Protectorate Security Review

Pashov Audit Group

Conducted by: pashov

July 12th, 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 **Protectorate** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Protectorate

The Protectorate protocol is a yield aggregation platform that uses strategies targeting NFT/NFTfi protocols on Ethereum mainnet. Users provide liquidity and the platform decides where to deploy it for the highest yield possible across the ecosystem. The native token of the protocol `$PRTC` will initially be used for revenue sharing and eventually governance.

Protocol's Litepaper

Observations

The docs say 80% of yields will be going to depositors and 20% will be a "performance fee", where 10% will go to `xPTRC` stakers and 10% will go to the treasury. This 20% split happens in `Staking::distribute`.

Investors and treasury accounts won't have a cliff when it comes to vesting.

The `DutchAuction` contract trusts that it will be pre-funded with `PRTC` tokens.

The `LendingVault` contract is using the ERC46262 standard, which has common flaws related to it.

Privileged Roles & Actors

- Envoy (liquidity provider) - deposits an asset into individual Capsules (NFTs, ETH, Stablecoins etc)
- Elder (xPRTC staker) - earns revenue from the strategy's yield
- Vesting owner - can create vesting schedules and withdraw excessive funds from `Vesting`
- Vesting beneficiary - vests tokens over time depending on their configured schedule
- Staker - stakes `PRTC` with the expectation of receiving `WETH` rewards
- Staking `WETH` rewards distributor - deposits `WETH` into the `Staking` contract
- Treasury - receives half of the staking distributed `WETH`
- Vault depositor - deposits assets into the vault with expectation of a yield from strategies
- Bend Dao strategy owner - can deposit, withdraw and swap rewards for assets in the strategy contract
- DutchAuction participant - commits `ETH` to buy `PRTC` tokens
- DutchAuction beneficiary - sends `PRTC` tokens as balance to the contract and on auction finalization receives `ETH` on success and the initial deposit back on failure
- PRTC token recipient - gets the whole total supply balance on `PRTC` deployment

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 - **de93a8aff725b69f9f525858f6fdaefe92c8f966**

fixes review commit hash - **0888811ffcadff0c1385823b27897f02f04c2a2d**

Scope

The following smart contracts were in scope of the audit:

- `interfaces/**`
- `strategies/BendDaoLendingStrategy`
- `utils/Constants`
- `vaults/LendingVault`
- `DutchAuction`
- `ETHRouter`
- `PRTC`
- `StakingContract`
- `Vesting`

7. Executive Summary

Over the course of the security review, pashov engaged with Protectorate to review Protectorate. In this period of time a total of **8** issues were uncovered.

Protocol Summary

Protocol Name	Protectorate
Date	July 12th, 2023

Findings Count

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

Summary of Findings

ID	Title	Severity	Status
[C-01]	If strategy is losing money the last person left to claim from vault will handle all losses	Critical	Resolved
[H-01]	Vesting schedule for a beneficiary can be overwritten	High	Resolved
[H-02]	Vault depositors can be front-ran and lose their funds	High	Resolved
[M-01]	Some vesting recipients temporarily won't be able to claim	Medium	Resolved
[M-02]	Insufficient input validation can lead to loss of funds	Medium	Resolved
[L-01]	Strategy contract is incompatible with non-standard ERC20 tokens	Low	Resolved
[L-02]	Division before multiplication can lead to rounding errors	Low	Resolved
[L-03]	Trust assumption in DutchAuction	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] If strategy is losing money the last person left to claim from vault will handle all losses

Severity

Impact: High, as some users will bear substantial value losses

Likelihood: High, as it is possible that strategy is losing money at a given time

Description

Currently, the way that `LendingVault` is designed, is that the funds in the vault are transferred out to chosen strategies. Due to the fact that users can still withdraw funds from the vault's balance while some of the funds are lent out to a strategy, the following scenario can happen:

1. Alice deposits 100 ETH to the Vault
2. Bob deposits 100 ETH to the Vault
3. Strategy requests 200 ETH from the Vault - now Vault balance is 0, Strategy balance is 200
4. Chris deposits 100 ETH to the Vault
5. Strategy is not doing good and is left with 100 ETH balance
6. Bob sees this, and withdraws his share, which is 100 ETH
7. Now the strategy is losing but funds are returned back, leaving the Vault with 100 ETH balance and Strategy with 0 balance
8. Now Alice and Chris will bear all of the loss of the strategy, while Bob managed to get 100% of his initial deposit despite of the loss.

Recommendations

Possibly forbid withdraws while funds are lent out to a strategy or think of another design for Vault-Strategy lending.

8.2. High Findings

[H-01] Vesting schedule for a beneficiary can be overwritten

Severity

Impact: High, as the amount left to be vested will be stuck in the contract forever

Likelihood: Medium, as it requires more than 1 vesting schedule for the same beneficiary

Description

The vesting schedules in `Vesting` are saved in `schedules` mapping, which uses the `_beneficiary` address as the key. The problem is that if a beneficiary has a scheduled vesting already, if a second schedule is set to it, then the first one will be overwritten but the `schedulesTotalAmount` will still hold the first scheduled funds to vest. This means they will be stuck in the `Vesting` contract forever.

Recommendations

A possible solution is to use a vesting ID instead of the `beneficiary` address as the key in the `schedules` mapping or to disallow multiple schedules set for the same `beneficiary`.

[H-02] Vault depositors can be front-ran and lose their funds

Severity

Impact: High, as a theft of user assets is possible

Likelihood: Medium, as it works only if the attacker is the first vault depositor

Description

The following attack is possible:

1. The `LendingVault` contract has just been deployed and has 0 `assets` and `shares` deposited/minted
2. Alice sends a transaction to deposit `10e18` worth of `_assets`
3. Bob is a malicious user/bot and sees the transaction in the mempool and front-runs it by depositing 1 wei worth of the asset, receiving 1 wei of shares
4. Bob also front-runs Alice's transaction with a direct ERC20::transfer of 10e18 worth of the asset to the `LendingVault` contract
5. Now in Alice's transaction, the code calculates Alice's shares as `shares = assets.mulDiv(totalSupply(), totalAssets(), Math.Rounding.Down)` where the `totalAssets` returns `_asset.balanceOf(address(this))` which makes shares round down to 0
6. Alice gets minted 0 shares, even though she deposited 10e18 worth of the asset
7. Now Bob back-runs Alice's transaction with a call to `withdraw`` where assets` is the contract's balance of the asset, allowing him to burn his 1 share and withdraw his deposit + Alice's whole deposit

This can be replayed multiple times until the depositors notice the problem.

Recommendations

First, make sure that all deposits will go through Flashbots so the transactions are not sandwhichable/front-runnable.

Then we can look at how UniswapV2 fixed this with two types of protection:

First, on the first `mint` it actually mints the first 1000 shares to the zero-address

Second, it requires that the minted shares are not 0

Implementing all of those solutions will resolve this vulnerability.

8.3. Medium Findings

[M-01] Some vesting recipients temporarily won't be able to claim

Severity

Impact: Medium, as funds will be locked for 30 days

Likelihood: Medium, because it will only happen when the cliff is < 30 day

Description

The `SLICE_PERIOD` constant in `Vesting` is set to 30 days. Due to the following math in `_computeReleasableAmount`

```
uint256 vestedSeconds = (timeFromStart / SLICE_PERIOD) * SLICE_PERIOD;
```

If `timeFromStart` is less than 30 days this will round down to zero, which means the amount to claim until 30 days have passed will always be zero. This applies especially for vesting schedules that have no `cliff` (it is 0), which is expected for `Investors` and `Treasury`.

Recommendations

Make the `SLICE_PERIOD` smaller, or implement another design for handling no cliff vesting schedules that won't be using this calculation.

[M-02] Insufficient input validation can lead to loss of funds

Severity

Impact: High, as funds might be stuck forever in contracts

Likelihood: Low, as it requires a configuration error from the admin

Description

Multiple places in the codebase have insufficient input validation that can lead to stuck funds.

1. The `performanceFee` in `LendingVault` constructor is not validated as it is in the `adjustPerformanceFee` setter
2. The `_duration` parameter in `Vesting::createSchedule` is not validated, it can be too large or too small
3. The `AuctionDetails` set in the constructor of `DutchAuction` has timestamps and prices that can be too large or too small

If either `_duration` in `Vesting` is too big or the difference between `startTime` and `endTime` in `DutchAuction` is too big then funds can be stuck forever in the contracts.

Recommendations

Call the `adjustPerformanceFee` method in `LendingVault`'s constructor to use its input validation. When it comes to the `_duration` parameter in `createSchedule`, use a minimum of 7 days and a maximum of for example 2 years.

For the `AuctionDetails` you need to make check multiple things:

1. `startTime` and `endTime` have not passed already
2. `endTime` is after `startTime`
3. `endTime` is not too far away in the future - maybe max of 15 days

Same things for `startPrice` and `minimumPrice`:

1. Check that `minimumPrice` is smaller than `startPrice`
2. Check that their values are not too big

8.4. Low Findings

[L-01] Strategy contract is incompatible with non-standard ERC20 tokens

Some tokens (for example `USDT`) do not follow the ERC20 standard correctly and do not revert a `bool` on `approve` call. Such tokens are incompatible with the `BendDaoLendingStrategy` contract as the `approve` calls in `deposit` & `withdraw` will always revert due to zero return data. Use `SafeERC20`'s `forceApprove` instead

[L-02] Division before multiplication can lead to rounding errors

In `DutchAuction::priceFunction` there is division before multiplication here:

```
uint256 priceDeclinePerSecond =

              (auctionDetails.startPrice - auctionDetails.minimumPrice) / auctionDurat

return auctionDetails.startPrice - (timeSinceAuctionStart *
priceDeclinePerSecond);
```

This can lead to rounding errors but without a serious impact to the application. Still you should always do multiplication before division in Solidity.

[L-03] Trust assumption in `DutchAuction`

The `DutchAuction` contract expects to be holding `auctionDetails.totalTokens` balance of `PRTC` to work correctly, but this is not validated in any way. Also, on failed auction, `auctionDetails.totalTokens` is the amount of `PRTC` sent to the beneficiary, and if the amount is even 1 wei less it would revert. I suggest using `prtc.balanceOf(address(this))` instead and also you can consider

transferring the `PRTC` tokens into the contract in its constructor so it is trustless.