Report

v. 1.0

Customer

Pods Finance

Smart Contract Audit

Pods yield

6th October 2022

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	05.10.22	A. Zveryanskaya	Initial Draft
0.2	05.10.22	A. Zveryanskaya	Minor revision
1.0	11.10.22	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Following a formal product description, the first strategy on Pods Yield, stETHvv (Ethereum Volatility Vault) is a low-risk product focused on ETH accumulation. It combines Lido's yield with weekly strangles to earn more every time the ETH price bounces up or down.

3 Project scope

We were asked to review:

- Original Repository
- Fix Repository

Files:

configuration/				
	Configuration Manager.sol			
interfaces/				
	IConfiguration Manager.sol	ICurve Pool.sol	IERC4626.sol	IVault.sol
	IVaultMetadata.sol			
libs/				
	CastUint.sol	Deposit QueueLib.sol	FixedPoint Math.sol	
mixins/				
	Capped.sol			
proxy/				
	ETHAdapter.sol	Migration.sol		
vaults/				
	BaseVault.sol	STETH Vault.sol		

4 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and used technologies, as well as on what the client is expecting from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

5 Our findings

We found 1 critical, 10 major, and a few less important issues. All identified Critical issues have been fixed.

Issues	Active	Fixed
Severity		
Critical	**0**	**1**
Major	**6**	**4**
Moderate	**2**	**2**
Minor	**22**	**14**

Fixed 21 out of 51 issues

6 Critical Issues

CVF-13. FIXED

- **Category** Flaw
- **Source** BaseVault.sol

Description Here the assets amount is reduced by the fee, while it should actually be increased, as the higher fee is, the more shares a user needs to burn to get the same amount of assets.

```
165 return convertToShares(assets - _getFee(assets));
```

7 Major Issues

CVF-8. FIXED

- **Category** Flaw
- **Source** STETHVault.sol

Description In case the returned ”assets” value differs from the ”assets” value passed to the call, the ”shares” value is not updated accordingly.

Recommendation Consider either requiring that the actual obtained assets amount is the same as the desired amount, or updating the ”shares” value in case the actual obtained amount is different.

```
120 assets = _tryTransferSTETH(msg.sender, assets);
```

CVF-10. INFO

- **Category** Suboptimal
- **Source** BaseVault.sol

Description The expression ”_asset.symbol()” is calculated several times.

Recommendation Consider calculating once and reusing.

```
40 ERC20(string(abi.encodePacked("Pods Yield ", _asset.symbol())),
      ↪ string(abi.encodePacked("py", _asset.symbol())))
   ERC20Permit(string(abi.encodePacked("Pods Yield ", _asset.symbol()))
      ↪ )
```

CVF-11. INFO

- **Category** Suboptimal
- **Source** BaseVault.sol

Description Calling the ”asset.decimals()” contract each time is suboptimal.

Recommendation Consider calling once in the constructor and storing in an immutable variable.

```
60 return asset.decimals();
```

CVF-21. INFO

- **Category** Suboptimal
- **Source** Migration.sol

Description Approving each time is waste of gas.

Recommendation Consider approving a large amount once in the constructor.

Client Comment *We are not prioritazing gas optimizations for this contract because the migration contract should be an once in a lifetime event*

```
27 asset.safeApprove(address(to), balance);
```

```
43 asset.safeApprove(address(to), balance);
```

CVF-23. FIXED

- **Category** Flaw
- **Source** Migration.sol

Description The ”shares” value should be exactly the same as was used by the signer when creating the signature. A malicious user may send a few shares to the ”from” address to make the signature useless.

Recommendation Consider passing the ”shares” value as an argument.

```
37 uint256 shares = from.balanceOf(msg.sender);
   IERC20Permit(address(from)).permit(msg.sender, address(this), shares
   ↪ , deadline, v, r, s);
```

CVF-26. FIXED

- **Category** Suboptimal
- **Source** DepositQueueLib.sol

Description Her a value in the storage is updated multiple times.

Recommendation Consider calculating the new ”totalDeposited” value in memory and then writing into the storage once.

```
45 queue.totalDeposited -= queue.cache[queue.list[startIndex]];
```

CVF-28. INFO

- **Category** Unclear behavior
- **Source** DepositQueueLib.sol

Description In case of an invalid index, this function silently does nothing.

Recommendation Consider reverting in such a case.

Client Comment *Mitigated by _mint() function which won't mint tokens to address(0).*

```
60 function get(DepositQueue storage queue, uint256 index) internal
      ↪ view returns (DepositEntry memory depositEntry) {
```

CVF-29. FIXED

- **Category** Flaw
- **Source** ETHAdapter.sol

Description It is not ensured that the pool's token #0 is ether, while this contract assumes this fact.

Recommendation Consider adding an explicit check for this.

```
17 constructor(ICurvePool _pool) {
```

CVF-39. INFO

- **Category** Unclear behavior
- **Source** CastUint.sol

Description This function silently drops the higher 96 bits of the argument.

Recommendation Consider reverting in case these bits are not zero.

Client Comment *Since addresses comes from ConfigurationManager we don't see it as harmful.*

```
9 function toAddress(uint256 value) internal pure returns (address) {
```

CVF-48. INFO

- **Category** Unclear behavior
- **Source** IVault.sol

Description This argument allows processing deposits out of order.

Recommendation Consider removing this argument and always processing deposits in order.

Client Comment *We plan to process the deposits for the users of the protocol for free. In that case, we want to be able to skip certain deposits in case we receive a spam attack. If the caller think its an unfair judgment, it can always process his own deposit.*

```
74 * @param startIndex Zero-based index at which to start processing
     ↪ deposits
```

8 Moderate Issues

CVF-24. FIXED

- **Category** Flaw
- **Source** DepositQueueLib.sol

Description It is not ensured that ”amount” is not zero, so it is possible to add the same address into ”queue.list” several times.

Recommendation Consider adding an explicit ”require” statement to check that ”amount” is not zero. Alternatively, just do nothing in case of a zero ”amount”.

```
22 queue.cache[deposit.owner] += deposit.amount;
```

CVF-25. INFO

- **Category** Suboptimal
- **Source** DepositQueueLib.sol

Description This function overwrites the whole list even if only a few elements were removed. This is inefficient.

Recommendation Consider removing elements in place, and filling the gap with element from the end of the list.

```
26 function remove(
```

CVF-30. FIXED

- **Category** Flaw
- **Source** ETHAdapter.sol

Description It is not checked that ”vault.asset()” and pool’s token #1 are the same. This allows a malicious user to take arbitrary tokens from the adapter.

Recommendation Consider adding an explicit check to ensure that ”vault.asset()” is the same as the pool’s token #1.

```
35 IERC20(vault.asset()).safeApprove(address(vault), assets);
```

```
97 IERC20 asset = IERC20(vault.asset());
```

CVF-33. INFO

- **Category** Unclear behavior
- **Source** ETHAdapter.sol

Description Here, all the balance is transferred to the receiver, which could include some ether sent to this contract by mistake.

Recommendation Consider sending the exact output amount returned by the ”exchange” call.

Client Comment *Since we didn’t code the ”rescue method” we’re assuming no ether will be sent by mistake.*

```
103 payable(receiver).sendValue(address(this).balance);
```

9 Minor Issues

CVF-1. INFO

- **Category** Procedural
- **Source** STETHVault.sol

Recommendation Specifying a particular compiler version makes it harder to migrate to newer versions. Consider specifying as "^0.8.0". Also relevant for the next files: BaseVault.sol, Migration.sol, DepositQueueLib.sol, ETHAdapter.sol, Capped.sol, FixedPointMath.sol, CastUint.sol, ConfigurationManager.sol, IVaultMetadata.sol, IVault.sol, IERC4626.sol, ICurvePool.sol, IConfigurationManager.sol.

Client Comment *We always received this topic as a floating pragma from other auditors. We will keep it fixed for now.*

```
3 pragma solidity 0.8.9;
```

CVF-2. FIXED

- **Category** Documentation
- **Source** STETHVault.sol

Description The number format of this value is unclear.

Recommendation Consider documenting.

```
21 uint256 public constant investorRatio = 5000;
```

CVF-3. INFO

- **Category** Unclear behavior
- **Source** STETHVault.sol

Description In ERC20, the "decimals" property is used by UI to render token amounts in a human-friendly way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

Client Comment *Acknowleged.*

```
39 sharePriceDecimals = asset.decimals();
```

CVF-4. INFO

- **Category** Unclear behavior
- **Source** STETHVault.sol

Description Here, a price precision depends of token decimals. This could cause problems for tokens with few decimals, such as EURS.

Client Comment *This contract will only be used with the pair STETH<>ETH. We will take this in consideration if in the future we use this contract to handle other assets than those.*

```
39 sharePriceDecimals = asset.decimals();
```

CVF-5. INFO

- **Category** Suboptimal
- **Source** STETHVault.sol

Description Here, the name and symbol are calculated every time. This is gas consuming.

Recommendation Consider calculating once in the constructor and storing in variables. Immutable variables would be most efficient, but Solidity doesn't support immutable variable of dynamic types such as string. However, it is still possible to pack a small string into a 32-bytes word and store such word in an immutable variable: https://medium.com/coinmonks/immutable-string-variables-2a35fc385a41

```
46 return string(abi.encodePacked(asset.symbol(), " Volatility Vault"))
   ↪ ;
```

```
53 return string(abi.encodePacked(asset.symbol(), "vv"));
```

CVF-6. INFO

- **Category** Suboptimal
- **Source** STETHVault.sol

Description Calculating "10**sharePriceDecimals" each time is suboptimal.

Recommendation Consider calculating once and storing in a immutable variable.

Client Comment *Gas optimizations will be prioritized in a future version.*

```
65 uint256 sharePrice = lastSharePrice.denominator == 0 ? 0 :
   ↪ lastSharePrice.mulDivDown(10**sharePriceDecimals);
```

```
89     endSharePrice = (totalAssets()).mulDivDown(10**
          ↪ sharePriceDecimals, supply);
```

CVF-7. INFO

- **Category** Suboptimal
- **Source** STETHVault.sol

Description Here assets are first transferred from then investor and then transferred back to him. This is suboptimal.

Recommendation Consider calcualting the net amount to be transferred and then transfer once.

Client Comment *We are optmizing for readbility at this moment. We will focus in gas optimizations in the future.*

```
81 asset.safeTransferFrom(investor, address(this), investmentYield);
```

```
85 asset.safeTransfer(investor, investmentAmount);
```

CVF-9. INFO

- **Category** Readability
- **Source** BaseVault.sol

Recommendation This value could be rendered as ”0.1e4”.

Client Comment *Mittigated by adding documentation.*

```
34 uint256 public constant MAX_WITHDRAW_FEE = 1000;
```

CVF-12. INFO

- **Category** Unclear behavior
- **Source** BaseVault.sol

Recommendation This function doesn’t make much sense and the sender may approve before the call. No need for ”permit”.

Client Comment *The use of permit is just a matter of better UX in the frontend. So, the caller dont need to pay two transactions (approve + deposit).*

```
74 function depositWithPermit(
```

```
99 function mintWithPermit(
```

CVF-14. INFO

- **Category** Suboptimal
- **Source** BaseVault.sol

Description Calculations of both, ”covnertToAssets(shares)” and ”committedAssets” end with division by the total supply.

Recommendation Consider calculating the sum of numerators first and then divide once.

```
243 return convertToAssets(shares) + idleAssetsOf(owner) +
      ↪ committedAssets;
```

CVF-15. INFO

- **Category** Suboptimal
- **Source** BaseVault.sol

Description This loop doesn’t scale.

Recommendation Consider passing the index of a sender’s deposit as an argument to make this loop unnecessary.

```
299 for (uint256 i = 0; i < depositQueue.size(); i++) {
```

CVF-16. FIXED

- **Category** Overflow/Underflow
- **Source** BaseVault.sol

Description Phantom overflow is possible here.

Recommendation Consider using the ”muldiv” function.

```
344 return (assets * withdrawFeeRatio()) / DENOMINATOR;
```

CVF-17. FIXED

- **Category** Unclear behavior
- **Source** BaseVault.sol

Description This should be done only when ”fee” is not zero.

```
386 emit FeeCollected(fee);
```

```
389 asset.safeTransfer(controller(), fee);
```

CVF-18. FIXED

- **Category** Procedural
- **Source** BaseVault.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
395 function _beforeWithdraw(uint256 shares, uint256 assets) internal
      ↪ virtual {}
```

```
398 function _afterRoundStart(uint256 assets) internal virtual {}
```

```
401 function _afterRoundEnd() internal virtual {}
```

CVF-19. FIXED

- **Category** Unclear behavior
- **Source** Migration.sol

Description These functions should return the shares amount minted.

```
21 function migrate() external {
```

```
31 function migrateWithPermit(
```

CVF-20. INFO

- **Category** Suboptimal
- **Source** Migration.sol

Description Here the whole balance is deposited, which could include some tokens sent to this contract by mistake.

Recommendation Consider depositing exactly the redeemed amount.

Client Comment *Acknowleged*.

```
26 uint256 balance = asset.balanceOf(address(this));
```

```
42 uint256 balance = asset.balanceOf(address(this));
```

CVF-22. INFO

- **Category** Suboptimal
- **Source** Migration.sol

Description This function doesn't make much sens as the caller may just approve shares before the call. No need for "permit".

```
31 function migrateWithPermit(
```

CVF-27. INFO

- **Category** Readability
- **Source** DepositQueueLib.sol

Description Using an argument as a mutable variable is a bad practice that makes code harder to read.

Recommendation Consider using a separate variable instead of "startIndex".

```
48 startIndex++;
```

CVF-31. INFO

- **Category** Unclear behavior
- **Source** ETHAdapter.sol

Description These functions don't have much sense, as the message caller may just approve tokens before a call. No need to use "permit" here.

Client Comment *We want to improve UX in the frontend. The caller can avoid paying an approve transaction and use only Permit.*

```
49 function redeemWithPermit(
```

```
74 function withdrawWithPermit(
```

CVF-32. FIXED

- **Category** Documentation
- **Source** ETHAdapter.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
90 receive() external payable {}
```

CVF-34. INFO

- **Category** Suboptimal
- **Source** Capped.sol

Description This makes it impossible to restore cap when the cap is not set. This could be limiting.

Recommendation Consider not forbidding such scenario.

Client Comment *Our intention is to optionally set a cap, otherwise defaults to uncapped.*

```
40 if (availableCap() != type(uint256).max) {
```

CVF-35. INFO

- **Category** Bad naming
- **Source** FixedPointMath.sol

Description The name is completely misleading, as functions is this library are not related to fixed point math.

Recommendation Consider renaming.

```
5 library FixedPointMath {
```

CVF-36. INFO

- **Category** Readability
- **Source** FixedPointMath.sol

Description Brackets are redundant here.

Client Comment *The redundancy follow our linter rules.*

```
21 return (x * y) / denominator;
```

CVF-37. INFO

- **Category** Overflow/Underflow
- **Source** FixedPointMath.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.

Recommendation Consider using the ”mulDiv” function as described here: https://2π.com/21/muldiv/

Client Comment *Not a priority for now. We will take this into consideration into future versions.*

```
21 return (x * y) / denominator;
```

```
30 uint256 numerator = x * y;
   return numerator / denominator + (numerator % denominator > 0 ? 1 :
      ↪ 0);
```

CVF-38. FIXED

- **Category** Suboptimal
- **Source** CastUint.sol

Recommendation This function could be simplified to return address(uint160 (value));

```
9 function toAddress(uint256 value) internal pure returns (address) {
```

CVF-40. INFO

- **Category** Suboptimal
- **Source** ConfigurationManager.sol

Description These events are emitted event if nothing actually changed.

Client Comment *We want to track when it was emitted anyway.*

```
29 emit ParameterSet(target, name, value);
```

```
56 emit SetCap(target, value);
```

CVF-41. FIXED

- **Category** Suboptimal
- **Source** ConfigurationManager.sol

Description These functions superseded by the ”setParameter” and ”getParameter” functions.

Recommendation Consider just defining a constant for the cap parameter name and removing the cap-specific functions.

Client Comment *Acknowledged.*

```
53 function setCap(address target, uint256 value) external override
   ↪ onlyOwner {
```

```
64 function getCap(address target) external view override returns (
   ↪ uint256) {
```

CVF-42. FIXED

- **Category** Bad datatype
- **Source** ConfigurationManager.sol

Recommendation The value ”CAP” should be a named constant.

```
55 setParameter(target, "CAP", value);
```

```
65 return this.getParameter(target, "CAP");
```

CVF-43. FIXED

- **Category** Procedural
- **Source** IVaultMetadata.sol

Recommendation This function should be a part of the ”IERC4626” interface, as it is defined in the ERC-4626 standard.

```
9 function asset() external view returns (address);
```

CVF-44. INFO

- **Category** Bad naming
- **Source** IVault.sol

Recommendation Events are usually named via nouns, such as ”CollectedFee”, ”RoundStart”, ”RoundEnd”, etc.

Client Comment *Not a priority for now. We will take this into consideration into future versions.*

```
15 event FeeCollected(uint256 fee);
   event StartRound(uint256 indexed roundId, uint256
      ↪ amountAddedToStrategy);
   event EndRound(uint256 indexed roundId);
   event DepositProcessed(address indexed owner, uint256 indexed
      ↪ roundId, uint256 assets, uint256 shares);
   event DepositRefunded(address indexed owner, uint256 indexed roundId
      ↪ , uint256 assets);
```

CVF-45. FIXED

- **Category** Documentation
- **Source** IVault.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

```
24 function withdrawFeeRatio() external view returns (uint256);
```

CVF-46. FIXED

- **Category** Unclear behavior
- **Source** IVault.sol

Description This function should return the ID of the started round.

```
56 function startRound() external;
```

CVF-47. FIXED

- **Category** Unclear behavior
- **Source** IVault.sol

Description This function should return the amount of assets refunded.

```
67 function refund() external;
```

CVF-49. FIXED

- **Category** Flaw
- **Source** IERC4626.sol

Description This interface misses the ”asset” function defined by the ERC-4626 standard.

```
7 interface IERC4626 is IERC20 {
```

CVF-50. INFO

- **Category** Bad naming
- **Source** IConfigurationManager.sol

Recommendation Events are usually named via nouns, such as ”Cap” and ”Parameter”.

Client Comment *Not a priority for now. We will take this into consideration into future versions.*

```
6 event SetCap(address indexed target, uint256 value);
  event ParameterSet(address indexed target, bytes32 indexed name,
     ↪ uint256 value);
```

CVF-51. INFO

- **Category** Unclear behavior
- **Source** IConfigurationManager.sol

Recommendation This error should include the problematic target as a parameter.

Client Comment *The only invalid address is address(0).*

```
9 error ConfigurationManager__InvalidCapTarget();
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting