CredShields

Smart Contract Audit

July 7th, 2022 • CONFIDENTIAL

Description

This document details the process and result of the Pods Finance smart contract audit performed by CredShields Technologies PTE. LTD. on behalf of Pods Finance between June 20th, 2022, and July 4th, 2022. And a retest was performed on 5th Aug 2022.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

Pods Finance

Table of Contents

1. Executive Summary

Pods Finance engaged CredShields to perform a smart contract audit from June 20th, 2022, to July 4th, 2022. During this timeframe, six (6) vulnerabilities were identified. **A retest was performed on 5th Aug 2022 and all the bugs have been resolved.**

During the audit, zero (0) vulnerability was found that had a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Pods Finance" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
Pods Finance	0	0	0	3	1	2	**6**
	0	**0**	**0**	**3**	**1**	**2**	**6**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Pod's scope during the testing window while abiding by the policies set forth by "Pods Finance" team.

State of Security

Maintaining a healthy security posture requires constant review and refinement of existing security processes. Running a CredShields continuous audit allows “Pods Finance” internal security team and development team to not only uncover specific vulnerabilities but gain a better understanding of the current security threat landscape.

We recommend running regular security assessments to identify any vulnerabilities introduced after Pods Finance introduces new features or refactors the code.

Reviewing the remaining resolved reports for a root cause analysis can further educate “Pods Finance” internal development and security teams and allow manual or automated procedures to be put in place to eliminate entire classes of vulnerabilities in the future. This proactive approach helps contribute to future-proofing the security posture of Pods Finance’s assets.

2. Methodology

Pods Finance engaged CredShields to perform a smart contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation phase

CredShields team read all the provided documents and comments in the smart-contract code to understand the contract's features and functionalities. The team reviewed all the functions and prepared a mind map to review for possible security vulnerabilities in the order of the function with more critical and business-sensitive functionalities for the refactored code.
The team deployed a self-hosted version of the smart contract to verify the assumptions and validation of the vulnerabilities during the audit phase.

A testing window from June 20th, 2022, to July 4th, 2022, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed-upon:

IN SCOPE ASSETS
https://github.com/pods-finance/yield-contracts/tree/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts ● **BaseVault.sol** ● **STETHVault.sol** ● **Capped.sol** ● **CastUint.sol** ● **DepositQueueLib.sol** ● **FixedPointMath.sol** ● **TransferUtils.sol** ● **ConfigurationManager.sol**

Table: List of Files in Scope

2.1.2 Documentation

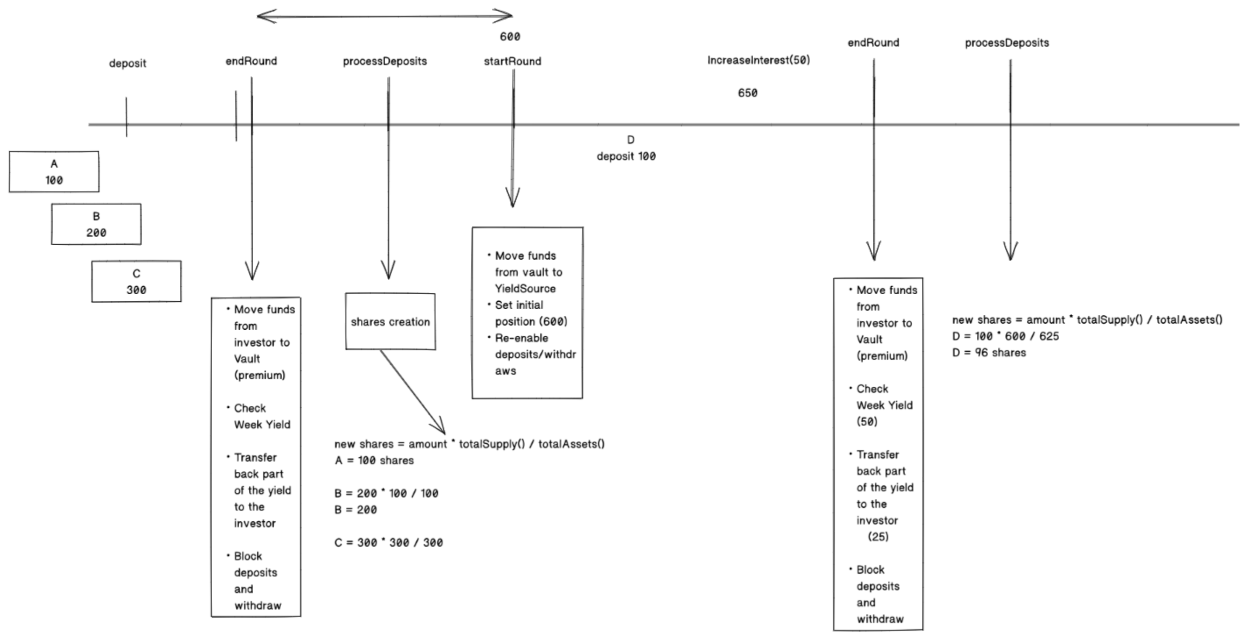
The following documentations were available to the audit team.

https://github.com/pods-finance/yield-contracts

https://app.tryeraser.com/integration/gather/MsqynSElTMiQvFvntable9

The audit team also mapped all the user flows and privileges and mapped functionalities of the smart contract. The control flow graph can be found below

Control Flow Graph

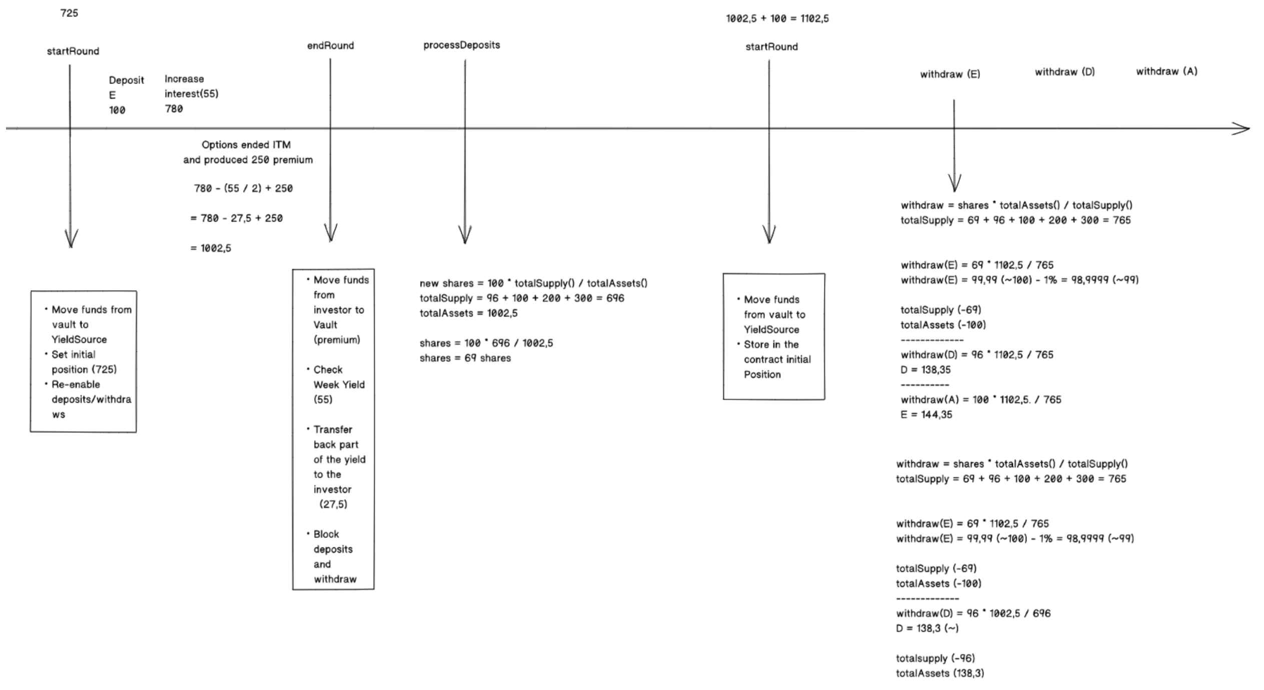

600
deposit
endRound
processDeposits
startRound
IncreaseInterest(50)
650
endRound
processDeposits
D
deposit 100
A
100
B
200
C
300
• Move funds from investor to Vault (premium)
• Check Week Yield
• Transfer back part of the yield to the investor
• Block deposits and withdraw
shares creation
• Move funds from vault to YieldSource
• Set initial position (600)
• Re-enable deposits/withdraws
• Move funds from investor to Vault (premium)
• Check Week Yield (50)
• Transfer back part of the yield to the investor (25)
• Block deposits and withdraw

new shares = amount * totalSupply() / totalAssets()
A = 100 shares

B = 200 * 100 / 100
B = 200

C = 300 * 300 / 300

new shares = amount * totalSupply() / totalAssets()
D = 100 * 600 / 625
D = 96 shares


725
startRound
Deposit
E
100
Increase
interest(55)
780
endRound
processDeposits
1002,5 + 100 = 1102,5
startRound
withdraw (E)
withdraw (D)
withdraw (A)
• Move funds from vault to YieldSource
• Set initial position (725)
• Re-enable deposits/withdraws
• Move funds from investor to Vault (premium)
• Check Week Yield (55)
• Transfer back part of the yield to the investor (27,5)
• Block deposits and withdraw
• Move funds from vault to YieldSource
• Store in the contract initial Position

Options ended ITM
and produced 250 premium

780 - (55 / 2) + 250

= 780 - 27,5 + 250

= 1002,5

new shares = 100 * totalSupply() / totalAssets()
totalSupply = 96 + 100 + 200 + 300 = 696
totalAssets = 1002,5

shares = 100 * 696 / 1002,5
shares = 69 shares

withdraw = shares * totalAssets() / totalSupply()
totalSupply = 69 + 96 + 100 + 200 + 300 = 765

withdraw(E) = 69 * 1102,5 / 765
withdraw(E) = 99,99 (~100) - 1% = 98,9999 (~99)

totalSupply (-69)
totalAssets (-100)
----
withdraw(D) = 96 * 1102,5 / 765
D = 138,35
----
withdraw(A) = 100 * 1102,5. / 765
E = 144,35

withdraw = shares * totalAssets() / totalSupply()
totalSupply = 69 + 96 + 100 + 200 + 300 = 765

withdraw(E) = 69 * 1102,5 / 765
withdraw(E) = 99,99 (~100) - 1% = 98,9999 (~99)

totalSupply (-69)
totalAssets (-100)
----
withdraw(D) = 96 * 1002,5 / 696
D = 138,3 (~)

totalsupply (-96)
totalAssets (138,3)

2.1.3 Audit Goals

CredShields' methodology uses individual tools and methods; however, tools are just used for aids. The majority of the audit methods involve manually reviewing the smart contract source code. The team followed the standards of the SWC registry for testing along with an extended self-developed checklist based on industry standards, but it was not limited to it. The team focused heavily on understanding the core concept behind all the functionalities along with preparing test and edge cases. Understanding the business logic and how it could have been exploited.

The audit's focus was to verify that the smart contract system is secure, resilient, and working according to its specifications. Breaking the audit activities into the following three categories:

- **Security** - Identifying security-related issues within each contract and the system of contracts.
- **Sound Architecture** - Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
- **Code Correctness and Quality** - A full review of the contract source code. The primary areas of focus include:
    - Correctness
    - Readability
    - Sections of code with high complexity
    - Improving scalability
    - Quantity and quality of test coverage

2.2 Retesting phase

Pods Finance is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

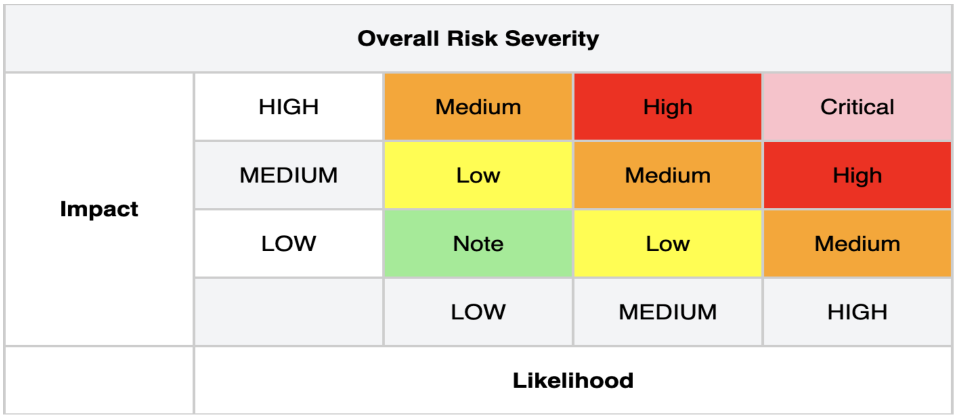
2.3 Vulnerability classification and severity

Discovering vulnerabilities is important, but estimating the associated risk to the business is just as important.

To adhere to industry guidelines, CredShields follows OWASP's Risk Rating Methodology. This is calculated using two factors - **Likelihood** and **Impact**. Each of these parameters can take three values - **Low**, **Medium**, and **High**.

These depend upon multiple factors such as Threat agents, Vulnerability factors (Ease of discovery and exploitation, etc.), and Technical and Business Impacts. The likelihood and the impact estimate are put together to calculate the overall severity of the risk.

CredShields also define an **Informational** severity level for vulnerabilities that do not align with any of the severity categories and usually have the lowest risk involved.



Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Overall, the categories can be defined as described below -

1. **Informational**

We believe in the importance of technical excellence and pay a great deal of attention to its details. Our coding guidelines, practices, and standards help ensure that our software is stable and reliable.

Informational vulnerabilities should not be a cause for alarm but rather a chance to improve the quality of the codebase by emphasizing readability and good practices.

They do not represent a direct risk to the Contract but rather suggest improvements and the best practices that can not be categorized under any of the other severity categories.

Code maintainers should use their own judgment as to whether to address such issues.

2. Low

Vulnerabilities in this category represent a low risk to the Smart Contract and the organization. The risk is either relatively small and could not be exploited on a recurring basis, or a risk that the client indicates is not important or significant, given the client's business circumstances.

3. Medium

Medium severity issues are those that are usually introduced due to weak or erroneous logic in the code.

These issues may lead to exfiltration or modification of some of the private information belonging to the end-user, and exploitation would be detrimental to the client’s reputation under certain unexpected circumstances or conditions. These conditions are outside the control of the adversary.

These issues should eventually be fixed under a certain timeframe and remediation cycle.

4. **High**

   High severity vulnerabilities represent a greater risk to the Smart Contract and the organization. These vulnerabilities may lead to a limited loss of funds for some of the end-users.

   They may or may not require external conditions to be met, or these conditions may be manipulated by the attacker, but the complexity of exploitation will be higher.

   These vulnerabilities, when exploited, will impact the client’s reputation negatively.

   They should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities. These issues do not require any external conditions to be met.

   The majority of vulnerabilities of this type involve a loss of funds and Ether from the Smart Contracts and/or from their end-users.

   The issue puts the vast majority of, or large numbers of, users' sensitive information at risk of modification or compromise.

   The client's reputation will suffer a severe blow, or there will be serious financial repercussions.

Considering the risk and volatility of smart contracts and how they use gas as a method of payment to deploy the contracts and interact with them, gas optimization becomes a major point of concern. To address this, CredShields also introduces another severity category called “**Gas Optimization**” or “**Gas**”. This category deals with code optimization techniques and refactoring due to which Gas can be conserved.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- **Shashank, Co-founder CredShields**
    - shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have around the engagement or this document.

3. Findings

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, six (6) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Multiple Zero Address Validations Missing	Low	Missing Input Validation
Dead/Unreachable Code	Informational	Missing best practices
Missing Constant Attribute in Variables	Gas	Gas Optimization
Functions should be declared External	Gas	Gas Optimization
Floating Pragma	Low	Floating Pragma
Missing Reentrancy Protections	Low	Reentrancy

Table: Findings in Smart Contracts

3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after v0.5.X (Currently using solidity v >= 0.8.6)
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X.
SWC-102	Outdated Compiler Version	Not Vulnerable	Version 0.8.6 is used
SWC-103	Floating Pragma	Vulnerable	Contract uses floating pragma
SWC-104	Unchecked Call Return Value	Not Vulnerable	call() is used but return values are checked
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	selfdestruct() is not used anywhere
SWC-107	Reentrancy	Vulnerable	Notable functions such as deposit, mint, redeem related function were found to be missing reentrancy guard. However, they had no exploitation scenarios for now.

SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, v0.5.0
SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like block.blockhash(), msg.gas, throw, sha3(), callcode(), suicide() are in use
SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	The logic is implemented in such a way that the external calls if they fail, they fail gracefully, i.e. without having a major impact on the functionality of the contract.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	tx.origin is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	Block.timestamp, block.number and other block values are not used.
SWC-117	Signature Malleability	Not Vulnerable	Oz's ECDSA library is used which is safe
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the

			constructor keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version 0.6.0
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found
SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Oz's ECDSA library is used which is safe
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.
SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.
SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Vulnerable	Two unused/unreachable functions were found

SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() is used but does not accept user inputs to exploit hash collision
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Vulnerable	Two unused/unreachable functions were found
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status

Pods Finance is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations.
Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Multiple Zero Address Validations Missing	Low	**Accepted Risk**
Dead/Unreachable Code	Informational	**Won't Fix**
Missing Constant Attribute in Variables	Gas	**Fixed [05/08/2022]**
Functions should be declared External	Gas	**Fixed [05/08/2022]**
Floating Pragma	Low	**Fixed [05/08/2022]**
Missing Reentrancy Protections	Low	**Won't Fix**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID#1 [Accepted Risk]

Multiple Zero Address Validations Missing

Vulnerability Type
Missing Input Validation

Severity
Low

Description:
Multiple Solidity contracts were found to be setting new addresses without proper validations for zero addresses.
Address type parameters should include a zero-address check otherwise, contract functionality may become inaccessible, or tokens burned forever.
Depending on the logic of the contract, this could prove fatal, and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

Affected function with address input missing zero address checks

- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L64-L75 (receiver)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L80-L90 (receiver)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L95-L120 (receiver)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L125-L148 (receiver)

- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/STETHVault.sol#L32-L39 (_investor)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/configuration/ConfigurationManager.sol#L23-L26 (target)

Impacts
If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Remediation
Add a zero address validation to all the functions where addresses are being set.

Retest:
The chance of exploitation is negligible here and hence to avoid increased gas cost this will remain as it is which is perfectly fine.

Bug ID#2 [won't fix]

Dead/Unreachable Code

Vulnerability Type
Code With No Effects - SWC-135

Severity
Informational

Description:
It is recommended to keep the production repository clean to prevent confusion and the introduction of vulnerabilities. The functions and parameters that are never used or called externally or from inside the contracts should be removed when the contract is deployed on the mainnet.

Vulnerable Code: (replace with links)

- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/libs/FixedPointMath.sol#L54-L56 (fractionRoundDown)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/libs/FixedPointMath.sol#L50-L52 (fractionRoundUp)

```
   function fractionRoundUp(Fractional memory x) internal pure returns
(uint256) {
       return x.numerator.mulDivUp(1, x.denominator);
   }

   function fractionRoundDown(Fractional memory x) internal pure returns
(uint256) {
       return x.numerator.mulDivDown(1, x.denominator);
   }
```

PoC:

1. Go to the contract “`FixedPointMath.sol`” and note the functions “fractionRoundUp” and “fractionRoundDown” areinternal functions.
2. This function is not called throughout the code in any of the contracts and therefore should be removed.

Impacts:
This does not impact the security aspect of the Smart contract but prevents confusion when the code is sent to other developers or auditors to understand and implement. This reduces the overall size of the contracts and also helps in saving gas.

Remediation:
Delete the internal functions “fractionRoundUp” and “fractionRoundDown”if they are not supposed to be used.

Retest:
The dead code mentioned was a part of ongoing development and hence the dead code would be used later on.

Bug ID#3 [Fixed]

Missing Constant Attribute in Variables

Vulnerability Type
Gas Optimization

Severity
Gas

Description:
State variables can be declared as constant or immutable. In both cases, the variables cannot be modified after the contract has been constructed. For constant variables, the value has to be fixed at compile time.
The compiler does not reserve a storage slot for these variables, and every occurrence is replaced by the respective value.
Compared to regular state variables, the gas costs of constant and immutable variables are much lower since no `SLOAD` is executed to retrieve constants from storage because they're interpolated directly into the bytecode.

Vulnerable Code:

- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/STETHVault.sol#L20 (investorRatio)

```
uint256 public investorRatio = 5000;
```

PoC

1. Go to the contract “`STETHVault.sol`” and note the large number literal at line 20.

Impacts:
Gas usage is increased if the variables that should be constants are not set as constants.

Remediation:

A “`constant`” attribute should be added to the parameters that never change to save the gas.

Retest:

This is fixed as the variable has been marked as constant at https://github.com/pods-finance/yield-contracts/blob/main/contracts/vaults/STETHVault.sol#L21

Bug ID#4 [Fixed]

Functions should be declared External

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Public functions that are never called by a contract should be declared external in order to conserve gas.
The following functions were declared as public but were not called anywhere in the contract, making the public visibility useless.

Affected Code

- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L64-L75 - (deposit)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L80-L90 - (mint)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L95-L120 - (redeem)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L125-L148 - (withdraw)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L269-L278 (startRound)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L283-L290 (endRound)
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L295-L305 (processQueuedDeposits)

Impacts

Smart Contracts are required to have effective Gas usage as they cost real money and each function should be monitored for the amount of gas it costs to make it gas efficient.
“**public**” functions cost more Gas than “**external**” functions.

Remediation
Use the “**external**” state visibility for functions that are never called from inside the contract.

Retest:
All the required public functions have been marked as external
https://github.com/pods-finance/yield-contracts/commit/835897214da8b457ebda4e1fe23a1bd6079d3e73

Bug ID#5 [Fixed]

Floating Pragma

Vulnerability Type
Floating Pragma (SWC-103)

Severity
Low

Description
Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.
The contracts found in the repository allowed floating or unlocked pragma to be used, i.e., **>=0.8.6**.
This allows the contracts to be compiled with all the solidity compiler versions above **0.8.6**.
The following contracts were found to be affected -

Affected Files

- ConfigurationManager.sol
- CastUint.sol
- DepositQueueLib.sol
- FixedPointMath.sol
- TransferUtils.sol
- Capped.sol
- BaseVault.sol
- STETHVault.sol

Impacts
If the smart contract gets compiled and deployed with an older version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions. Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.
The likelihood of exploitation is really low therefore, this is only informational.

Remediation
Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere.
Reference: https://swcregistry.io/docs/SWC-103

Retest:
All the contracts pragma has been set to strict pragma with value 0.8.9

Bug ID#6 [Won't Fix]

Missing Reentrancy Protections

Vulnerability Type
Reentrancy

Severity
Low

Description
In a Reentrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.
The smart contracts were missing reentrancy protection on the following functions making external calls.

Affected Code

- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/STETHVault.sol#L54-L82 - In **_afterRoundEnd()** function with external calls **asset.safeTransfer** and **asset.safeTransferFrom**. This is emitting events after the external call - **EndRoundData** and **SharePrice.**
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L64-L75 - In **deposit()** function with external call **asset.safeTransferFrom**. This is emitting an event after the external call - **Deposit.**
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L80-L90 - In **mint()** function with external call **asset.safeTransferFrom**. This is emitting an event after the external call - **Deposit.**
- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L95-L120 - In **redeem()** function with external call **asset.safeTransfer**. This is emitting events after the external call - **FeeCollected** and **Withdraw.**

- https://github.com/pods-finance/yield-contracts/blob/03e0f39ca6cd0449d8f597faaf9526f808ca2606/contracts/vaults/BaseVault.sol#L125-L148 - In **withdraw()** function with external call **asset.safeTransfer**. This is emitting events after the external call - **FeeCollected** and **Withdraw.**

Impacts

Lacking reentrancy protection could allow threat actors to abuse the functions and reenter the contract.

However, it should be noted that right now there's little to no impact due to all the state changes being done before the external calls. This is an event-based reentrancy so events can be flooded with spam data.

Remediation

Add a Reentrancy guard to all the functions making external calls.

Retest:

The current scenario is not exploitable and hence to avoid extra gas costs the team would leave it as it is.

6. Appendix 1

6.1 Files in scope:

Contracts Description Table:

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
BaseVault	**Implementation**	**IVault, ERC20, ERC20Permit, Capped**		
└	<Constructor>	Public ❗️	🛑	ERC20 ERC20Permit Capped
└	decimals	Public ❗️		NO ❗️
└	deposit	Public ❗️	🛑	NO ❗️

└	mint	Public ❗️	🛑	NO ❗️
└	redeem	Public ❗️	🛑	NO ❗️
└	withdraw	Public ❗️	🛑	NO ❗️
└	totalAssets	Public ❗️		NO ❗️
└	previewDeposit	Public ❗️		NO ❗️
└	previewMint	Public ❗️		NO ❗️
└	previewWithdraw	Public ❗️		NO ❗️
└	previewRedeem	Public ❗️		NO ❗️
└	convertToShares	Public ❗️		NO ❗️
└	convertToAssets	Public ❗️		NO ❗️
└	maxDeposit	Public ❗️		NO ❗️
└	maxMint	Public ❗️		NO ❗️
└	maxWithdraw	Public ❗️		NO ❗️

L	maxRedeem	Public ❗		NO ❗
L	withdrawFeeRatio	Public ❗		NO ❗
L	idleAssetsOf	Public ❗		NO ❗
L	assetsOf	Public ❗		NO ❗
L	totalIdleAssets	Public ❗		NO ❗
L	depositQueueSize	Public ❗		NO ❗
L	controller	Public ❗		NO ❗
L	startRound	Public ❗	🛑	onlyController
L	endRound	Public ❗	🛑	onlyController
L	processQueuedDeposits	Public ❗	🛑	NO ❗
L	_processDeposit	Internal 🔒	🛑	
L	_getFee	Internal 🔒		
L	_beforeWithdraw	Internal 🔒	🛑	

└	_afterRoundStart	Internal 🔒	🛑	
└	_afterRoundEnd	Internal 🔒	🛑	
STETHVault	**Implementation**	**BaseVault**		
└	<Constructor>	Public ❗️	🛑	BaseVault
└	name	Public ❗️		NO ❗️
└	symbol	Public ❗️		NO ❗️
└	deposit	Public ❗️	🛑	NO ❗️
└	mint	Public ❗️	🛑	NO ❗️
└	_afterRoundStart	Internal 🔒	🛑	
└	_afterRoundEnd	Internal 🔒	🛑	
└	_beforeWithdraw	Internal 🔒	🛑	
└	totalAssets	Public ❗️		NO ❗️

└	_tryTransferSTETH	Internal 🔒	🛑	
IERC20Metadata	**Interface**	**IERC20**		
└	name	External ❗️		NO❗️
└	symbol	External ❗️		NO❗️
└	decimals	External ❗️		NO❗️
IERC20	**Interface**			
└	totalSupply	External ❗️		NO❗️
└	balanceOf	External ❗️		NO❗️
└	transfer	External ❗️	🛑	NO❗️
└	allowance	External ❗️		NO❗️

└	approve	External ❗️	🛑	NO ❗️
└	transferFrom	External ❗️	🛑	NO ❗️
ERC20Permit	**Implementation**	**ERC20, IERC20Permit, EIP712**		
└	<Constructor>	Public ❗️	🛑	EIP712
└	permit	Public ❗️	🛑	NO ❗️
└	nonces	Public ❗️		NO ❗️
└	DOMAIN_SEPARATOR	External ❗️		NO ❗️
└	_useNonce	Internal 🔒	🛑	
IERC20Permit	**Interface**			
└	permit	External ❗️	🛑	NO ❗️

└	nonces	External ❗		NO ❗
└	DOMAIN_SEPARATOR	External ❗		NO ❗
ERC20	**Implementation**	**Context, IERC20, IERC20Metadata**		
└	<Constructor>	Public ❗	🛑	NO ❗
└	name	Public ❗		NO ❗
└	symbol	Public ❗		NO ❗
└	decimals	Public ❗		NO ❗
└	totalSupply	Public ❗		NO ❗
└	balanceOf	Public ❗		NO ❗
└	transfer	Public ❗	🛑	NO ❗
└	allowance	Public ❗		NO ❗

└	approve	Public ❗️	🛑	NO❗️
└	transferFrom	Public ❗️	🛑	NO❗️
└	increaseAllowance	Public ❗️	🛑	NO❗️
└	decreaseAllowance	Public ❗️	🛑	NO❗️
└	_transfer	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_spendAllowance	Internal 🔒	🛑	
└	_beforeTokenTransfer	Internal 🔒	🛑	
└	_afterTokenTransfer	Internal 🔒	🛑	
Context	**Implementation**			

└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
EIP712	**Implementation**			
└	<Constructor>	Public ❗️	🛑	NO❗️
└	_domainSeparatorV4	Internal 🔒		
└	_buildDomainSeparator	Private 🔐		
└	_hashTypedDataV4	Internal 🔒		
ECDSA	**Library**			
└	_throwError	Private 🔐		
└	tryRecover	Internal 🔒		
└	recover	Internal 🔒		

└	tryRecover	Internal 🔒		
└	recover	Internal 🔒		
└	tryRecover	Internal 🔒		
└	recover	Internal 🔒		
└	toEthSignedMessageHash	Internal 🔒		
└	toEthSignedMessageHash	Internal 🔒		
└	toTypedDataHash	Internal 🔒		
Strings	**Library**			
└	toString	Internal 🔒		
└	toHexString	Internal 🔒		
└	toHexString	Internal 🔒		

Counters	**Library**			
└	current	Internal 🔒		
└	increment	Internal 🔒	🛑	
└	decrement	Internal 🔒	🛑	
└	reset	Internal 🔒	🛑	
IConfigurationManager	**Interface**			
└	setParameter	External ❗	🛑	NO ❗
└	getParameter	External ❗		NO ❗
└	getGlobalParameter	External ❗		NO ❗
└	setCap	External ❗	🛑	NO ❗
└	getCap	External ❗		NO ❗

IVault	**Interface**	**IERC4626**		
└	withdrawFeeRatio	External ❗		NO ❗
└	controller	External ❗		NO ❗
└	idleAssetsOf	External ❗		NO ❗
└	assetsOf	External ❗		NO ❗
└	totalIdleAssets	External ❗		NO ❗
└	depositQueueSize	External ❗		NO ❗
└	startRound	External ❗	🛑	NO ❗
└	endRound	External ❗	🛑	NO ❗
└	processQueuedDeposits	External ❗	🛑	NO ❗
IERC4626	**Interface**	**IERC20**		

└	totalAssets	External ❗		NO ❗
└	deposit	External ❗	🛑	NO ❗
└	mint	External ❗	🛑	NO ❗
└	redeem	External ❗	🛑	NO ❗
└	withdraw	External ❗	🛑	NO ❗
└	previewDeposit	External ❗		NO ❗
└	previewMint	External ❗		NO ❗
└	previewWithdraw	External ❗		NO ❗
└	previewRedeem	External ❗		NO ❗
└	convertToShares	External ❗		NO ❗
└	convertToAssets	External ❗		NO ❗
└	maxDeposit	External ❗		NO ❗
└	maxMint	External ❗		NO ❗

└	maxWithdraw	External ❗		NO❗
└	maxRedeem	External ❗		NO❗
TransferUtils	**Library**			
└	safeTransfer	Internal 🔒	🛑	
└	safeTransferFrom	Internal 🔒	🛑	
└	_callOptionalReturn	Private 🔐	🛑	
FixedPointMath	**Library**			
└	mulDivDown	Internal 🔒		
└	mulDivUp	Internal 🔒		
└	mulDivUp	Internal 🔒		

└	mulDivDown	Internal 🔒		
└	mulDivUp	Internal 🔒		
└	mulDivDown	Internal 🔒		
└	fractionRoundUp	Internal 🔒		
└	fractionRoundDown	Internal 🔒		
└	min	Internal 🔒		
DepositQueueLib	**Library**			
└	push	Internal 🔒	🛑	
└	remove	Internal 🔒	🛑	
└	get	Internal 🔒		
└	balanceOf	Internal 🔒		

└	size	Internal 🔒		
CastUint	**Library**			
└	toAddress	Internal 🔒		
Capped	**Implementation**			
└	<Constructor>	Public ❗️	🛑	NO❗️
└	availableCap	Public ❗️		NO❗️
└	_spendCap	Internal 🔒	🛑	
└	_restoreCap	Internal 🔒	🛑	
ConfigurationManager	**Implementation**	**IConfigurationManager, Ownable**		

L	setParameter	Public ❗	🛑	onlyOwner
L	getParameter	External ❗		NO ❗
L	getGlobalParameter	External ❗		NO ❗
L	setCap	External ❗	🛑	onlyOwner
L	getCap	External ❗		NO ❗
Ownable	**Implementation**	**Context**		
L	<Constructor>	Public ❗	🛑	NO ❗
L	owner	Public ❗		NO ❗
L	renounceOwnership	Public ❗	🛑	onlyOwner
L	transferOwnership	Public ❗	🛑	onlyOwner
L	_transferOwnership	Internal 🔒	🛑	

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

6.2 Disclosure:

The Reports neither endorse nor condemn any specific project or team, nor do they guarantee the security of any specific project. The contents of this report do not, and should not be interpreted as having any bearing on, the economics of tokens, token sales, or any other goods, services, or assets.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. There is no warranty or representation made by this report to any Third Party in regards to the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business.

In no way should a third party use these reports to make any decisions about buying or selling a token, product, service, or any other asset. It should be noted that this report is not investment advice, is not intended to be relied on as investment advice, and has no endorsement of this project or team. It does not serve as a guarantee as to the project's absolute security.

CredShields Audit team owes no duty to any third party by virtue of publishing these Reports.