Bio Security Review

Pashov Audit Group

Conducted by: T1MOH, pontifex, peanuts

June 13st 2024 - June 15th 2024

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

4. About Bio

The FairAuctionVesting smart contract facilitates a token sale where users can swap DAO tokens for BIO and VBIO tokens, with the latter subject to a vesting schedule. It includes mechanisms for handling contributions, claiming tokens, vesting schedules, and emergency fund withdrawals.

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 - c28a41517d1f426d49e9b477e565432f1307ba58

fixes review commit hash - 70e2f84927752fd43446dee282df189993f55548

Scope

The following smart contracts were in scope of the audit:

- `FairAuctionVesting`

7. Executive Summary

Over the course of the security review, T1MOH, pontifex, peanuts engaged with Bio to review Bio. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	Bio
Repository	https://github.com/MTXstudio/bio-contracts
Date	June 13st 2024 - June 15th 2024
Protocol Type	Token sale vesting

Findings Count

Severity	Amount
Medium	1
Low	6
Total Findings	**7**

Summary of Findings

ID	Title	Severity	Status
[M-01]	User can lose the right to claim tokens when calling claim	Medium	Resolved
[L-01]	Consider transferring unsold Bio to Treasury	Low	Resolved
[L-02]	Reasonable values for START_TIME and END_TIME	Low	Resolved
[L-03]	Lack of DAO token address check	Low	Resolved
[L-04]	Centralization risk whereby owner can withdraw all BIO tokens	Low	Acknowledged
[L-05]	swap and claim functions will be unavailable	Low	Acknowledged
[L-06]	Unexpected vesting schedule when claiming	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] User can lose the right to claim tokens when calling `claim`

Severity

Impact: High

Likelihood: Low

Description

In case of emergency the `DEFAULT_ADMIN_ROLE` can withdraw `BIO_TOKEN` balance from the contract. At the same time there are no restrictions for users to try `claim` tokens. Since the `_safeClaimTransfer` can transfer the actual balance instead of the amount users will receive 0 `BIO_TOKEN` and lose the right to claim tokens again. Moreover the `VBIO_TOKEN` contract also will receive 0 but will accept a vesting schedule with the correct amount. So there will not be enough tokens in the `VBIO_TOKEN` contract balance for all vestings.

```
function claim() external nonReentrant isClaimable {
        UserInfo storage user = userInfo[msg.sender];

        if (totalRaised == 0 || user.contribution == 0) revert ZeroContribution
          ();
>>      if (user.hasClaimed) revert AlreadyClaimed();
>>      user.hasClaimed = true;

        (uint256 bioAmount, uint256 vbioAmount) = getExpectedClaimAmount
          (msg.sender);

        emit Claim(msg.sender, bioAmount, vbioAmount);

>>      if (bioAmount > 0) {
            // send BIO_TOKEN contribution
>>          _safeClaimTransfer(BIO_TOKEN, msg.sender, bioAmount);
        }
>>      if (vbioAmount > 0) {
            // send VBIO_TOKEN contribution
>>          _safeClaimTransfer(BIO_TOKEN, address(VBIO_TOKEN), vbioAmount);
            // create vesting schedule
            VBIO_TOKEN.createVestingSchedule(
>>
                msg.sender, vestingStart, vestingCliff, vestingDuration, vestingSliceP
            );
        }
    }
<...>
    function _safeClaimTransfer
      (IERC20 token, address to, uint256 amount) internal {
>>      uint256 balance = token.balanceOf(address(this));
        bool transferSuccess = false;

        if (amount > balance) {
    >>      transferSuccess = token.transfer(to, balance);
        } else {
            transferSuccess = token.transfer(to, amount);
        }

        if (!transferSuccess) revert TransferFailed();
    }
```

Recommendations

Consider excluding the `_safeClaimTransfer` functionality since it can not be used for the rounding error protection but can cause unexpected behavior and asset losses.

8.2. Low Findings

[L-01] Consider transferring unsold Bio to Treasury

In all other places Treasury is the receiver of tokens: during swaps, emergency withdrawals. However for some reason Admin claims unsold Bio:

```
function withdrawUnsoldTokens() external onlyRole(DEFAULT_ADMIN_ROLE) {
        if (!hasEnded()) revert SaleNotEnded();
        if (unsoldTokensWithdrew) revert TokensAlreadyWithdrawn();

        uint256 totalBIOSold = bioToDistribute();
        uint256 totalVBIOSold = vbioToDistribute();

        unsoldTokensWithdrew = true;
        //because VBIO is BIO and is sent to VBIO contract before claiming
@>      BIO_TOKEN.transfer(
  msg.sender,
  MAX_BIO_TO_DISTRIBUTE+MAX_VBIO_TO_DISTRIBUTE-totalBIOSold-totalVBIOSold
);
    }
```

[L-02] Reasonable values for `START_TIME` and `END_TIME`

There is an insufficient input check of `startTime_` and `endTime_` parameters in the contract constructor. Since the input is used for immutables it is important to check that the parameters have reasonable values. At least `startTime_` and `endTime_` should not be less than block.timestamp and the gap between them is not too small or huge.

```
constructor(
        IERC20 bioToken_,
        TokenVesting vbioToken_,
        IERC20 daoToken_,
        uint256 startTime_,
        uint256 endTime_,
        address treasury_,
        uint256 maxToDistribute_,
        uint256 maxToDistribute2_,
        uint256 minToRaise_
    ) AccessControlDefaultAdminRules(0, msg.sender) {
>>      if (startTime_ >= endTime_) revert InvalidConstructorParameters();
        if (treasury_ == address(0)) revert InvalidConstructorParameters();
        if (address(bioToken_) == address(0) || address(vbioToken_) == address
          (0)) revert InvalidConstructorParameters();

        BIO_TOKEN = bioToken_;
        VBIO_TOKEN = vbioToken_;
        DAO_TOKEN = daoToken_;
>>      START_TIME = startTime_;
>>      END_TIME = endTime_;
        vestingStart = endTime_;
        treasury = treasury_;
        MAX_BIO_TO_DISTRIBUTE = maxToDistribute_;
        MAX_VBIO_TO_DISTRIBUTE = maxToDistribute2_;
        MIN_DAO_RAISED_FOR_MAX_BIO = minToRaise_;
    }
<...>
    function hasStarted() public view returns (bool) {
>>      return _currentBlockTimestamp() >= START_TIME;
    }
<...>
    function hasEnded() public view returns (bool) {
>>      return END_TIME <= _currentBlockTimestamp();
    }
```

The `start` and `endTime` in the constructor can be set in the past.

```
if (startTime_ >= endTime_) revert InvalidConstructorParameters();
```

The `vestingStart` time can also be set in the past.

```
function setVestingStart(uint256 start_) external onlyRole
   (DEFAULT_ADMIN_ROLE) {
        // _start should be no further away than 30 weeks
        if
          (start_ > block.timestamp + 30 weeks) revert InvalidScheduleParameter();
        vestingStart = start_;
    }
```

To prevent this error, check that the start time is greater than `block.timestamp`.

```
For constructor:

+       if (startTime_ < block.timestamp) revert InvalidConstructorParameters();
        if (startTime_ >= endTime_) revert InvalidConstructorParameters();
```

```
For setVestingStart:

 function setVestingStart(uint256 start_) external onlyRole
   (DEFAULT_ADMIN_ROLE) {
        // _start should be no further away than 30 weeks
+       if (start_ < block.timestamp) revert InvalidScheduleParameter();
        if
          (start_ > block.timestamp + 30 weeks) revert InvalidScheduleParameter();
        vestingStart = start_;
    }
```

[L-03] Lack of `DAO` token address check

The `DAO` token address can be set only during deployment. In case the `DAO` address immutable is zero the contract will be useless.

```
constructor(
        IERC20 bioToken_,
        TokenVesting vbioToken_,
        IERC20 daoToken_,
        uint256 startTime_,
        uint256 endTime_,
        address treasury_,
        uint256 maxToDistribute_,
        uint256 maxToDistribute2_,
        uint256 minToRaise_
    ) AccessControlDefaultAdminRules(0, msg.sender) {
        if (startTime_ >= endTime_) revert InvalidConstructorParameters();
        if (treasury_ == address(0)) revert InvalidConstructorParameters();
>>      if (address(bioToken_) == address(0) || address(vbioToken_) == address
  (0)) revert InvalidConstructorParameters();

        BIO_TOKEN = bioToken_;
        VBIO_TOKEN = vbioToken_;
>>      DAO_TOKEN = daoToken_;
        START_TIME = startTime_;
        END_TIME = endTime_;
        vestingStart = endTime_;
        treasury = treasury_;
        MAX_BIO_TO_DISTRIBUTE = maxToDistribute_;
        MAX_VBIO_TO_DISTRIBUTE = maxToDistribute2_;
        MIN_DAO_RAISED_FOR_MAX_BIO = minToRaise_;
    }
```

[L-04] Centralization risk whereby owner can withdraw all BIO tokens

The owner can withdraw all bio tokens in an emergency, which will affect the user's claims.

```
function emergencyWithdrawBIO() external onlyRole(DEFAULT_ADMIN_ROLE) {
        uint256 amount = BIO_TOKEN.balanceOf(address(this));
        BIO_TOKEN.safeTransfer(treasury, amount);

        emit EmergencyWithdraw(address(BIO_TOKEN), amount);
    }
```

Also, the owner can not set `forceClaimable` to true, then users will lose their DAO tokens to the treasury.

[L-05] `swap` and `claim` functions will be unavailable

The `swap` function can not be invoked if the `FairAuctionVesting` has no `VBIO_TOKEN.VESTING_CREATOR_ROLE` role.

```
modifier isSaleActive() {
        if (!hasStarted() || hasEnded()) revert SaleInactive();
        if (BIO_TOKEN.balanceOf(address(this)) <
          (MAX_BIO_TO_DISTRIBUTE + MAX_VBIO_TO_DISTRIBUTE)) revert SaleNotFilled();
        // check if this contract has vesting creator role
>>      if (!VBIO_TOKEN.hasRole(VBIO_TOKEN.VESTING_CREATOR_ROLE(), address
  (this))) revert SaleNotFilled();
        _;
    }
<...>
>>  function swap
  (uint256 amount) external isSaleActive isNotPaused nonReentrant {
```

The `claim` function also reverts due to a check in the `TokenVesting.createVestingSchedule` function for the same condition.

```
function claim() external nonReentrant isClaimable {
        UserInfo storage user = userInfo[msg.sender];

        if (totalRaised == 0 || user.contribution == 0) revert ZeroContribution
          ();
        if (user.hasClaimed) revert AlreadyClaimed();
        user.hasClaimed = true;

        (uint256 bioAmount, uint256 vbioAmount) = getExpectedClaimAmount
          (msg.sender);

        emit Claim(msg.sender, bioAmount, vbioAmount);

        if (bioAmount > 0) {
            // send BIO_TOKEN contribution
            _safeClaimTransfer(BIO_TOKEN, msg.sender, bioAmount);
        }
        if (vbioAmount > 0) {
            // send VBIO_TOKEN contribution
            _safeClaimTransfer(BIO_TOKEN, address(VBIO_TOKEN), vbioAmount);
            // create vesting schedule
>>          VBIO_TOKEN.createVestingSchedule(

                                msg.sender, vestingStart, vestingCliff, vestingDurati
            );
        }
    }
<...>
    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
        );
    }
```

Thus the contract is vulnerable to the actions of the `VBIO_TOKEN` owner.

Though the `VBIO_TOKEN` contract admin can be trusted it is also possible to deploy a new `VBIO_TOKEN` contract from the `FairAuctionVesting` to grant it both `DEFAULT_ADMIN_ROLE` and `VESTING_CREATOR_ROLE` for the vesting token full control.

[L-06] Unexpected vesting schedule when claiming

Though the vesting schedule can be really important for users during auction it can be changed during the whole the contract lifetime or being not set at all.

```
function setVestingStart(uint256 start_) external onlyRole
      (DEFAULT_ADMIN_ROLE) {
        // _start should be no further away than 30 weeks
        if
          (start_ > block.timestamp + 30 weeks) revert InvalidScheduleParameter();
        vestingStart = start_;
    }

    function setCliff(uint256 cliff_) external onlyRole(DEFAULT_ADMIN_ROLE) {
        // _duration must be longer than _cliff
        if (vestingDuration < cliff_) revert InvalidScheduleParameter();
        vestingCliff = cliff_;
    }

    function setDuration(uint256 duration_) external onlyRole
      (DEFAULT_ADMIN_ROLE) {
        // _duration should be at least 7 days and max 50 years
        if (duration_ < 7 days || duration_ > 50 *
          (365 days)) revert InvalidScheduleParameter();
        vestingDuration = duration_;
    }

    function setSlicePerSecond(uint256 slicePerSecond_) external onlyRole
      (DEFAULT_ADMIN_ROLE) {
        // _slicePeriodSeconds should be between 1 and 60 seconds
        if
          (slicePerSecond_ == 0 || slicePerSecond_ > 60) revert InvalidScheduleParamet
        vestingSlicePerSecond = slicePerSecond_;
    }

    function setIsRevocable(bool isRevocable_) external onlyRole
      (DEFAULT_ADMIN_ROLE) {
        vestingIsRevocable = isRevocable_;
    }
```

Consider setting a vesting schedule only before the sale begins.