Bio Security Review

Pashov Audit Group

Conducted by: Ch_301, Oblivionis, zark, Opeyemi, Bretzel, Afriauditor

March 12th 2025 - March 14th 2025

Contents

1. About Pashov Audit Group 2
2. Disclaimer 2
3. Introduction 2
4. About Bio 2
5. Risk Classification 3
   5.1. Impact 3
   5.2. Likelihood 3
   5.3. Action required for severity levels 4
6. Security Assessment Summary 4
7. Executive Summary 5
8. Findings 7
   8.1. Critical Findings 7
      [C-01] Incorrect vestingDuration is passed 7
   8.2. High Findings 9
      [H-01] Whitelisted users can bypass maxTotalPledgePerAddress requirement 9
      [H-02] updateFundraiser() can be DoSed forever if attacker donates 1 wei 10
   8.3. Medium Findings 12
      [M-01] bioDaoTokenToAmount is not precise 12
      [M-02] TotalPledgedForWhitelist not decreased upon _unpledge causing incorrect whitelistAllocation 13
   8.4. Low Findings 15
      [L-01] Curation with whitelistAllocation == bioTarget cannot be settled 15
      [L-02] User transaction to pledge() function can arbitrarily revert 15
      [L-03] Users cannot top up pledges when remaining allowance is less than minPledgePerAddress 16

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 **bio-xyz/launchpad-evm** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Bio

BIO Curation contract enables BIO token holders to crowdfund projects through pledging mechanisms, featuring whitelist access, vesting schedules and automated fundraiser deployment. Users can deposit BIO tokens to fund projects, withdraw their pledged BIO tokens, lock up BIODAO tokens for vesting, and later claim the vested BIODAO tokens.

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 - e232dca8a7899324be31ac35090e7b88221b13a6

fixes review commit hash - f5c92a54ec87310d59b8e4e6af7cbec914d2c435

Scope

The following smart contracts were in scope of the audit:

- `Curation`
- `LaunchFactory`

7. Executive Summary

Over the course of the security review, Ch_301, Oblivionis, zark, Opeyemi, Bretzel, Afriauditor engaged with Bio to review Bio. In this period of time a total of **8** issues were uncovered.

Protocol Summary

Protocol Name	Bio
Repository	https://github.com/bio-xyz/launchpad-evm
Date	March 12th 2025 - March 14th 2025
Protocol Type	Fundraising factory

Findings Count

Severity	**Amount**
Critical	1
High	2
Medium	2
Low	3
Total Findings	**8**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Incorrect vestingDuration is passed	Critical	Resolved
[H-01]	Whitelisted users can bypass maxTotalPledgePerAddress requirement	High	Resolved
[H-02]	updateFundraiser() can be DoSed forever if attacker donates 1 wei	High	Resolved
[M-01]	bioDaoTokenToAmount is not precise	Medium	Resolved
[M-02]	TotalPledgedForWhitelist not decreased upon _unpledge causing incorrect whitelistAllocation	Medium	Resolved
[L-01]	Curation with whitelistAllocation == bioTarget cannot be settled	Low	Resolved
[L-02]	User transaction to pledge() function can arbitrarily revert	Low	Resolved
[L-03]	Users cannot top up pledges when remaining allowance is less than minPledgePerAddress	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Incorrect `vestingDuration` is passed

Severity

Impact: High

Likelihood: High

Description

During `Curation::claim()`, the amount of `bioDaoToken` is used to create a vesting schedule for the claimer. However, the `vestingDuration` being used and passed to the `TokenVesting::createVestingSchedule()` function is incorrect.

```
function createVestingSchedule(
        address _beneficiary,
        uint256 _start,
        uint256 _cliff,
        uint256 _duration,
        uint256 _slicePeriodSeconds,
        bool _revokable,
        uint256 _amount
    ) external whenNotPaused onlyRole(VESTING_CREATOR_ROLE) {
        _createVestingSchedule(
          _beneficiary,
          _start,
          _cliff,
          _duration,
          _slicePeriodSeconds,
          _revokable,
          _amount
        );
    }
```

As shown above, this function expects `_duration` in seconds to determine the vesting period. However, `vestingDuration` is being passed as a Unix timestamp instead.

```
function claim() external {
        // ...
        // If the vesting period has ended, transfer tokens directly to the user
        if (vestingDuration < block.timestamp) {
            // ...
        } else {
            bioDaoToken.safeTransfer(address(vesting), bioDaoTokenToAmount);
            // Create a vesting schedule for the sender
            vesting.createVestingSchedule(

                                msg.sender, vestingStart, vestingCliff, vestingDurati
            );
        }
        // ...
    }
```

This would be catastrophic for vesting schedule creation, as the duration would be an extremely large number.

Recommendations

`TokenVesting::createVestingSchedule()` expects a duration in seconds, not a Unix timestamp. Ensure that the duration of each vesting schedule is calculated correctly and passed accordingly.

8.2. High Findings

[H-01] Whitelisted users can bypass `maxTotalPledgePerAddress` requirement

Severity

Impact: Medium

Likelihood: High

Description

The `maxTotalPledgePerAddress` requirement is intended to limit how much an address can pledge in a `Curation`. However, due to the fact that during `_unpledge`, the `whitelistPledges` of the user is not decreased, a whitelisted user can bypass this requirement and pledge any amount they want.

First, let's examine how the pledge function works for whitelisted users:

```
function _whitelistPledge(uint256 _amount, bytes32[] calldata _proof) internal {
        bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode
          (msg.sender))));
        if (!MerkleProofLib.verify(_proof, merkleRoot, leaf))
              revert Curation__InvalidProof();

        if (_amount + totalPledgedForWhitelist > whitelistAllocation)
              revert Curation__WhitelistPledgedReached();
@>        if (_amount + pledges[msg.sender] > maxAmountPerWhitelistedAddress)
              revert Curation__OverMaxWhitelistPledge();

        BIO_TOKEN.safeTransferFrom(msg.sender, address(this), _amount);

        totalPledged += _amount;
        pledges[msg.sender] += _amount;
        totalPledgedForWhitelist += _amount;
        whitelistPledges[msg.sender] += _amount;

        emit Pledged(msg.sender, _amount);
    }
```

The highlighted check ensures that a whitelisted user cannot pledge more than `maxAmountPerWhitelistedAddress`. However, the issue arises in `_unpledge`, where the `whitelistPledges` value is not updated. This allows a whitelisted

user to repeatedly pledge and `_unpledge`, leading to an inflated `whitelistPledges` balance.

Why is this a problem? The inflated `whitelistPledges` value is later used during the non-whitelisted phase in the following check:

```
function pledge(uint256 _amount, bytes32[] calldata _proof) external {
        // ...

        // Ensure the sender has not exceeded their max pledge limit

                uint256 whitelistPlegde = whitelistPledges[msg.sender] > 0 ? whitelis
@>          if (
            pledges[msg.sender] + _amount
            > maxTotalPledgePerAddress + whitelistPlegde) {
              revert Curation__OverMaxPledge();
        }

        // ...
        }
    }
```

`whitelistPlegde` can be a big number and in this way the user will be able to pledge up to `maxTotalPledgePerAddress + whitelistPlegde`. To understand it better, lets think of this scenario :

- `maxAmountPerWhitelistedAddress` : 10
- `maxTotalPledgePerAddress` : 3
- Whitelisted user pledge `10` during whitelist phase. Unpledges `10`. Pledges `10`. Unpledges `10`. At this point, `whitelistPledges` is 20, but the user has no actual pledged amount.
- After the whitelist phase ends, the user can now pledge `3 + 20 = 23`, exceeding both `maxTotalPledgePerAddress` and `maxAmountPerWhitelistedAddress`.

Recommendations

Consider decreasing the `whitelistPledges` value when a whitelisted user unpledges. However, fixing this issue may be nontrivial, so an alternative solution could be to disallow unpledging for whitelisted users.

[H-02] `updateFundraiser()` can be DoSed forever if attacker donates 1 wei

Severity

Impact: High

Likelihood: Medium

Description

Fundraiser is supposed to transfer the required `bioDaoTokenToDistribute` before calling `updateFundraiser` and make the fundraise successful. However, an attacker can permanently prevent the execution of `Curation::updateFundraiser()` by frontrunning the call with a minimal 1 wei donation of `bioDaoToken`. In this way, the following check, will always revert since the actual balance would be bigger than the `bioDaoTokenToDistribute`.

```
if (bioDaoTokenToDistribute != bioDaoToken.balanceOf(address(this))) {
    revert Curation__BioDAOTokenForVestingNotSent();
}
```

Recommendations

Modify the check to allow the function to proceed even if the contract holds an excess balance.

```
function updateFundraiser() external onlyFundraiserContract {
        if (isFundraiseSuccessful) revert Curation__FundraiserAlreadyUpdated();
-        if (bioDaoTokenToDistribute != bioDaoToken.balanceOf(address(this))) {
-            revert Curation__BioDAOTokenForVestingNotSent();
-        }
+        if (bioDaoTokenToDistribute > bioDaoToken.balanceOf(address(this))) {
+            revert Curation__BioDAOTokenForVestingNotSent();
+        }

        // ...
    }
```

8.3. Medium Findings

[M-01] `bioDaoTokenToAmount` is not precise

Severity

Impact: Medium

Likelihood: Medium

Description

`Curation::claim()` calculates the tokens to be paid to users as:

`bioDaoTokenToAmount = amount * bioDaoTokenToDistribute / bioTarget.`

This is problematic because users can still `unPledge` after the fundraise succeeds, causing `totalPledged` to decrease. At this point, the contract no longer holds the full `bioTarget` amount of $BIO, meaning users receive a smaller proportion of `bioDaoToken`, and the remaining `bioDaoToken` will be locked in the contract.

```
// Curation.sol::L192
    function claim() external {
        if (!isFundraiseSuccessful) revert Curation__FundraiserNotSuccessful();

        uint256 amount = pledges[msg.sender];
        if (amount == 0) revert Curation__ZeroValue();

        pledges[msg.sender] = 0;

                uint256 bioDaoTokenToAmount = amount * bioDaoTokenToDistribute / bioT

        ...
    }
```

Recommendations

```
---
-         uint256 bioDaoTokenToAmount = amount * bioDaoTokenToDistribute / bioTarget;
+++
+         uint256 bioDaoTokenToAmount = amount * bioDaoTokenToDistribute / totalPledge
```

[M-02] `TotalPledgedForWhitelist` not decreased upon `_unpledge` causing incorrect `whitelistAllocation`

Severity

Impact: Medium

Likelihood: Medium

Description

During the whitelisted phase, users can pledge into a `Curation`, and `totalPledgedForWhitelist` will be increased by the same amount.

```
function _whitelistPledge
      (uint256 _amount, bytes32[] calldata _proof) internal {
        bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode
          (msg.sender))));
        if (!MerkleProofLib.verify
          (_proof, merkleRoot, leaf)) revert Curation__InvalidProof();

        if
          (_amount + totalPledgedForWhitelist > whitelistAllocation) revert Curation__
        if
          (_amount + pledges[msg.sender] > maxAmountPerWhitelistedAddress) revert Cura

        BIO_TOKEN.safeTransferFrom(msg.sender, address(this), _amount);

        totalPledged += _amount;
        pledges[msg.sender] += _amount;
        totalPledgedForWhitelist += _amount;
        whitelistPledges[msg.sender] += _amount;

        emit Pledged(msg.sender, _amount);
    }
```

As we can see, `whitelistAllocation` is intended to limit the total pledges made during the whitelist phase. However, during `_unpledge`, `totalPledgedForWhitelist` is not decreased. This creates a situation where the contract believes that `totalPledgedForWhitelist` reflects the actual pledged amount from whitelisted users, but in reality, the actual amount is lower. As a result, the `whitelistAllocation` limit will appear to have been reached, even though the true pledged amount does not match it.

Recommendations

Fixing this issue may not be trivial, but an alternative solution could be to disallow unpledging for whitelisted users.

8.4. Low Findings

[L-01] `Curation` with `whitelistAllocation == bioTarget` cannot be settled

There is a valid edge case where, if a curation allocates all `bioTarget` to whitelist users, it will never be able to be locked or trigger success actions. This is because `Curation::pledge` will return early if whitelist is enabled and the whitelist period is still active, so if the `bioTarget` is filled during whitelist period, the Curation can never get locked.

Consider doing such modification:

```
// Curation::L276
    function updateWhitelistAllocation
      (uint256 _whitelistAllocation) external onlyApproved {
---         if
- (_whitelistAllocation > bioTarget) revert Curation__AllocationAboveBioTarget();
+++         if
+ (_whitelistAllocation >= bioTarget) revert Curation__AllocationAboveBioTarget();
        if
          (_whitelistAllocation < totalPledgedForWhitelist) revert Curation__Allocatio
        emit WhitelistAllocationUpdated
          (whitelistAllocation, _whitelistAllocation);
        whitelistAllocation = _whitelistAllocation;
    }
```

[L-02] User transaction to `pledge()` function can arbitrarily revert

In the `pledge()` function When the last user tried to pledge and this check `totalPledged + _amount > bioTarget` is true the `_amount` value will get reduced The new `_amount` is the one that should be used in the check that ensures the sender has not exceeded their max pledge limit.

The logic should be like this:

```
File: Curation.sol

        // If the total pledges exceed the target, only allow the remaining
        // amount to be pledged
        if (totalPledged + _amount > bioTarget) {
            _amount = bioTarget - totalPledged;
        }


                uint256 whitelistPlegde = whitelistPledges[msg.sender] > 0 ? whitelis
        if
          (pledges[msg.sender] + _amount > maxTotalPledgePerAddress + whitelistPlegde)
            revert Curation__OverMaxPledge();
        }
```

[L-03] Users cannot top up pledges when remaining allowance is less than `minPledgePerAddress`

`Curation.sol` implements pledge limits with `minPledgePerAddress` and `maxTotalPledgePerAddress`. However, this creates a situation where a user who has already pledged close to their `maxTotalPledgePerAddress` limit cannot top up to reach their maximum allowance if the difference is less than `minPledgePerAddress` :

Example Scenario:

- `maxTotalPledgePerAddress`: 0.5 ether
- `minPledgePerAddress`: 0.05 ether
- User already pledged: 0.46 ether

If this user attempts to pledge the remaining 0.04 ether to reach their maximum capacity, the transaction will revert with `Curation__UnderMinPledge()` because 0.04 ether < 0.05 ether (the minimum pledge amount).

If the user attempts to pledge more than 0.04 ether (to meet the minimum threshold), the transaction will revert with `Curation__OverMaxPledge()` because their total pledge would exceed their maximum allowance.

This edge case creates a frustrating "dead zone" for users where they cannot utilize their full pledge allocation if the remaining amount falls below the minimum pledge requirement.