USP Security Review

Reviewer

Undisclosed

April 8, 2025

Contents

1 Executive Summary

Pareto engaged with to review over the course of 5 business days in total.

Summary

Type of Project	Defi
Timeline	2nd April, 2025 - 7th April, 2025
Methods	Manual Review

A comprehensive security review identified a total of 11 issues.

Repository	Initial Commit
USP	7b246ae66bf9414ad568ebe930e9da0c5029011e

Total Issues

High Risk	0
Medium Risk	3
Low Risk	8

The reported vulnerabilities were addressed by the Pareto team.

Repository	Final Commit
USP	7b246ae66bf9414ad568ebe930e9da0c5029011e

2 Protocol Summary

USP is a synthetic dollar that is backed primarily by Pareto Credit Vaults. Pareto Credit Vaults are a collection of institutional-grade lending strategies that generate yield on the underlying assets via proprietary strategies of institutional borrowers.

3 Findings

3.1 Medium Risk

3.1.1 Incorrect validation while depositing funds to yield sources

Severity: Medium

Context: ParetoDollarQueue.sol#L524

Description: When the manager deposits funds to yield sources using the `depositFunds()` function, it validates the unlent balance is not less than `totReservedWithdrawals` which indicates the total amount of withdrawal requests including the current epoch.

```
    // check if collaterals balances (scaled to 18 decimals) are greater than the totReservedWithdrawals
    if (getUnlentBalanceScaled() < totReservedWithdrawals) {
      revert InsufficientBalance();
}
```

This function can be called with 0 amount to enable a claim for `epochNumber - 1`.

In this case, the above validation is incorrect because the unlent balance doesn't need to cover the current epoch's withdrawal requests.

Impact: The `depositFunds()` wouldn't work properly due to the incorrect validation.

Recommendation: Recommend changing the validation like the below.

```
--  if (getUnlentBalanceScaled() < totReservedWithdrawals) {
++  if (getUnlentBalanceScaled() < totReservedWithdrawals - epochPending[epochNumber]) {
        revert InsufficientBalance();
    }
```

Pareto: Fixed at commit 8acfed3.

3.1.2 The `depositRewards()` handles the unvested rewards incorrectly

Severity: Medium

Context: ParetoDollarStaking.sol#L197

Description: The `depositRewards()` marks the previous unvested rewards as vested. This will cause an issue because there will be a cliff in the `totalAssets()` calculation and some users might claim rewards unfairly by depositing before calling `depositRewards()`.

```
  /// @dev if method is called when prev rewards are not yet vested, old rewards become vested
  /// @param amount The amount of rewards to deposit.
  function depositRewards(uint256 amount) external
```

Impact: The rewards might be distributed incorrectly.

Recommendation: Recommend starting a new vesting including the previous unvested rewards.

Pareto: Fixed at commit f3759b8.

3.1.3 Users might avoid losses by frontrunning `emergencyWithdraw()`

Severity: Medium

Context: EmergencyUtils.sol#L89

Description: If one of the borrowers fails to deliver the funds back, `sUSP` holders will cover the losses. This will be done by the owner who can redeem `USP` via `emergencyWithdraw()` from the `sUSP` contract and then burn `USP` in the `ParetoDollar` contract.

```
 function emergencyWithdraw(address token, uint256 amount) external {
   _checkOwner();
   IERC20(token).safeTransfer(msg.sender, amount);
}
```

But some `sUSP` holders might withdraw their `sUSP` before the owner calls `emergencyWithdraw()` to avoid the losses.

Impact: The loss socialization mechanism might be bypassed.

Recommendation: Recommend to implement a delay period when withdrawing `sUSP`.

Pareto: Acknowledged. This may be prevented by sending the tx via a private mempool to avoid the tx frontrunning.

3.2 Low Risk

3.2.1 Lack of validation in `updateRewardsVesting()`

Description: The `updateRewardsVesting()` should be called after all rewards are vested with the old and new `rewardsVesting` periods. Otherwise, there will be a cliff in the `totalAssets()` calculation.

```
function updateRewardsVesting(uint256 _rewardsVesting) external {
    _checkOwner();
    rewardsVesting = _rewardsVesting;
}
```

Pareto: Fixed at commit e88aeb7.

3.2.2 The same collateral token might be added to the `collaterals` list twice

Description: The `addCollateral()` should check if the `token` is added already. Otherwise, the same token might be added to the `collaterals` list twice, and the total collaterals might be calculated incorrectly.

```
 function addCollateral(
   address token,
   uint8 tokenDecimals,
   address priceFeed,
   uint8 priceFeedDecimals,
   address fallbackPriceFeed,
   uint8 fallbackPriceFeedDecimals
) external {
   _checkOwner();

   if (token == address(0) || priceFeed == address(0)) revert InvalidData();
   collateralInfo[token] = CollateralInfo({
       allowed: true,
       priceFeed: priceFeed,
       fallbackPriceFeed: fallbackPriceFeed,
       tokenDecimals: tokenDecimals,
       priceFeedDecimals: priceFeedDecimals,
       fallbackPriceFeedDecimals: fallbackPriceFeedDecimals
   });
}
```

Pareto: Fixed at commit 81a1d84.

3.2.3 Incorrect validation when `maxCap` equals zero

Description: The `Deploy.s.sol` says 0 `maxCap` means an unlimited cap but there is no relevant logic in the queue contract.

```
File: Deploy.s.sol
98:     // add sky.money USDS-USDC PSM as a "yield source" with 0 max cap (ie unlimited)

File: ParetoDollarQueue.sol
514:       if (_yieldSource.depositedAmount > _yieldSource.maxCap) {
515:         revert MaxCap();
516:       }
```

Pareto: Fixed at commit c587a0d.

3.2.4 The same oracle validity period is used for all oracles

Description: Each collateral token has two price feeds - primary and fallback, and the same validity period is used for these price feeds of several stablecoins. The current validity period(24 hours) is good for Chainlink price feeds, but the fallback price feeds might have different heartbeats.

```
File: ParetoDollar.sol
203:     uint256 _oracleValidity = oracleValidityPeriod;
204:     // if validity period is 0, it means that we accept any price > 0
205:     // othwerwise, we check if the price is updated within the validity period
206:     if (answer > 0 && (_oracleValidity == 0 || (updatedAt >= block.timestamp - _oracleValidity))) {
207:       return uint256(answer) * 10 ** (18 - feedDecimals);
208:     }
```

It would be good to set an individual validity period for each oracle feed rather than using a global period.

Pareto: Fixed at commit b99fe32.

3.2.5 The `MANAGER_ROLE` is not used in the `ParetoDollarStaking` contract

Description: The `ParetoDollarStaking` contract doesn't need to have a `MANAGER_ROLE` role.

```
File: ParetoDollarStaking.sol
54:    /// @notice role for managing the contract (can deposit rewards)
55:    bytes32 public constant MANAGER_ROLE = keccak256("MANAGER_ROLE");
```

Pareto: Fixed at commit 6ce73c9.

3.2.6 Possible underflow in `totalAssets()`

Description: The `totalAssets()` might revert due to underflow. When some underlying tokens are withdrawn by the owner during an emergency, the staking contract might have fewer assets than `unvestedRewards`. In this case, `totalAssets()` should return 0 rather than reverting.

```
File: ParetoDollarStaking.sol
150:     // return total assets minus unvested rewards
151:     return IERC20(asset()).balanceOf(address(this)) - unvestedRewards;
```

Pareto: Fixed at commit 51ca1b9.

3.2.7 Lack of validation while removing a yield source

Description: It would be good to add a validation while removing a yield source.

```
  /// @dev only the owner can call this function. Yield source should be
  /// removed only when everything is withdrawn from it.
  /// @param _source The address of the yield source.
  function removeYieldSource(address _source) external
```

Everything should be withdrawn before removing the yield source. However, right before the owner removes the yield source of zero funds, one of the managers might deposit funds into it and the above assumption might be broken.

Pareto: Fixed at commit cba3477.

3.2.8 Lack of validations with the `MANAGER_ROLE` functions

Description: There are several implicit assumptions associated with three functions that interact with the yield sources. If one of these assumptions is broken due to a mistake by the manager, it will have a serious impact on the protocol.

- As the comment has mentioned, `callWhitelistedMethods()` shouldn't be used for deposits/redeems.

```
File: ParetoDollarQueue.sol
534:    /// @notice Call multiple whitelisted methods on yield sources.
535:    /// @dev only the manager can call this function. This should not be used for deposits/redeems
↪   which have
536:    /// their own functions and account for balance changes
```

- Another potential risk is that `redeemFunds()` might be used to call `PSM.buyGem()`.

```
    balPre = _yieldSource.token.balanceOf(address(this)); // USDC balance
    // do call on the yield source
    _externalCall(source, method, _args[i]); // swap USDS to USDC
    // calculate redeemed amount
    redeemed = _yieldSource.token.balanceOf(address(this)) - balPre; // increased USDC balance
```

In this case, the swapped `USDC` will be considered as `Redeemed` which will update depositedAmount and epochPending incorrectly.

Pareto: Fixed at commit 7b246ae.