SOLIDIFIED

Audit Report for Pods Finance - July 12, 2021

Summary

Audit Report prepared by Solidified covering the Pods Finance smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on July 12, 2021, and the results are presented here.

Audited Files

The source code has been supplied in a public source code repository:

https://github.com/pods-finance/contracts (branch: `develop`)

Commit number: `8549326c93e5542438ff75ed9f8075952a505f8e`

UPDATE: Latest Fixes received on August 24th in PR:
https://github.com/pods-finance/contracts/pull/300

Final commit number: `6145d25a2faf5b09b834203183e32ef95c012cd3`

Intended Behavior

Pods Finance is a decentralized non-custodial options protocol on Ethereum.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Issues Found

Solidified found that the Pods Finance contracts contain no critical issues, no major issues, 1 warning, 5 minor issues, and 8 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	AMM.sol: Accounting logic does not properly handle cases where a user deposits directly into AMM via token.transfer()	Warning	Acknowledged
2	RequiredDecimals.sol: maximum decimals amount should be reduced to avoid potential overflows	Minor	Resolved
3	OptionAMMPool.sol/AMM.sol: Arbitrary addresses can update a user's fImp value by adding liquidity on their behalf	Minor	Resolved
4	FeePool.sol: Validation mismatch between function setFee() and the contract's constructor	Minor	Resolved
5	IPodOption.sol: Documentation discrepancy for function unmint()	Minor	Resolved
6	FlashloanProtection.sol: Function _nonReentrant() is an insecure re-entrancy guard	Minor	Resolved
7	PodOption.sol: Function underlyingAssetDecimals() omits using tryDecimals()	Note	Resolved
8	WPodCall / WPodPut: Redundant underlyingAsset constructor parameter	Note	Resolved
9	FeePool.sol: Functions balanceOf() and sharesOf() parameter owner shadows parent	Note	Resolved

	contract		
10	BlackScholes.sol: Function _getZScores() does not use PRBMathSD59x18 consistently	Note	Resolved
11	PriceProvider.sol.sol: Consider adding an additional updateAssetFeeds() function	Note	Resolved
12	OptionHelper.sol: Function _mint() does not validate parameter option	Note	Resolved
13	NormalDistribution.sol: Use named constants instead of magic numbers	Note	-
14	Misc Notes	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Warnings

1. `AMM.sol`: Accounting logic does not properly handle cases where a user deposits directly into AMM via `token.transfer()`

The accounting logic in the `addLiquidity()` and `removeLiquidity()` functions relies on calling `balanceOf(this)` to compute reserves, and uses the ratio between that and internally maintained “amortized” balance variables to derive FImp and price multipliers. Given that this balance can change “transparently” in the background without any calls to `addLiquidity()`, it leaves open the possibility that the accounting can be skewed by a malicious actor to manipulate the return of `_getWithdrawAmounts()` or skew the amount that is captured by the fee calculation in the `onAddLiquidity()` hooks.

Recommendation
Separate state variables should be used to track the actual reserves deposited and removed at the end of each `addLiquidity()` and `removeLiquidity()` calls, and if the balances have changed transparently by someone directly depositing since the last invocation of either, the internal auxiliary variables that keep track of amortized balances should be appropriately scaled.

Status
Acknowledged. Team's response: "*No real exploit scenario was found. If someone transfers tokens directly to the contract, It will only increase the total balances, resulting in an impermanent gain distributed evenly for the LPs*".

Minor Issues

2. RequiredDecimals.sol: maximum decimals amount should be reduced to avoid potential overflows

Function `tryDecimals()` currently enforces a maximum of 77 decimal places. Since these already take up the entire 2^256 digit space, any multiplication operation will cause an overflow (revert when using SafeMath).

Recommendation
Maximum decimal places should be at most 38 (~77/2).

Status
Resolved

3. OptionAMMPool.sol/AMM.sol: Arbitrary addresses can update a user's fImp value by adding liquidity on their behalf

Since the `addLiquidity()` function is permissionless, this can change the calculation at the subsequent `removeLiquidity()` call, since that function fetches the FImp value stored from the last call. This can potentially be used to grief other users by updating that value at unfavorable moments.

Recommendation
Only allow `msg.sender` to add liquidy for themselves.

Status
Resolved

4. FeePool.sol: Validation mismatch between function setFee() and the contract's constructor

The `setFee()` function does not validate parameters `feeBaseValue` and `decimals` as per the contract's constructor validation.

Recommendation
Have `setFee()` require that `feeDecimals <= 77 && feeBaseValue <= uint256(10)**feeDecimals`.

Status
Resolved

5. IPodOption.sol: Documentation discrepancy for function unmint()

The documentation for function `unmint()` states that the caller might receive a mix of underlying asset and strike asset in case of American options, while function `PodOption._unmintOptions()` never implements this.

Recommendation
Update `PodOption._unmintOptions()` implementation to match the specification required for American options.

Note
Same issue exists in `AaveCallPut.unmintWithRewards()` and `AavePodPut.unmintWithRewards()`.

Status
Resolved

6. FlashloanProtection.sol: Function _nonReentrant() is an insecure re-entrancy guard

Function `_nonReentrant()` is an insecure re-entrancy guard since it uses `tx.origin`.

Recommendation
Use a modifier with the `msg.sender` for checks. Separate out re-entrancy protection from flash loan protection.

Status
Resolved

Informational Notes

7. PodOption.sol: Function underlyingAssetDecimals() omits using tryDecimals()

Function `underlyingAssetDecimals()` raw calls the ERC20 for the decimals.

Recommendation
The decimal call should be done via `RequiredDecimals.tryDecimals()` for consistent restrictions of decimal precision.

Status
Resolved

8. `WPodCall` / `WPodPut`: Redundant `underlyingAsset` constructor parameter

Both `WPodCall` and `WPodPut` require the `underlyingAsset` parameter in their constructor, even though `underlyingAsset` will always be equal to the `WETH` contract.

Recommendation
To eliminate redundancy and potential mistakes, consider removing the `underlyingAsset` constructor requirement and directly setting the `underlyingAsset` to the `WETH` contract.

Status
Resolved

9. `FeePool.sol`: Functions `balanceOf()` and `sharesOf()` parameter owner shadows parent contract

The functions `balanceOf()` and `sharesOf()` parameter owner shadows the `owner` storage variable declared in the parent `Ownable` contract.

Recommendation
Consider renaming the functions' parameters.

Status
Resolved

10. BlackScholes.sol: Function _getZScores() does not use PRBMathSD59x18 consistently

Consider adding `PRBMathSD59x18` to the divisions and additions that are currently not using the safe mathematical operations (lines 136, 138, 142).

Status
Resolved

11. PriceProvider.sol.sol: Consider adding an additional updateAssetFeeds() function

Consider adding a new function `updateAssetFeeds()` in addition to `setAssetFeeds()` that can only be called on assets that already have a feed, such as to not accidentally overwrite an existing asset feed when adding new ones.

Status
Resolved

12. OptionHelper.sol: Function _mint() does not validate parameter option

Consider validating the `option` parameter in order to avoid any unexpected behaviour.

Status
Resolved

13. `NormalDistribution.sol`: Use named constants instead of magic numbers

Consider using named constants instead of magic numbers for the thresholds in `getProbability()`. Also consider documenting how these values were chosen and how the code was generated so that it can be validated

14. Misc Notes

- AMM.sol: Function name `_isRecipient()` can be misleading. Consider renaming the function to `_isValidAddress()` or `_isNonZeroAddress()` instead.
  - Status: Resolved.
- OptionAMMPool.sol: Function name `_getTradeInfo()` can be misleading. Consider renaming the function to `_emitTradeInfo()` instead.
  - Status: Resolved.
- NormalDistribution.sol: Incorrect documentation for function `_abs()`.
  - Status: Resolved.
- Consider using Solidity's latest compiler version with built-in safe math operations instead of relying on OpenZeppelin's SafeMath library.
  - Status: Acknowledged. Team's response: "The team prefered to keep with the old compiler version especially because Echidna (Our fuzzying tool) still dont support solidity ^0.8 with asserts".

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Pods Finance or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.