Pirex Security Review

Pashov Audit Group

Conducted by: pashov

October 1st, 2023

Contents

1. About pashov

Krum Pashov, or **pashov**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Check his 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 I try to find as many vulnerabilities as possible. I 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 **Pirex** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Pirex

Pirex allows ETH holders to receive staking rewards through pooling. Staking through Pirex you benefit through block validation/attestation & MEV yield and compounding rewards through `pxETH`. The `pxETH` token is the Pirex-wrapped version of ETH, where its holders can earn yield by staking in `AutoPxETH` or just redeem their tokens for ETH.

More docs

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* - 12b25265e17d141faf1c10fa84d9826838107e67**

***fixes review commit hash* - 277ca47a598694a51261beac86dc21ec4937a699**

7. Executive Summary

Over the course of the security review, pashov engaged with Pirex to review Pirex. In this period of time a total of **5** issues were uncovered.

Protocol Summary

Protocol Name	Pirex
Date	October 1st, 2023

Findings Count

Severity	**Amount**
Medium	1
Low	4
Total Findings	**5**

Summary of Findings

ID	Title	Severity	Status
[M-01]	The emergencyWithdraw method can leave PirexEth in a broken state	Medium	Resolved
[L-01]	Updating withdrawalCredentials is flawed	Low	Acknowledged
[L-02]	Penalty calculation in previewRedeem rounds in the wrong direction	Low	Resolved
[L-03]	Upper bound setter checks missing	Low	Partially Resolved
[L-04]	Setting allowances to 0 should be a valid operation	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] The `emergencyWithdraw` method can leave `PirexEth` in a broken state

Severity

Impact: High, as the logic in `PirexEth` will be broken

Likelihood: Low, as it requires an emergency and using the contract after it

Description

The `emergencyWithdraw` method in `PirexEth` allows for withdrawal of ETH. This ETH could have been the `pendingDeposit` balance, which is not yet deposited to the ETH 2.0 deposit contract, and if it is withdrawn from the `emergencyWithdraw` method then the contract will be in a broken state. The `pendingDeposit` variable will have a value that is more than the ETH balance in the contract which will make deposit transactions revert if they are used post `emergencyWithdraw` call.

Recommendations

Change the `emergencyWithdraw` method so that it can withdraw only excessive balance without the `pendingDeposit` one, or when using `pendingDeposit` force it to withdraw the whole balance and zero out the state variable.

8.2. Low Findings

[L-01] Updating `withdrawalCredentials` is flawed

The `setContract` method in `PirexEthValidators` updates the `withdrawalCredentials` state variable when the `RewardRecipient` contract address is updated. If there already are validators in the `_initializedValidators` array, they would have been configured with different `withdrawalCredentials`. This is not an issue, as if a validator has deposited even 1 ETH, when depositing more the `withdrawalCredentials` argument is not used in the deposit contract, but ignored. Still, for extra safety and logical correctness, make sure to only allow updating `withdrawalCredentials` when there are no elements in the `_initializedValidators` array.

[L-02] Penalty calculation in `previewRedeem` rounds in the wrong direction

The `previewRedeem` method in `AutoPxEth` calculates the penalty with the following code:

```
uint256 penalty = (_totalSupply == 0 || _totalSupply - shares == 0)
    ? 0
    : assets.mulDivDown(withdrawalPenalty, FEE_DENOMINATOR);
```

The issue is that the math rounds down the penalty, but following the ERC4626 standard you should follow best security for the vault and when calculating the assets a user will receive when burning shares, the assets amount should be rounded down, where here since the penalty is rounded down, but penalty is subtracted from the `assets`, now the opposite effect happens. Make sure to use `mulDivUp` instead `mulDivDown` in the `penalty` calculation.

[L-03] Upper bound setter checks missing

In `PirexEth` the `setMaxFee` method is missing an upper bound. Setting the max fee to be more than 100% allows setting the fee to be more than 100% as well, which shouldn't be possible. Add a max upper bound for the `setMaxFee` method, for example 50%.

Same issue with the `setMaxProcessedValidatorCount` method in `PirexEthValidators`, where putting a very high value can lead to a DoS of deposits. Add a max value for the `_count` argument, for example 20.

[L-04] Setting allowances to 0 should be a valid operation

The `operatorApprove` method of `PxEth` currently reverts when the allowance amount is 0 which is a logical error - removing allowance (or setting it to 0) should be a valid operation. Remove the following line from `operatorApprove`:

```
if (_amount == 0) revert Errors.ZeroAmount();
```