BOB Staking Security Review

Pashov Audit Group

Conducted by: Ch_301, Hals, Oblivionis

March 4th 2025 - March 7th 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **bob-collective/bob-staking** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About BOB Staking

BOB Staking is a staking system that implements an unbonding mechanism and instant withdrawal features. Users can stake tokens, earn rewards, and choose between a standard unbonding process or instant withdrawal with a penalty.

5. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - ab1b12f91edd7b147cbbccb8e71aa154087d2408

fixes review commit hash - f1d999826b4081f200a8c64059d6b49d19b4a9ca

Scope

The following smart contracts were in scope of the audit:

- `BonusWrapper`
- `SafeERC20Ext`
- `UnbondableStake`
- `Address`
- `CurrencyTransferLib`

7. Executive Summary

Over the course of the security review, Ch_301, Hals, Oblivionis engaged with BOB to review BOB Staking. In this period of time a total of **6** issues were uncovered.

Protocol Summary

Protocol Name	BOB Staking
Repository	https://github.com/bob-collective/bob-staking
Date	March 4th 2025 - March 7th 2025
Protocol Type	Token Staking

Findings Count

Severity	Amount
Medium	2
Low	4
Total Findings	**6**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Failure to claim remaining unclaimedRewards for unbonded stakes	Medium	Resolved
[M-02]	Incorrect check on the staker address in UnbondableStake.stake()	Medium	Resolved
[L-01]	DepositRewardTokens issue due to fee-on-transfer tokens	Low	Resolved
[L-02]	InstantWithdraw transfers stake to owner instead of specified receiver	Low	Resolved
[L-03]	Missing _disableInitializers() in the constructor	Low	Resolved
[L-04]	Precision loss in _calculateRewards	Low	Acknowledged

8. Findings

8.1. Medium Findings

[M-01] Failure to claim remaining `unclaimedRewards` for unbonded stakes

Severity

Impact: Medium

Likelihood: Medium

Description

In the `UnbondableStake.unbond()` function, when stakers want to withdraw their stake via `unbond()` or `instantWithdraw()`, the `_claimRewards(_stakeMsgSender(), false)` is called to collect any unclaimed rewards and add them to the staked amount of the user to be withdrawn. The `shouldRevert` flag indicates whether the call should revert if the rewards to be paid to the staker are zero, where this flage is set to `false` to indicate the reverting is not allowed in this case:

```
function unbond() external {
         //...
        _claimRewards(_stakeMsgSender(), false);

        uint256 amountStaked = stakers[_stakeMsgSender()].amountStaked;
        if (amountStaked == 0) revert ZeroTokenUnbond();

        stakers[_stakeMsgSender()].amountStaked = 0;
        //...
    }
```

The rewards to be paid to the staker can be zero if the contract's balance of reward tokens is zero (`rewardTokenBalance = 0`):

```
function _claimRewards(address receiver, bool shouldRevert) internal {
        //...
        if (stakers[receiver].amountStaked == 0) revert NoRewardsError(); <@2

        uint256 rewards = stakers[receiver].unclaimedRewards +
            _calculateRewards(receiver);

        uint256 rewardsToPay = rewards;

        if (rewards > rewardTokenBalance) {
            rewardsToPay = rewardTokenBalance;
        } // Leaving amount for future possible withdraw

        if (rewardsToPay == 0) { <@1
            if (shouldRevert) revert InsufficientContractRewardBalance();
            else return;
        }

        stakers[receiver].unclaimedRewards = rewards - rewardsToPay;
        stakers[receiver].amountStaked += rewardsToPay;

        //...
    }
```

If the contract reward token balance is zero or insufficient to cover the staker rewards, the rewards entitled to the staker will not be fully added to the staked amount that will be withdrawn. The `unbond()` and `instantWithdraw()` functions set `stakers[_stakeMsgSender()].amountStaked = 0`, and if the contract's reward token balance becomes sufficient again later, **the user who unbonded (or called `instantWithdraw()`) while there was insufficient reward token balance will not be able to claim their remaining unclaimed rewards via the `claimRewards()` function**, as the `_claimRewards()` function checks if `amountStaked > 0` (@1), so the user will not be able to claim these remaining rewards unless they stake again -> then create unbond -> then withdraw.

Recommendations

Consider implementing a mechanism to track the unclaimed rewards that haven't been moved to the `amountStaked` of the user, where these remaining unclaimed rewards should be claimable later regardless of the user's staked. Also, you can allow users to set the value of `bool shouldRevert` in `_claimRewards()` function.

[M-02] Incorrect check on the staker address in `UnbondableStake.stake()`

Severity

Impact: Medium

Likelihood: Medium

Description

In the `UnbondableStake.stake()` function, the contract includes a check to prevent users from staking again if they have already requested to unbond their tokens but not finalized their withdrawals, where the check is currently performed on the caller address (`_stakeMsgSender()`)

```
function stake(uint256 _amount, address receiver) external nonReentrant {
    //...
    if (unbondEndTimes[_stakeMsgSender()] != 0) {
        revert UnbondAlreadyStarted();
    }
    //...
    }
```

The issue with this implementation is that the check should be performed on the `receiver` address (the address where the tokens will be staked on behalf of) instead of the caller address. This incorrect check allows any user to bypass the `unbondEndTimes` check by staking from a different address or calling the `UnbondableStake.stake()` function indirectly via the `BonusWrapper.stake()` function, even if they have initiated an unbonding process.

Recommendations

Update the check to verify the `unbondEndTimes` on the receiver address instead of the caller address:

```
function stake(uint256 _amount, address receiver) external nonReentrant {
    //...
-   if (unbondEndTimes[_stakeMsgSender()] != 0) {
+   if (unbondEndTimes[receiver] != 0) {
        revert UnbondAlreadyStarted();
    }
    //...
    }
```

8.2. Low Findings

[L-01] `DepositRewardTokens` issue due to fee-on-transfer tokens

The `depositRewardTokens()` function, as currently implemented, likely assumes that the amount of tokens transferred into the contract matches the amount recorded for reward distribution. This assumption is vulnerable to fee-on-transfer (FoT) tokens. When an FoT token is deposited, the actual amount received by the contract will be less than the amount specified in the `safeTransferFrom()` function call due to the automatic deduction of fees during the transfer. this will affect both `rewardTokenBalance` and `stakingTokenBalance` values. The `BonusWrapper.sol#stake()` has the same issue.

To resolve this, use the same logic in `UnbondableStake.sol#stake()` function.

[L-02] `InstantWithdraw` transfers stake to owner instead of specified receiver

In the `UnbondableStake.instantWithdraw()` function, stakers are allowed to withdraw their stakes with a penalty, where the function takes a `_receiver` address, which is supposed to receive the withdrawn stake. However, the function incorrectly transfers the staked tokens to the caller (the stake owner) instead of the specified `_receiver` address:

```
function instantWithdraw(address _receiver) external nonReentrant {
        //...
        IERC20(stakingToken).safeTransfer(_stakeMsgSender(), _amountForUser);
        //...
    }
```

Recommendation: update the `instantWithdraw()` function to transfer the staked tokens to the receiver address instead of the caller’s address:

```
function instantWithdraw(address _receiver) external nonReentrant {
        //...
-       IERC20(stakingToken).safeTransfer(_stakeMsgSender(), _amountForUser);
+       IERC20(stakingToken).safeTransfer(_receiver, _amountForUser);
        //...
    }
```

[L-03] Missing `_disableInitializers()` in the constructor

In the `UnbondableStake` contract, the UUPSUpgradeable pattern is used, but the constructor does not call the `_disableInitializers()` function. As per best practices outlined by OpenZeppelin, the `initialize()` function in an upgradeable contract should be disabled within the constructor to prevent it from being called by anyone.

Recommendation: add `_disableInitializers()` call in the constructor of the upgradeable contract:

```
constructor() {
    _disableInitializers();
}
```

[L-04] Precision loss in `_calculateRewards`

There are two rounding operations in `UnbondableStake::_calculateRewards()`. If the protocol uses a lower-decimal token and a lower reward numerator, this rounding will cause users' rewards to be reduced. If users claim rewards too frequently, they will continuously fail to receive rewards, and their rewards during the period will be permanently lost.

```
uint256 rewardsProduct = (endTime - startTime) *
                (staker.amountStaked) *
                (condition.rewardRatioNumerator);

            uint256 rewardsSum = _rewards +
                ((rewardsProduct / condition.timeUnit) /
                    condition.rewardRatioDenominator);

            _rewards = rewardsSum;
```

Consider the following scenario: An `UnbondableStake` uses the value in test file:

- 1-day time unit -> timeUnit = 86400
- 1 rewardRatioNumerator
- 100 rewardRatioDenominator.
- WBTC staking token -> 8 decimals

Each claim may result in a loss of up to 86400/100 = 86 satoshi. If we consider the mathematical expectation (where each rounding loss is a value between 0 and 86400), each claim results in an average loss of 43 satoshis. Every 23 calls will result in a loss of 1 USD.

If using an 18-decimal token, no fix is necessary. If you want to use a lower decimal token, you can:

- Specify a higher rewardRatioNumerator and rewardRatioDenominator.
- Modify the protocol logic so that if current reward is rounded to 0, `stakers[_staker].timeOfLastUpdate` is not updated.