanations.

Alleviation

The team made the suggested changes at commit `3d209c0f9a4a3f5ec7778ab077c1ee4bab3f357a`.

About Common Prefix

Common Prefix is a blockchain research, development, and consulting company consisting of a small number of scientists and engineers specializing in many aspects of blockchain science. We work with industry partners who are looking to advance the state-of-the-art in our field to help them analyze and design simple but rigorous protocols from first principles, with provable security in mind.

Our consulting and audits pertain to theoretical cryptographic protocol analyses as well as the pragmatic auditing of implementations in both core consensus technologies and application layer smart contracts.