Report

v. 1.0

Customer

Stakewise

Smart Contract Audit

v3-core

2nd September 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	02.09.25	A. Zveryanskaya	Initial Draft
0.2	02.09.25	A. Zveryanskaya	Minor revision
1.0	02.09.25	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) 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.

Stakewise is a liquid staking protocol for DeFi natives, solo stakers, and institutions on Ethereum and Gnosis Chain.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

curators/		
BalancedCurator.sol	CuratorsRegistry.sol	
base/		
ERC20Upgradeable.sol	Multicall.sol	
keeper/		
Keeper.sol	KeeperOracles.sol	KeeperRewards.sol
KeeperValidators.sol		
libraries/		
EIP712Utils.sol	Errors.sol	ExitQueue.sol
OsTokenUtils.sol	ValidatorUtils.sol	
misc/		
EthRewardSplitter.sol	GnoRewardSplitter.sol	RewardSplitter.sol
RewardSplitterFactory.sol		
tokens/		
EthOsTokenVaultEscrow.sol	GnoOsTokenVault Escrow.sol	OsToken.sol
OsTokenConfig.sol	OsTokenFlashLoans.sol	OsTokenRedeemer.sol
OsTokenVaultController.sol	OsTokenVaultEscrow.sol	PriceFeed.sol
validators/		
ConsolidationsChecker.sol	DepositDataRegistry.sol	

interfaces/		
IBalancer RateProvider.sol	IBalancerVault.sol	IChainlinkAggregator.sol
IChainlinkV3 Aggregator.sol	IConsolidations Checker.sol	ICuratorsRegistry.sol
IDepositDataRegistry.sol	IEthBlocklist Erc20Vault.sol	IEthBlocklistVault.sol
IEthErc20Vault.sol	IEthFoxVault.sol	IEthGenesisVault.sol
IEthMetaVault.sol	IEthMetaVaultFactory.sol	IEthPoolEscrow.sol
IEthPrivErc20Vault.sol	IEthPrivVault.sol	IEthValidatorsRegistry.sol
IEthVault.sol	IEthVaultFactory.sol	IGnoBlocklist Erc20Vault.sol
IGnoBlocklistVault.sol	IGnoErc20Vault.sol	IGnoGenesisVault.sol
IGnoMetaVault Factory.sol	IGnoPoolEscrow.sol	IGnoPrivErc20Vault.sol
IGnoPrivVault.sol	IGnoTokensConverter.sol	IGnoValidators Registry.sol
IGnoVault.sol	IGnoVaultFactory.sol	IKeeper.sol
IKeeperOracles.sol	IKeeperRewards.sol	IKeeperValidators.sol
IMerkleDistributor.sol	IMulticall.sol	IOsToken.sol
IOsTokenConfig.sol	IOsTokenFlashLoan Recipient.sol	IOsTokenFlashLoans.sol
IOsTokenRedeemer.sol	IOsTokenVault Controller.sol	IOsTokenVaultEscrow.sol
IOsTokenVault EscrowAuth.sol	IOwnMevEscrow.sol	IRewardEthToken.sol
IRewardGnoToken.sol	IRewardSplitter.sol	IRewardSplitter Factory.sol
ISavingsXDaiAdapter.sol	ISharedMevEscrow.sol	ISubVaultsCurator.sol
ITokensConverter Factory.sol	IValidatorsChecker.sol	IValidatorsRegistry.sol
IVaultAdmin.sol	IVaultBlocklist.sol	IVaultEnterExit.sol
IVaultEthStaking.sol	IVaultFee.sol	IVaultGnoStaking.sol
IVaultMev.sol	IVaultOsToken.sol	IVaultsRegistry.sol
IVaultState.sol	IVaultSubVaults.sol	IVaultToken.sol
IVaultValidators.sol	IVaultVersion.sol	IVaultWhitelist.sol
IGnoMetaVault Factory.sol		
vaults/ethereum/custom/		
EthFoxVault.sol	EthMetaVault.sol	EthMetaVaultFactory.sol

vaults/ethereum/mev/		
OwnMevEscrow.sol	SharedMevEscrow.sol	
vaults/ethereum/		
EthBlocklistErc20Vault.sol	EthBlocklistVault.sol	EthErc20Vault.sol
EthGenesisVault.sol	EthPrivErc20Vault.sol	EthPrivVault.sol
EthVault.sol	EthVaultFactory.sol	
vaults/gnosis/custom/		
GnoMetaVault.sol	GnoMetaVaultFactory.sol	
vaults/gnosis/mev/		
GnoOwnMevEscrow.sol	GnoSharedMevEscrow.sol	
vaults/gnosis/		
GnoBlocklistErc20Vault.sol	GnoBlocklistVault.sol	GnoErc20Vault.sol
GnoGenesisVault.sol	GnoPrivErc20Vault.sol	GnoPrivVault.sol
GnoVault.sol	GnoVaultFactory.sol	
vaults/		
VaultsRegistry.sol		
vaults/modules/		
VaultAdmin.sol	VaultBlocklist.sol	VaultEnterExit.sol
VaultEthStaking.sol	VaultFee.sol	VaultGnoStaking.sol
VaultImmutables.sol	VaultMev.sol	VaultOsToken.sol
VaultState.sol	VaultSubVaults.sol	VaultToken.sol
VaultValidators.sol	VaultVersion.sol	VaultWhitelist.sol

4 Methodology

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

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code 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 as well as 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 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 if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure 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.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 1 critical, 11 major, and a few less important issues. All identified Critical and Major issues have been fixed.

Fixed 11 out of 11 issues

6 Critical Issues

CVF-1 FIXED

- **Category** Flaw
- **Source** BalancedCurator.sol

Description In case “ejectingVault” is not zero, but is not preset inside the “subVaults” array, the “amountPerVault” value will be calculated for subVaultsCount - 1 vaults, but will be distributed among subVaultsCount, which means that the total distributed amount will be greater than “assetsToDeposit”.

Recommendation Explicitly require “ejectingVault” to either be zero or to present in the “subVaults” array.

Client Comment *The ‘critical’ severity seems too high. The contract is used only with the VaultSubVaults contract, where the ejectingVault will always be present in the subVaults set.*

```
37 subVault = subVaults[i];
   if (subVault == ejectingVault) {
```

7 Major Issues

CVF-2 FIXED

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

Description It is unclear why existingAssetsPenalty <= totalExistingAssets.

Recommendation Use checked subtraction or clearly explain why underflow isn't possible.

158
```
// cannot underflow as exitingAssetsPenalty <= totalExitingAssets
_totalExitingAssets = SafeCast.toUint128(totalExitingAssets -
    ↪ exitingAssetsPenalty);
```

CVF-3 FIXED

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

Description Underflow is possible here.

Recommendation Use checked subtraction.

251
```
unchecked {
    _totalAssets -= SafeCast.toUint128(receivedAssets);
}
```

CVF-4 FIXED

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

Description The rounding error could be more than one, as the "leftTickets" value could be calculated as a sum of several rounded values.

Recommendation Refactor the code to not rely on particular maximum rounding error.

112
```
// the exit request must be fully processed (1 ticket could be a
    ↪ rounding error)
if (leftTickets > 1) revert Errors.ExitRequestNotProcessed();
```

CVF-5 FIXED

- **Category** Flaw
- **Source** RewardSplitter.sol

Description As zero “onBehalf” value has a special meaning, there should be an explicit check to guarantee, the “onBehalf” argument is not zero.

Recommendation Implement such a check.

```
161 exitPositions[positionTicket] = onBehalf;
```

```
172 if (onBehalf == address(0)) revert Errors.InvalidPosition();
```

CVF-6 FIXED

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

Description The actual rounding errors could be higher than 1 as “leftTickets” could be calculated as a sum of several rounded values.

Recommendation Refactor the code to not rely on particular maximum rounding error.

```
177 // disallow partial claims (1 ticket could be a rounding error)
    if (leftTickets > 1) revert Errors.ExitRequestNotProcessed();
```

CVF-7 FIXED

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

Description There is no check to ensure positionShares <= totalShares, so a value greater than “checkpoint.exitedAssets” could be returned.

Recommendation Revert in case positionShares > totalShares.

```
109 return (positionShares, Math.mulDiv(positionShares, checkpoint.
      ↪ exitedAssets, totalShares));
```

CVF-8 FIXED

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

Description In case the “subVaults” array contains the zero address and “ejectingVault” is zero, the “amountPerVault” will be calculated for subVaultsCount vaults, but will be distributed among subVaultsCount - n values (where n is the number of zero addresses in the “subVaults” array), which means that the total distributed amount will be lower than “assetsToDeposit”.

Recommendation Explicitly forbid zero addresses in the “subVaults” array.

```
37 subVault = subVaults[i];
   if (subVault == ejectingVault) {
```

CVF-9 FIXED

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

Description In case a non-zero “ejectingVault” address appears in the “subVaults“ array more than once, the “amountPerVault” value will be calculated for subVaultsCount - vaults, but will be distributed amount subVaultsCount - n vaults, where n is the number of times “ejectingVault” appears in the “subVaults” array, which means that the total distributed amount will be lower than “assetsToDeposit”.

Recommendation Explicitly forbid the “ejectingVault” address to appear in the “subVaults” array more than once.

Client Comment *The VaultSubVaults.sol contract is the only place where the BalancedCurator contract is used. VaultSubVaults performs all necessary checks on sub-vaults and the ejectingVault. Adding additional checks to BalancedCurator would increase gas costs without providing any extra validation beyond what is already handled by VaultSubVaults.*

```
37 subVault = subVaults[i];
   if (subVault == ejectingVault) {
```

CVF-11 FIXED

- **Category** Suboptimal
- **Source** BalancedCurator.sol

Description In order to guarantee even distribution, the “amountPerVault” value should be recalculated after finishing this loop, using the remaining “assetsToExit” and the number of remaining vaults with non-zero balances.

Recommendation Recalculate the “amountPerVault” value after the loop.

```
73 for (uint256 i = 0; i < subVaultsCount;) {
```

CVF-12 FIXED

- **Category** Suboptimal
- **Source** Multicall.sol

Description That StackExchange answer was posted before custom errors were introduced in Solidity. But event at that time, a contract was able to revert with a custom data, shorter than 68 bytes. Dropping such error data would make error investigations harder.

Recommendation Forward the error data from inner call to the caller regardless of the error data length.

```
22 // Next 5 lines from https://ethereum.stackexchange.com/a/83577
   if (result.length < 68) revert();
```

8 Moderate Issues

CVF-10 INFO

- **Category** Flaw
- **Source** BalancedCurator.sol

Description Tis loop will never end in case the sum of vault balances is lower than assetsToExit.

Recommendation Revert in case sum of balances is insufficient.

Client Comment *In such a case, it would indicate that the meta vault is attempting to withdraw more assets than the sub-vaults can provide, which would point to a more critical issue at the protocol level. We prefer not to complicate the checks here, as that would increase gas costs. In any case, the transaction will revert with an out-of-gas error if the loop fails to terminate.*

```
72 while (assetsToExit > 0) {
```

CVF-13 INFO

- **Category** Suboptimal
- **Source** VaultEnterExit.sol

Description Technically, the value of the “assets” argument isn’t capped here. Relying on assumptions about the calling code is a very dangerous practice.

Recommendation Use checked addition here.

Client Comment *Even if the entire supply of Gnosis or Ethereum tokens is staked, it will not exceed uint256.*

```
108 unchecked {
        // cannot overflow as it is capped with underlying asset total
            ↪ supply
110     totalAssetsAfter = _totalAssets + assets;
    }
```

CVF-14 INFO

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

Description Underflow is possible here.

Recommendation Explicitly check for underflow and revert with a meaningful error message.

Client Comment *The explicit check will increase gas costs. The revert can be investigated using common tools (e.g. Tenderly).*

265
```
// update total assets, vault state
subVaultsTotalAssets -= exitedAssets;
```

CVF-15 FIXED

- **Category** Suboptimal
- **Source** VaultSubVaults.sol

Description Comparison with one looks odd. If it is here to address possible rounding errors, then way is unreliable, as rounding errors could add up and be greater than one.

Recommendation Refactor the code to address rounding errors properly and compare with zero here.

370
```
if (unprocessedTickets <= 1) {
    // nothing to process
    return;
```

CVF-16 INFO

- **Category** Procedural
- **Source** VaultToken.sol

Description Forbidding transfers to zero address could break some contracts, that transfer tokens to zero address in order to make them permanently inaccessible.

Recommendation Don't forbid transfers to zero address.

Client Comment *The OpenZeppelin implementation of the ERC-20 standard forbids transfers to the zero address: L159. If someone implements such logic in their contract using an arbitrary token, there is a high chance it will revert, as many ERC-20 tokens inherit from the OpenZeppelin contract.*

```
48 if (from == address(0) || to == address(0)) revert Errors.
   ↪ ZeroAddress();
```

CVF-17 INFO

- **Category** Procedural
- **Source** VaultToken.sol

Description Here a low-level compiler introduced underflow check is used to enforce a business-level constraint, which is a bad practice, as it makes code harder to read, more error-prone, and also throws without any meaningful errors message, consuming all the remaining gas.

Recommendation Explicitly check that balance is sufficient, and revert with a meaningful error message in case it is not.

Client Comment *We won't add the extra balance check to avoid additional gas costs in the normal scenario where the sender has enough tokens for the transfer. In cases where the transaction is reverted due to insufficient balance, it can be investigated using modern tools such as Tenderly.*

```
49 _balances[from] -= amount;
```

CVF-18 INFO

- **Category** Procedural
- **Source** GnoBlocklistErc20Vault.sol

Description This check uses a business-level property as a low-level flag, which is a bad practice. It is unclear, why “admin” cannot be zero for an already initialized contract, and even if it is guaranteed in this particular version, relying on this created an implicit relationship between distant parts of the code, which could easily be broken.

Recommendation Refactor the code to use a more straightforward and reliable way for detecting contract updates. A common practice is to just use different functions for initialization and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
40 // if admin is already set, it's an upgrade from version 2 to 3
   if (admin != address(0)) {
```

CVF-19 INFO

- **Category** Procedural
- **Source** GnoBlocklistVault.sol

Description This check uses a business-level property as a low-level flag, which is a bad practice. It is unclear, why “admin” cannot be zero for an already initialized contract, and even if it is guaranteed in this particular version, relying on this created an implicit relationship between distant parts of the code, which could easily be broken.

Recommendation Refactor the code to use a more straightforward and reliable way for detecting contract updates. A common practice is to just use different functions for initialization and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
34 // if admin is already set, it's an upgrade from version 2 to 3
   if (admin != address(0)) {
```

CVF-20 INFO

- **Category** Procedural
- **Source** GnoErc20Vault.sol

Description This check uses a business-level property as a low-level flag, which is a bad practice. It is unclear, why “admin” cannot be zero for an already initialized contract, and even if it is guaranteed in this particular version, relying on this created an implicit relationship between distant parts of the code, which could easily be broken.

Recommendation Refactor the code to use a more straightforward and reliable way for detecting contract updates. A common practice is to just use different functions for initialization and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
73 // if admin is already set, it's an upgrade from version 2 to 3
   if (admin != address(0)) {
```

CVF-21 FIXED

- **Category** Procedural
- **Source** GnoGenesisVault.sol

Description Here a business-level constraint is used to prevent a low-level underflow, which is a bad practice, as it makes code harder to read and more error-prone. Also, it introduces implicit relationship between distant parts of the code, which could easily be proken.

Recommendation Use checked subtraction.

```
110 // cannot underflow because mintedShares < userMaxOsTokenShares
    return mintedShares < userMaxOsTokenShares ? userMaxOsTokenShares -
       ↪ mintedShares : 0;
```

CVF-22 INFO

- **Category** Procedural
- **Source** GnoPrivErc20Vault.sol

Description This check uses a business-level property as a low-level flag, which is a bad practice. It is unclear, why “admin” cannot be zero for an already initialized contract, and even if it is guaranteed in this particular version, relying on this created an implicit relationship between distant parts of the code, which could easily be broken.

Recommendation Refactor the code to use a more straightforward and reliable way for detecting contract updates. A common practice is to just use different functions for initialization and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
40 // if admin is already set, it's an upgrade from version 2 to 3
   if (admin != address(0)) {
```

CVF-23 INFO

- **Category** Procedural
- **Source** GnoPrivVault.sol

Description This check uses a business-level property as a low-level flag, which is a bad practice. It is unclear, why “admin” cannot be zero for an already initialized contract, and even if it is guaranteed in this particular version, relying on this created an implicit relationship between distant parts of the code, which could easily be broken.

Recommendation Refactor the code to use a more straightforward and reliable way for detecting contract updates. A common practice is to just use different functions for initialization and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
3 pragma solidity ^0.8.22;
```

CVF-24 INFO

- **Category** Procedural
- **Source** GnoPrivVault.sol

Description This check uses a business-level property as a low-level flag, which is a bad practice. It is unclear, why “admin” cannot be zero for an already initialized contract, and even if it is guaranteed in this particular version, relying on this created an implicit relationship between distant parts of the code, which could easily be broken.

Recommendation Refactor the code to use a more straightforward and reliable way for detecting contract updates. A common practice is to just use different functions for initialization and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
34 // if admin is already set, it's an upgrade from version 2 to 3
   if (admin != address(0)) {
```

CVF-25 INFO

- **Category** Procedural
- **Source** GnoVault.sol

Description This check uses a business-level property as a low-level flag, which is a bad practice. It is unclear, why “admin” cannot be zero for an already initialized contract, and even if it is guaranteed in this particular version, relying on this created an implicit relationship between distant parts of the code, which could easily be broken.

Recommendation Refactor the code to use a more straightforward and reliable way for detecting contract updates. A common practice is to just use different functions for initialization and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
69 // if admin is already set, it's an upgrade from version 2 to 3
70 if (admin != address(0)) {
```

CVF-26 INFO

- **Category** Procedural
- **Source** EthErc20Vault.sol

Description This check uses a business-level property as a low-level flag, which is a bad practice. It is unclear, why “admin” cannot be zero for an already initialized contract, and even if it is guaranteed in this particular version, relying on this created an implicit relationship between distant parts of the code, which could easily be broken.

Recommendation Refactor the code to use a more straightforward and reliable way for detecting contract updates. A common practice is to just use different functions for initialization and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
73 if (admin != address(0)) {
```

CVF-27 INFO

- **Category** Unclear behavior
- **Source** EthErc20Vault.sol

Description This call is performed even if “success” isn’t true.

Recommendation Don’t perform this check in case transfer didn’t succeed.

Client Comment *We won’t implement an ‘if success’ check here, as we believe it’s safer to perform the verification explicitly.*

```
111 _checkOsTokenPosition(msg.sender);
```

```
123 _checkOsTokenPosition(from);
```

CVF-28 INFO

- **Category** Procedural
- **Source** EthVault.sol

Description Here is business-level property is used as a low-level flag, which is a bad practice, as It is unclear, why “admin” cannot be zero for an initialized contract, and even if the current implementation guarantees this, relying on this creates an implicit relationship between distant parts of the code, which could beeasily broken.

Recommendation Refactor the code to use a more straightforward way for detecting updated. A common practice is to use different functions for initializaiton and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
68 // if admin is already set, it's an upgrade from version 4 to 5
   if (admin != address(0)) {
```

CVF-29 INFO

- **Category** Procedural
- **Source** EthBlocklistErc20Vault.sol

Description Here is business-level property is used as a low-level flag, which is a bad practice, as It is unclear, why “admin” cannot be zero for an initialized contract, and even if the current implementation guarantees this, relying on this creates an implicit relationship between distant parts of the code, which could beeasily broken.

Recommendation Refactor the code to use a more straightforward way for detecting updated. A common practice is to use different functions for initializaiton and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
43 // if admin is already set, it's an upgrade from version 4 to 5
   if (admin != address(0)) {
```

CVF-30 INFO

- **Category** Procedural
- **Source** EthBlocklistVault.sol

Description Here is business-level property is used as a low-level flag, which is a bad practice, as It is unclear, why “admin” cannot be zero for an initialized contract, and even if the current implementation guarantees this, relying on this creates an implicit relationship between distant parts of the code, which could beeasily broken.

Recommendation Refactor the code to use a more straightforward way for detecting updated. A common practice is to use different functions for initializaiton and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
42 // if admin is already set, it's an upgrade from version 4 to 5
   if (admin != address(0)) {
```

CVF-31 INFO

- **Category** Procedural
- **Source** EthPrivErc20Vault.sol

Description This check uses a business-level property as a low-level flag, which is a bad practice. It is unclear, why “admin” cannot be zero for an already initialized contract, and even if it is guaranteed in this particular version, relying on this created an implicit relationship between distant parts of the code, which could easily be broken.

Recommendation Refactor the code to use a more straightforward and reliable way for detecting contract updates. A common practice is to just use different functions for initialization and updates.

Client Comment *We won’t separate the initialization and upgrade functions in order to reduce the contract’s bytecode size. We’re optimizing for a minimal set of functions that are used only once during the contract’s lifetime, such as initialization.*

```
43 // if admin is already set, it's an upgrade from version 4 to 5
   if (admin != address(0)) {
```

CVF-32 INFO

- **Category** Procedural
- **Source** EthPrivVault.sol

Description This check uses a business-level property as a low-level flag, which is a bad practice. It is unclear, why "admin" cannot be zero for an already initialized contract, and even if it is guaranteed in this particular version, relying on this created an implicit relationship between distant parts of the code, which could easily be broken.

Recommendation Refactor the code to use a more straightforward and reliable way for detecting contract updates. A common practice is to just use different functions for initialization and updates.

Client Comment *We won't separate the initialization and upgrade functions in order to reduce the contract's bytecode size. We're optimizing for a minimal set of functions that are used only once during the contract's lifetime, such as initialization.*

```
42 // if admin is already set, it's an upgrade from version 4 to 5
   if (admin != address(0)) {
```

CVF-33 INFO

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

Description This event is emitted even if nothing actually change.

Recommendation Revert in case the vault is already registered, or at least don't emit event in such a case.

Client Comment *The contract has already been deployed and is immutable.*

```
36 emit VaultAdded(msg.sender, vault);
```

CVF-34 FIXED

- **Category** Suboptimal
- **Source** ConsolidationsChecker.sol

Description There is no check to ensure "signatures.length" is a multiple of "_signature-Length".

Recommendation Implement such a check.

```
63 function _isValidSignatures(uint256 requiredSignatures, bytes32
   ↪ message, bytes calldata signatures)
```

CVF-35 INFO

- **Category** Suboptimal
- **Source** OsToken.sol

Description Having both, a single variable and a mapping for the same thing looks redundant.

Recommendation Remove the variable and use only the mapping, or clearly explain why both are needed.

Client Comment *The contract has already been deployed and is immutable.*

```
16 address private immutable _vaultController;
```

```
19 mapping(address controller => bool enabled) public override
   ↪ controllers;
```

CVF-36 INFO

- **Category** Procedural
- **Source** PriceFeed.sol

Recommendation The value "10 ** decimals()" should be precomputed.

Client Comment *The contract has already been deployed and is immutable.*

```
38 return IOsTokenVaultController(osTokenVaultController).
   ↪ convertToAssets(10 ** decimals());
```

CVF-37 INFO

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

Recommendation Unchained initializers should be used here.

Client Comment *We won't implement this fix as the result is the same as when calling __ReentrancyGuard_init(). We use _init() in other places, so we want to keep it*

```
27 __ReentrancyGuard_init();
__RewardSplitter_init(_vault);
```

CVF-38 FIXED

- **Category** Bad datatype
- **Source** EIP712Utils.sol

Recommendation This constant hash value should be a named constant.

```
21 keccak256("EIP712Domain(string␣name,string␣version,uint256␣chainId,
       ↪ address␣verifyingContract)"),

23 keccak256("1"),
```

CVF-39 INFO

- **Category** Suboptimal
- **Source** ExitQueue.sol

Description This value is always equal to “availableShares”.

Recommendation Remove the “checkpointShares” variable and use “availableShres” instead.

Client Comment *We won’t apply this suggestion, as ’checkpointShares’ and ’available-Shares’ are different and applying it would break the logic.*

```
125 checkpointShares = currTotalTickets - prevTotalTickets;
```

CVF-40 FIXED

- **Category** Suboptimal
- **Source** OsTokenUtils.sol

Description The code structure is overcomplicated and confusing.

Recommendation Refactor like this: if (data.isLiquidation && Math.mulDiv (...) >= _hfLiqTreshold) revert ...;

```
70 if (!data.isLiquidation) {
       return receivedAssets;
   }

75 if (
       Math.mulDiv(data.depositedAssets * _wad, config.
           ↪ liqThresholdPercent, data.mintedAssets * _maxPercent)
            >= _hfLiqThreshold
   ) {
       revert Errors.InvalidHealthFactor();
80 }
```

CVF-41 INFO

- **Category** Suboptimal
- **Source** ValidatorUtils.sol

Description There is no length check for the “validator” argument.

Recommendation Implement proper length check.

Client Comment *The check is performed in getIsV1Validators function of the ValidatorUtils contract.*

```
70 function getValidatorDeposit(bytes calldata validator, bool
   ↪ isV1Validator)
```

CVF-42 INFO

- **Category** Suboptimal
- **Source** ValidatorUtils.sol

Description This way of detecting V1 validators looks weird.

Recommendation Implement a more straightforward way, such as explicit flag.

Client Comment *We must keep the signature of the registerValidators function compatible with previous versions of the vault, so we can’t use the flag here.*

```
98  bool isV1Validators = validatorsLength % _validatorV1DepositLength
    ↪ == 0;
    bool isV2Validators = validatorsLength % _validatorV2DepositLength
    ↪ == 0;
100 if (validatorsLength == 0 || (isV1Validators && isV2Validators) ||
    ↪ (!isV1Validators && !isV2Validators)) {
     revert Errors.InvalidValidators();
```

CVF-43 INFO

- **Category** Suboptimal
- **Source** KeeperOracles.sol

Description The assumption that underflow is not possible here is based on a business-level constraint that the total number of “true” values in the “isOracle” mapping equals to the value of the “totalOracles” variables. Relying on business-level constraints when preventing low-level issues such as underflows is a very dangerous practice, as an inconsistency in business logic could open an attack vector.

Recommendation Use checked subtraction here.

Client Comment *The contract has already been deployed and is immutable.*

```
57 // cannot underflow
   _totalOracles = totalOracles - 1;
```

CVF-44 INFO

- **Category** Suboptimal
- **Source** KeeperOracles.sol

Description There is no check to ensure the length of “signatures” is a multiple of “_signatureLength”.

Recommendation Add such a check.

Client Comment *The contract has already been deployed and is immutable.*

```
78 function _verifySignatures(uint256 requiredSignatures, bytes32
      ↪ message, bytes calldata signatures) internal view {
```

CVF-45 INFO

- **Category** Suboptimal
- **Source** KeeperRewards.sol

Description There are no range checks for these arguments.

Recommendation Implement proper checks.

Client Comment *The contract has already been deployed with rewardsDelay = 43200 and maxAvgRewardPerSecond = 6341958397. The range checks are unnecessary.*

```
67 uint256 _rewardsDelay,
   uint256 maxAvgRewardPerSecond
```

CVF-46 INFO

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

Description Overflow is indeed possible here, as “rewardsDelay” is “uint256”.

Recommendation Use checked addition, or change the type of “rewardsDelay” to “uint64”.

Client Comment *The contract has already been deployed with rewardsDelay = 43200. The overflow is not possible*

```
133 // cannot overflow as lastRewardsTimestamp & rewardsDelay are uint64
    return lastRewardsTimestamp + rewardsDelay < block.timestamp;
```

CVF-47 INFO

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

Description The “none + 1’ calculation may overflow even when “nonce” is “uint64”, because the sum is also calculated as “uint64”.

Recommendation Use checked addition or cast “nonce” to “uint256” before addition.

Client Comment *The nonce is updated every 43,200 seconds, so it would take over 13 million years to reach the maximum value of a uint64.*

```
144 // cannot overflow as nonce is uint64
    return nonce != 0 && nonce + 1 < rewardsNonce;
```

CVF-48 FIXED

- **Category** Documentation
- **Source** IBalancerVault.sol

Description This sounds like documentation for a function, while this comment actually belongs to a struct.

Recommendation Describe the structure fields in the comment, and move the logic description into the comment belonging to the corresponding function.

```
36  * @dev All tokens in a swap are either sent from the `sender`
        ↪ account to the Vault, or from the Vault to the
    * `recipient` account.
    *
    * If the caller is not `sender`, it must be an authorized relayer
        ↪ for them.
40  *
    * If `fromInternalBalance` is true, the `sender`'s␣Internal␣Balance␣
        ↪ will␣be␣preferred,␣performing␣an␣ERC20
    *␣transfer␣for␣the␣difference␣between␣the␣requested␣amount␣and␣the␣
        ↪ User's Internal Balance (if any). The `sender`
    * must have allowed the Vault to use their tokens via `IERC20.
        ↪ approve()`. This matches the behavior of
    * `joinPool`.
    *
    * If `toInternalBalance` is true, tokens will be deposited to `
        ↪ recipient`'s␣internal␣balance␣instead␣of
    *␣transferred.␣This␣matches␣the␣behavior␣of␣`exitPool`.
    *
    *␣Note␣that␣ETH␣cannot␣be␣deposited␣to␣or␣withdrawn␣from␣Internal␣
        ↪ Balance:␣attempting␣to␣do␣so␣will␣trigger␣a
50  *␣revert.
    ␣␣␣␣␣␣
```

CVF-49 INFO

- **Category** Procedural
- **Source** IDepositDataRegistry.sol

Recommendation The “depositDataManager” parameters should be indexed.

Client Comment *The contract has already been deployed and is immutable.*

```
20 event DepositDataManagerUpdated(address indexed vault, address
   ↪ depositDataManager);
```

```
37     address indexed vault, bytes32 depositDataRoot, uint256
          ↪ validatorIndex, address depositDataManager
```

CVF-50 INFO

- **Category** Procedural
- **Source** IGnoGenesisVault.sol

Recommendation Address parameters should be indexed.

Client Comment *The events are emitted only once during the lifetime of the contract, so there is no point in indexing the arguments.*

```
19 event Migrated(address receiver, uint256 assets, uint256 shares);
```

```
28 event GenesisVaultCreated(address admin, uint256 capacity, uint16
   ↪ feePercent, string metadataIpfsHash);
```

CVF-51 INFO

- **Category** Procedural
- **Source** IOsTokenConfig.sol

Recommendation The “vault” parameter should be indexed.

Client Comment *The contract is already deployed and immutable. The event is emitted at most once annually.*

```
18 event OsTokenConfigUpdated(address vault, uint128 liqBonusPercent,
   ↪ uint64 liqThresholdPercent, uint64 ltvPercent);
```

CVF-52 INFO

- **Category** Procedural
- **Source** IOsTokenConfig.sol

Recommendation The “newRedeemer” parameter should be indexed.

Client Comment *The contract is already deployed and is immutable. The event is emitted at most once annually.*

```
24 event RedeemerUpdated(address newRedeemer);
```

CVF-53 FIXED

- **Category** Procedural
- **Source** IOsTokenRedeemer.sol

Recommendation The parameters should be indexed.

```
39 event RedeemerUpdated(address newRedeemer);
```

```
45 event PositionsManagerUpdated(address positionsManager);
```

CVF-54 FIXED

- **Category** Procedural
- **Source** IOsTokenRedeemer.sol

Recommendation The “merkleRoot” parameters should be indexed.

```
52 event RedeemablePositionsProposed(bytes32 merkleRoot, string
   ↪ ipfsHash);
```

```
59 event RedeemablePositionsAccepted(bytes32 merkleRoot, string
   ↪ ipfsHash);
```

```
66 event RedeemablePositionsDenied(bytes32 merkleRoot, string ipfsHash)
   ↪ ;
```

```
73 event RedeemablePositionsRemoved(bytes32 merkleRoot, string ipfsHash
   ↪ );
```

CVF-55 INFO

- **Category** Procedural
- **Source** IOsTokenVaultEscrow.sol

Recommendation The parameter should be indexed.

Client Comment *The contract is already deployed and is immutable. The event is emitted at most once annually.*

```
117 event AuthenticatorUpdated(address newAuthenticator);
```

CVF-56 INFO

- **Category** Suboptimal
- **Source** BalancedCurator.sol

Description In case assetsToDeposit < depositSubVaultsCount, an array of zero deposits will be returned.

Recommendation Return an empty array in such a case.

Client Comment *It returns empty array.*

```
22 if (assetsToDeposit == 0) {
       return deposits;
```

```
31 uint256 amountPerVault = assetsToDeposit / depositSubVaultsCount;
```

CVF-57 FIXED

- **Category** Suboptimal
- **Source** BalancedCurator.sol

Description Due to rounding errors, the actual distributed value could be lower than “assetsToDeposit”.

Recommendation Change to code to distribute exactly “assetsToDeposit” amount of assets, even if distributed amounts for different values will differ by 1 unit.

```
31 uint256 amountPerVault = assetsToDeposit / depositSubVaultsCount;
```

CVF-58 FIXED

- **Category** Suboptimal
- **Source** BalancedCurator.sol

Description In case assetsToExit < depositSubVaultsCount, an array of zero exit requests will be returned.

Recommendation Return an empty array in such a case.

```
57 if (assetsToExit == 0) {
       return exitRequests;
```

```
68 uint256 amountPerVault = assetsToExit / exitSubVaultsCount;
```

CVF-59 FIXED

- **Category** Suboptimal
- **Source** BalancedCurator.sol

Description This implementation is inefficient.

Recommendation The following strategy seems to be more efficient: 1. Sort non-ejecting vaults by balance in ascending order. 2. Iterate through sorted array of vaults. 3. For each vault calculate the amount to exit, by dividing the remaining “assetsToExit” value by the number of remaining vaults. 4. Remove that much assets from the vault, by not more than the vaults balance. 5. After the iteration ensure the remaining assetsToExit is zero.

```
72 while (assetsToExit > 0) {
       for (uint256 i = 0; i < subVaultsCount;) {
```

CVF-60 INFO

- **Category** Procedural
- **Source** ERC20Upgradeable.sol

Description This check implements logic that deviates from the ERC20 standard. Also it increases gas consumption without any obvious benefits.

Recommendation Remove the check.

Client Comment *It decreases gas consumption, as the allowance won’t be updated if it’s set to uint256.max, which is common for dApps. The similar check is done in OZ contract: L296.*

```
73 if (allowed != type(uint256).max) allowance[from][msg.sender] =
   ↪ allowed - amount;
```

CVF-61 INFO

- **Category** Procedural
- **Source** ERC20Upgradeable.sol

Description Here an implicit underflow protection implemented by the compiler is used to enforce a business-level constraint. This is a bad practice as it makes the code harder to read and more error-prone. Also, such an implicit check throws without any meaningful error message, consuming all the remaining gas.

Recommendation Explicitly check that allowance is sufficient and revert with a meaningful error in case it is not.

Client Comment *We won't add the extra allowance check to avoid additional gas costs in the normal scenario where the sender has enough allowance. In cases where the transaction is reverted due to insufficient allowance, it can be investigated using modern tools such as Tenderly.*

```
73 if (allowed != type(uint256).max) allowance[from][msg.sender] =
   ↪ allowed - amount;
```

CVF-62 INFO

- **Category** Suboptimal
- **Source** ERC20Upgradeable.sol

Description It is not guaranteed that "_initialDomainSeparator" was calculated using the "_initialChainId" chain ID value, as "_initialChainId" and "_initialDomainSeparator" could be assigned in different transactions, and "_initialDomainSeparator" calculation uses "block.chainId" rather than "_initialChainId".

Recommendation Refactor the code to use "_initialChainId" instead of "block.chainId" when calculating the "_initialDomainSeparator" value.

Client Comment *The likelihood of the block.chainId value changing between transactions is very low. To use _initialChainId, we would need to add an extra argument to compute-DomainSeparator, which would complicate the logic in all calls and increase gas costs.*

```
116 return block.chainid == _initialChainId ? _initialDomainSeparator :
    ↪ _computeDomainSeparator();
```

CVF-63 FIXED

- **Category** Suboptimal
- **Source** Multicall.sol

Description Here a string is decoded from the error message and then encoded back in exactly the same format.

Recommendation Forward the returned error data to the called as is, without decoding.

```
27 revert(abi.decode(result, (string)));
```

CVF-64 FIXED

- **Category** Suboptimal
- **Source** Multicall.sol

Description The error data returned here doesn't contain the index of the failed inner call, which could make error investigations harder.

Recommendation Revert with a custom error, that includes as parameters:
1. The index of the failed inner call. 2. The raw error data returned from the failed inner call. 3. The data returned from the successful inner calls preceding the failed one.
Such information would help during error investigations.

```
27 revert(abi.decode(result, (string)));
```

9 Recommendations

CVF-65 INFO

- **Category** Procedural
- **Source** VaultImmutables.sol

Description Consider specifying as “^0.8.0” unless there is something special regarding this particular version.

Recommendation Also relevant for the following files: VaultAdmin.sol, VaultFee.sol, VaultState.sol, VaultValidators.sol, VaultVersion.sol, VaultEnterExit.sol, VaultMev.sol, VaultEthStaking.sol, VaultBlocklist.sol, VaultOsToken.sol, VaultSubVaults.sol, VaultToken.sol, VaultWhitelist.sol, VaultGnoStaking.sol, GnoOwnMevEscrow.sol, GnoSharedMevEscrow.sol, GnoMetaVault.sol, GnoMetaVaultFactory.sol, GnoBlocklistErc20Vault.sol, GnoBlocklistVault.sol, GnoErc20Vault.sol, GnoGenesisVault.sol, GnoPrivErc20Vault.sol, GnoVault.sol, GnoVaultFactory.sol, OwnMevEscrow.sol, SharedMevEscrow.sol, EthFoxVault.sol, EthMetaVault.sol, EthMetaVaultFactory.sol, EthErc20Vault.sol, EthVault.sol, EthBlocklistErc20Vault.sol, EthBlocklistVault.sol, EthGenesisVault.sol, EthPrivErc20Vault.sol, EthPrivVault.sol, EthVaultFactory.sol, VaultsRegistry.sol, ConsolidationsChecker.sol, DepositDataRegistry.sol, OsTokenVaultEscrow.sol, EthOsTokenVaultEscrow.sol, GnoOsTokenVaultEscrow.sol, OsToken.sol, OsTokenConfig.sol, OsTokenFlashLoans.sol, OsTokenRedeemer.sol, OsTokenVaultController.sol, PriceFeed.sol, RewardSplitter.sol,EthRewardSplitter.sol, GnoRewardSplitter.sol,RewardSplitterFactory.sol, Errors.sol, EIP712Utils.sol, ExitQueue.sol , OsTokenUtils.sol.

```
3 pragma solidity ^0.8.22;
```

CVF-66 FIXED

- **Category** Documentation
- **Source** VaultImmutables.sol

Description This description is incomplete, as this contract also define several internal functions.

Recommendation Fis the comment, or move the internal functions into some other place.

```
11 * @notice Defines the Vault common immutable variables
```

CVF-67 INFO

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

Recommendation The type for this variable should be “IKeeper”.

```
15 address internal immutable _keeper;
```

CVF-68 INFO

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

Recommendation The type for this variable should be “IVaultsRegistry”.

```
18 address internal immutable _vaultsRegistry;
```

CVF-69 INFO

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

Recommendation The type for the “keeper” argument should be “IKeeper”.

```
28 constructor(address keeper, address vaultsRegistry) {
```

CVF-70 INFO

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

Recommendation The type for the “vaultsRegistry” argument should be “IVaultsRegistry”.

```
28 constructor(address keeper, address vaultsRegistry) {
```

CVF-71 INFO

- **Category** Suboptimal
- **Source** VaultAdmin.sol

Recommendation This check is redundant as it is anyway possible to pass a dead admin address.

```
42 if (newAdmin == address(0)) revert Errors.ZeroAddress();
```

CVF-72 FIXED

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

Description This event is emitted even if nothing actually changed.

Recommendation This event is emitted even if nothing actually changed.

```
44 emit AdminUpdated(msg.sender, newAdmin);
```

CVF-73 INFO

- **Category** Procedural
- **Source** VaultFee.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
17 uint256 internal constant _maxFeePercent = 10_000; // @dev 100.00 %
   uint256 private constant _feeUpdateDelay = 3 days;
```

CVF-74 INFO

- **Category** Suboptimal
- **Source** VaultFee.sol

Description The semantics of the “false” value passed as the last argument is unclear.

Recommendation Put the argument name as a comment next to the value.

```
37 _setFeePercent(_feePercent, false);
```

CVF-75 INFO

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

Recommendation This check is redundant, as it is anyway possible to pass a dead fee recipient address.

```
46 if (_feeRecipient == address(0)) revert Errors.InvalidFeeRecipient()
   ↪ ;
```

CVF-76 FIXED

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

Description This event is emitted even if nothing actually changed.

```
50 emit FeeRecipientUpdated(msg.sender, _feeRecipient);
```

CVF-77 FIXED

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

Recommendation The values “120” and “100” should be named constants.

```
70 uint256 maxFeePercent = currentFeePercent > 0 ? (currentFeePercent *
   ↪  120) / 100 : 100;
```

CVF-78 FIXED

- **Category** Procedural
- **Source** VaultState.sol

Recommendation Brackets around comparison are redundant.

```
85 return (totalShares_ == 0) ? shares : Math.mulDiv(shares,
   ↪ _totalAssets, totalShares_);
```

CVF-79 FIXED

- **Category** Documentation
- **Source** VaultState.sol

Description The logic of this function is quite complicated and uses many different state variables.

Recommendation Clearly explain the logic in the documentation comment.

```
135 * @dev Internal function for processing rewards and penalties
```

CVF-80 INFO

- **Category** Readability
- **Source** VaultState.sol

Description The code below loos like it is always executed, while actually, it is executed only when “totalAssetsDelta” is positive.

Recommendation Put the rest of the function into an explicit “else” branch.

```
168     return;
}
```

CVF-81 INFO

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

Recommendation The type for these variables should be more specific.

```
32  address private immutable _depositDataRegistry;
```

```
38  address internal immutable _validatorsRegistry;
```

```
41  address private immutable _validatorsWithdrawals;
```

```
44  address private immutable _validatorsConsolidations;
```

```
47  address private immutable _consolidationsChecker;
```

CVF-82 INFO

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

Recommendation The type for these arguments should be more specific.

```
77  address depositDataRegistry,
    address validatorsRegistry,
    address validatorsWithdrawals,
80  address validatorsConsolidations,
    address consolidationsChecker
```

CVF-83 INFO

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

Description This event is emitted even if nothing actually changed.

```
188 emit ValidatorsManagerUpdated(msg.sender, validatorsManager_);
```

CVF-84 INFO

- **Category** Procedural
- **Source** VaultVersion.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
20 bytes4 private constant _initSelector = bytes4(keccak256("initialize
   ↪ (bytes)"));
```

CVF-85 INFO

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

Recommendation The return type should be more specific.

```
23 function implementation() external view override returns (address) {
```

CVF-86 INFO

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

Recommendation The type for the “newImplementation” argument should be more specific.

```
28 function upgradeToAndCall(address newImplementation, bytes memory
   ↪ data) public payable override onlyProxy {
```

```
33 function _authorizeUpgrade(address newImplementation) internal view
   ↪ override {
```

CVF-87 INFO

- **Category** Suboptimal
- **Source** VaultVersion.sol

Description Comparison with zero address is redundant, as it is anyway possible to pass a dead implementation, and dead implementation will most probably be caught in the subsequent checks, such as vault ID or version check.

Recommendation Remove comparison with zero address.

```
36 newImplementation == address(0) || ERC1967Utils.getImplementation()
      ↪ == newImplementation // cannot reinit the same implementation
```

CVF-88 INFO

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

Description There is no range check for the argument.

Recommendation Implement an appropriate check.

```
32 constructor(uint256 exitingAssetsClaimDelay) {
```

CVF-89 INFO

- **Category** Procedural
- **Source** VaultEnterExit.sol

Description The storage address for the exit request is calculated twice: once inside the “calculateExitAssets” call and another time here.

Recommendation Refactor the code to calculate this storage address only once.

```
71     calculateExitedAssets(msg.sender, positionTicket, timestamp,
          ↪ exitQueueIndex);
```

```
80 delete _exitRequests[keccak256(abi.encode(msg.sender, timestamp,
      ↪ positionTicket))];
```

CVF-90 INFO

- **Category** Suboptimal
- **Source** VaultEnterExit.sol

Description This check is redundant, as it is anyway possible to pass dead destination address.

Recommendation Remove this check.

```
104 if (to == address(0)) revert Errors.ZeroAddress();
```

CVF-91 INFO

- **Category** Suboptimal
- **Source** VaultEnterExit.sol

Description This check is redundant, as it is anyway possible to pass a dead receiver address.

Recommendation Remove this check.

```
137 if (receiver == address(0)) revert Errors.ZeroAddress();
```

CVF-92 INFO

- **Category** Readability
- **Source** VaultEnterExit.sol

Description The rest of the function looks like it is always executed, while actually it is executed only when the vault is collateralized.

Recommendation Put the rest of the function into an explicit “else” branch.

```
155     return type(uint256).max;
}
```

CVF-93 INFO

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

Recommendation The type for these variables should be more specific.

```
19 address private immutable _sharedMevEscrow;
20 address private _ownMevEscrow;
```

CVF-94 INFO

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

Recommendation The argument type should be more specific.

```
29 constructor(address sharedMevEscrow) {
```

CVF-95 INFO

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

Recommendation The return type should be more specific.

```
34 function mevEscrow() public view override returns (address) {
```

CVF-96 INFO

- **Category** Procedural
- **Source** VaultEthStaking.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
30 uint256 private constant _securityDeposit = 1e9;
```

CVF-97 INFO

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

Description This function should emit some event.

```
55 function receiveFromMevEscrow() external payable override {
```

CVF-98 FIXED

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

Description The variable name is confusing, as it sound like the total sum of deposits, while actually it is just the total number of deposits.

Recommendation Choose an unambiguous name.

```
70 uint256 totalDeposits = deposits.length;
```

CVF-99 FIXED

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

Description This event is emitted even if nothing actually changed.

```
52 emit BlocklistManagerUpdated(msg.sender, _blocklistManager);
```

CVF-100 INFO

- **Category** Procedural
- **Source** VaultOsToken.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
23 uint256 private constant _maxPercent = 1e18;
```

CVF-101 INFO

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

Description The semantics of the keys in this mapping is unclear.

Recommendation Give the key a descriptive name and/or explain in a documentation comment.

```
34 mapping(address => OsTokenPosition) private _positions;
```

CVF-102 INFO

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

Recommendation The type for the “osTokenVaultController” argument should be “IOsTokenVaultController”.

```
45 constructor(address osTokenVaultController, address osTokenConfig,
   ↪ address osTokenVaultEscrow) {
```

CVF-103 INFO

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

Recommendation The type for the “osTokenConfig” argument should be “IOsTokenConfig”.

```
45 constructor(address osTokenVaultController, address osTokenConfig,
   ↪ address osTokenVaultEscrow) {
```

CVF-104 INFO

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

Recommendation The type for the “osTokenVaultEscrow” argument should be “IOsTokenVaultEscrow”.

```
45 constructor(address osTokenVaultController, address osTokenConfig,
   ↪ address osTokenVaultEscrow) {
```

CVF-105 INFO

- **Category** Procedural
- **Source** VaultOsToken.sol

Description Here a low-level compiler-introduced underflow check is used to enforce a business-level constraints, which is a bad practice, as it makes code more error-prone, harder to read, and also throws without any meaningful error message consuming all the remaining gas.

Recommendation Explicitly check that there are enough shares, and revert with a meaningful error message if there are not.

```
78 // update osToken position
   position.shares -= osTokenShares;
```

CVF-106 INFO

- **Category** Procedural
- **Source** VaultSubVaults.sol

Description Constants are usually named IN_UPPER_CASE.

```
40 uint256 private constant _maxSubVaults = 50;
```

CVF-107 INFO

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

Recommendation The type for these variables should be more specific.

```
43 address private immutable _curatorsRegistry;
```

```
46 address public override subVaultsCurator;
```

```
49 address public override ejectingSubVault;
```

CVF-108 INFO

- **Category** Suboptimal
- **Source** VaultSubVaults.sol

Recommendation It would be more efficient to merge these two mappings into a single mapping whose keys are vaults and values are structs encapsulating the values of the original mappings.

```
52 mapping(address vault => DoubleEndedQueue.Bytes32Deque) private
   ↪ _subVaultsExits;
   mapping(address vault => SubVaultState state) private
   ↪ _subVaultsStates;
```

CVF-109 INFO

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

Recommendation The argument type should be more specific.

```
68 constructor(address curatorsRegistry) {
```

```
73 function subVaultsStates(address vault) external view override
   ↪ returns (SubVaultState memory) {
```

```
83 function setSubVaultsCurator(address curator) external override {
```

```
89 function addSubVault(address vault) public virtual override {
```

```
131 function ejectSubVault(address vault) public virtual override {
```

CVF-110 INFO

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

Recommendation The type for the “vault” arguments should be more specific.

```
73  function subVaultsStates(address vault) external view override
      ↪ returns (SubVaultState memory) {

89  function addSubVault(address vault) public virtual override {

131 function ejectSubVault(address vault) public virtual override {

547 function _peekSubVaultExit(address vault) private view returns (
      ↪ uint160 positionTicket, uint96 shares) {

563 function _pushSubVaultExit(address vault, uint160 positionTicket,
      ↪ uint96 shares, bool front) private {

579 function _popSubVaultExit(address vault) private returns (uint160
      ↪ positionTicket, uint96 shares) {

617 function _depositToVault(address vault, uint256 assets) internal
      ↪ virtual returns (uint256);
```

CVF-111 INFO

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

Recommendation The return type should be more specific.

```
78  function getSubVaults() public view override returns (address[]
      ↪ memory) {
```

CVF-112 INFO

- **Category** Suboptimal
- **Source** VaultSubVaults.sol

Description The comparison with zero address is redundant, as it is anyway possible to pass a dead vault address, and invalid vaults will anyway be filtered aout by the vaults registry check.

Recommendation Remove comparison with zero address.

92
```
if (vault == address(0) || vault == address(this) || !
   ↪ IVaultsRegistry(_vaultsRegistry).vaults(vault)) {
```

CVF-113 INFO

- **Category** Documentation
- **Source** VaultSubVaults.sol

Description The semantics of the last boolean argument is unclear.

Recommendation Put the argument name as a comment next to the argument value.

150
```
_pushSubVaultExit(vault, SafeCast.toUint160(positionTicket),
   ↪ SafeCast.toUint96(state.stakedShares), false);
```

CVF-114 INFO

- **Category** Suboptimal
- **Source** VaultSubVaults.sol

Recommendation Consider refactoring the code to increment the loop counter in one place.

208
```
    unchecked {
        // cannot realistically overflow
210     ++i;
    }
    continue;
```

222
```
unchecked {
    // cannot realistically overflow
    ++i;
}
```

CVF-115 FIXED

- **Category** Procedural
- **Source** VaultSubVaults.sol

Description These variables are only used with a loop and don't carry their values into next iteration.

Recommendation Move declarations of these variables into the loop.

233
```
uint256 leftShares;
uint256 exitedAssets;
uint256 exitedShares;
uint256 positionTicket;
uint256 positionShares;
SubVaultState memory subVaultState;
SubVaultExitRequest calldata exitRequest;
```

308
```
address vault;
```

310
```
uint256 vaultTotalShares;
SubVaultState memory vaultState;
```

CVF-116 FIXED

- **Category** Suboptimal
- **Source** VaultSubVaults.sol

Description Comparison with one looks odd here. If it's purpose is to address possible rounding errors, then such a way doesn't look reliable, as rounding errors could accumulate and be greater than one.

Recommendation Refactor the code if necessary and compare with zero.

254
```
if (leftShares > 1) {
```

CVF-117 FIXED

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

Description The semantics of the returned values is unclear.

Recommendation Give descriptive names to the returned values and/or explain in a documentation comment.

353
```
function _harvestAssets(IKeeperRewards.HarvestParams calldata)
    ↪ internal pure override returns (int256, bool) {
```

CVF-118 INFO

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

Recommendation The type for the “vaults” argument should be more specific.

```
363 function _enterSubVaultsExitQueue(address[] memory vaults, uint256[]
    ↪  memory balances) private nonReentrant {
```

```
440 function _checkSubVaultsExitClaims(address[] memory vaults) private
    ↪ view {
```

```
474 function _syncRewardsNonce(address[] memory vaults) private returns
    ↪ (bool) {
```

CVF-119 INFO

- **Category** Procedural
- **Source** VaultSubVaults.sol

Recommendation Consider refactoring the code to increment the loop counter in one place.

```
403     unchecked {
            // cannot realistically overflow
            ++i;
        }
```

```
426 unchecked {
        // cannot realistically overflow
        ++i;
    }
```

CVF-120 FIXED

- **Category** Readability
- **Source** VaultSubVaults.sol

Description These variables are used only inside the loop and don’t carry their values to the next iteration.

Recommendation Move these declarations inside the loop.

```
441 address vault;
    uint256 totalExitedTickets;
    uint256 positionTicket;
    uint256 exitShares;
```

CVF-121 INFO

- **Category** Suboptimal
- **Source** VaultSubVaults.sol

Description This check is redundant, as it is anyway possible to pass a dead curator address. Also, an invalid curator address would be caught by the curators registry check.

Recommendation Remove this check.

```
602 if (curator == address(0)) revert Errors.ZeroAddress();
```

CVF-122 FIXED

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

Recommendation This event is emitted even if nothing actually changed.

```
51 emit WhitelisterUpdated(msg.sender, _whitelister);
```

CVF-123 INFO

- **Category** Procedural
- **Source** VaultGnoStaking.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
32 uint256 private constant _securityDeposit = 1e9;
```

CVF-124 INFO

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

Recommendation The type for the “gnoToken” argument should be “IERC20”.

```
47 constructor(address gnoToken, address tokensConverterFactory) {
```

CVF-125 INFO

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

Recommendation The type for the “tokenConverterFactory” should be “ITokenConverterFactory”.

```
47 constructor(address gnoToken, address tokensConverterFactory) {
```

CVF-126 INFO

- **Category** Suboptimal
- **Source** VaultGnoStaking.sol

Description The value calculated here could be rounded.

Recommendation Consider explicitly requiring the original “depositAmount” value to be a multiple of 32.

```
93 // divide by 32 to convert mGNO to GNO
   depositData.depositAmount /= 32;
```

CVF-127 INFO

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

Recommendation The type for this variable should be more specific.

```
16 address payable public immutable override vault;
```

CVF-128 INFO

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

Recommendation The argument type should be more specific.

```
22 constructor(address _vault) {
```

CVF-129 INFO

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

Description The semantics of the returned value is unclear and the comment doesn't help much.

Recommendation Give a descriptive name to the returned value and/or explain in a documentation comment.

28
```
function harvest() external returns (uint256) {
```

38
```
    // always returns 0 as xDAI must be converted to GNO first
    return 0;
```

CVF-130 INFO

- **Category** Suboptimal
- **Source** GnoOwnMevEscrow.sol

Description As the function anyway always returns zero.

Recommendation Consider returning nothing.

38
```
// always returns 0 as xDAI must be converted to GNO first
return 0;
```

CVF-131 INFO

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

Recommendation The argument type should be "IVaultsRegistry".

22
```
constructor(address vaultsRegistry) {
```

CVF-132 INFO

- **Category** Procedural
- **Source** GnoMetaVault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

44
```
uint8 private constant _version = 3;
uint256 private constant _securityDeposit = 1e9;
```

CVF-133 INFO

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

Recommendation The type for the “vault” argument should be more specific.

```
90 function addSubVault(address vault) public virtual override(
   ↪ IVaultSubVaults, VaultSubVaults) {
```

```
97 function ejectSubVault(address vault) public virtual override(
   ↪ IVaultSubVaults, VaultSubVaults) {
```

```
153 function _depositToVault(address vault, uint256 assets) internal
   ↪ override returns (uint256) {
```

CVF-134 INFO

- **Category** Procedural
- **Source** GnoMetaVaultFactory.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
20 uint256 private constant _securityDeposit = 1e9;
```

CVF-135 INFO

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

Recommendation The type for this variable should be more specific.

```
27 address public immutable override implementation;
```

CVF-136 INFO

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

Recommendation The type for the “_implementation’ argument should be more specific.

```
39 constructor(address initialOwner, address _implementation,
   ↪ IVaultsRegistry vaultsRegistry, address gnoToken)
```

CVF-137 INFO

- **Category** Suboptimal
- **Source** GnoMetaVaultFactory.sol

Description This check is redundant, as it is anyway possible to pass a dead admin address.

Recommendation Remove this check.

```
49 if (admin == address(0)) revert Errors.ZeroAddress();
```

CVF-138 INFO

- **Category** Procedural
- **Source** GnoBlocklistErc20Vault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
22 uint8 private constant _version = 3;
```

CVF-139 INFO

- **Category** Procedural
- **Source** GnoBlocklistErc20Vault.sol

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

```
31 constructor(GnoErc20VaultConstructorArgs memory args) GnoErc20Vault(
   ↪ args) {}
```

CVF-140 INFO

- **Category** Procedural
- **Source** GnoBlocklistVault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
21 uint8 private constant _version = 3;
```

CVF-141 INFO

- **Category** Procedural
- **Source** GnoBlocklistVault.sol

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

```
30 constructor(GnoVaultConstructorArgs memory args) GnoVault(args) {}
```

CVF-142 INFO

- **Category** Procedural
- **Source** GnoErc20Vault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
45 uint8 private constant _version = 3;
```

CVF-143 INFO

- **Category** Procedural
- **Source** GnoGenesisVault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
25 uint8 private constant _version = 4;
```

CVF-144 INFO

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

Recommendation The type for the “poolEscrow” argument should be “IGnoPoolEscrow”.

```
44 constructor(GnoVaultConstructorArgs memory args, address poolEscrow,
   ↪  address rewardGnoToken) GnoVault(args) {
```

CVF-145 INFO

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

Recommendation The type for the “rewardGnoToken” should be “IRewardGnoToken”.

```
44 constructor(GnoVaultConstructorArgs memory args, address poolEscrow,
   ↪  address rewardGnoToken) GnoVault(args) {
```

CVF-146 INFO

- **Category** Suboptimal
- **Source** GnoGenesisVault.sol

Description This check is redundant, as it is anyway possible to pass a dead receiver address.

Recommendation Remove this check.

```
75 if (receiver == address(0)) revert Errors.ZeroAddress();
```

CVF-147 INFO

- **Category** Procedural
- **Source** GnoPrivErc20Vault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
22 uint8 private constant _version = 3;
```

CVF-148 INFO

- **Category** Procedural
- **Source** GnoPrivErc20Vault.sol

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

```
31 constructor(GnoErc20VaultConstructorArgs memory args) GnoErc20Vault(
   ↪ args) {}
```

CVF-149 INFO

- **Category** Procedural
- **Source** GnoPrivVault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
21 uint8 private constant _version = 3;
```

CVF-150 INFO

- **Category** Procedural
- **Source** GnoPrivVault.sol

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

```
30 constructor(GnoVaultConstructorArgs memory args) GnoVault(args) {}
```

CVF-151 INFO

- **Category** Procedural
- **Source** GnoVault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
41 uint8 private constant _version = 3;
```

CVF-152 INFO

- **Category** Procedural
- **Source** GnoVaultFactory.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
19 uint256 private constant _securityDeposit = 1e9;
```

CVF-153 INFO

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

Recommendation The type for these variables should be more specific.

```
26 address public immutable override implementation;
```

```
29 address public override ownMevEscrow;
```

CVF-154 INFO

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

Recommendation The type for the “_implementation” argument should be more specific.

```
40 constructor(address _implementation, IVaultsRegistry vaultsRegistry,
   ↪  address gnoToken) {
```

CVF-155 INFO

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

Recommendation The type for the “gnoToken” argument should be “IERC20”.

```
40 constructor(address _implementation, IVaultsRegistry vaultsRegistry,
   ↪  address gnoToken) {
```

CVF-156 INFO

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

Recommendation The return type should be more specific.

```
47 function createVault(bytes calldata params, bool isOwnMevEscrow)
   ↪ external override returns (address vault) {
```

CVF-157 INFO

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

Recommendation The type for this variable should be more specific.

```
15 address payable public immutable override vault;
```

CVF-158 INFO

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

Recommendation The argument type should be more specific.

```
18 constructor(address _vault) {
```

CVF-159 INFO

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

Description This event is emitted even if “msg.value” is zero.

```
38 emit MevReceived(msg.value);
```

CVF-160 INFO

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

Recommendation The argument type should be “IVaultsRegistry”.

```
19 constructor(address vaultsRegistry) {
```

CVF-161 INFO

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

Description This event is emitted even if “msg.value” is zero.

```
36 emit MevReceived(msg.value);
```

CVF-162 INFO

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

Recommendation Constants are usually named IN_UPPER_CASE.

```
40 uint8 private constant _version = 2;
```

CVF-163 INFO

- **Category** Documentation
- **Source** EthFoxVault.sol

Description The logic behind this check is unclear.

Recommendation Explain it in a documentation comment.

```
66 if (admin == address(0)) {
       revert Errors.UpgradeFailed();
   }
```

CVF-164 INFO

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

Recommendation This hash value should be a named constant.

```
107 return keccak256("EthFoxVault");
```

CVF-165 INFO

- **Category** Procedural
- **Source** EthMetaVault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
43 uint8 private constant _version = 5;
   uint256 private constant _securityDeposit = 1e9;
```

CVF-166 INFO

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

Recommendation The type for this variable should be more specific.

```
21 address public immutable override implementation;
```

CVF-167 INFO

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

Recommendation The type for the "_implementation" argument should be more specific.

```
32 constructor(address initialOwner, address _implementation,
      ↪ IVaultsRegistry vaultsRegistry) Ownable(initialOwner) {
```

CVF-168 INFO

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

Recommendation The return type should be more specific.

```
43 returns (address vault)
```

CVF-169 INFO

- **Category** Suboptimal
- **Source** EthMetaVaultFactory.sol

Description This check is redundant, as it is anyway possible to pass a dead admin address.

Recommendation Remove this check.

```
45 if (admin == address(0)) revert Errors.ZeroAddress();
```

CVF-170 INFO

- **Category** Procedural
- **Source** EthErc20Vault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
45 uint8 private constant _version = 5;
```

CVF-171 INFO

- **Category** Readability
- **Source** EthErc20Vault.sol

Description The rest of the function looks like it is always executed, while actually it is executed only when “admin” is zero.

Recommendation Put the rest of the function into an explicit “else” branch.

```
75     return;
}
```

CVF-172 INFO

- **Category** Suboptimal
- **Source** EthErc20Vault.sol

Description The semantics of the returned value is unclear.

Recommendation Give a descriptive name to the returned value and/or explain in the documentation comment.

```
91 returns (uint256)
```

CVF-173 INFO

- **Category** Procedural
- **Source** EthVault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
41 uint8 private constant _version = 5;
```

CVF-174 INFO

- **Category** Procedural
- **Source** EthBlocklistErc20Vault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
22 uint8 private constant _version = 5;
```

CVF-175 INFO

- **Category** Procedural
- **Source** EthBlocklistVault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
21 uint8 private constant _version = 5;
```

CVF-176 INFO

- **Category** Procedural
- **Source** EthGenesisVault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
27 uint8 private constant _version = 5;
```

CVF-177 INFO

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

Recommendation The type for the “poolEscrow” argument should be “IEthPoolEscrow”.

```
44 constructor(EthVaultConstructorArgs memory args, address poolEscrow,
   ↪  address rewardEthToken) EthVault(args) {
```

CVF-178 INFO

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

Recommendation The type for the “rewardEthToken” argument should be “IRewardEth-Token”.

```
44 constructor(EthVaultConstructorArgs memory args, address poolEscrow,
   ↪  address rewardEthToken) EthVault(args) {
```

CVF-179 INFO

- **Category** Procedural
- **Source** EthPrivErc20Vault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
22 uint8 private constant _version = 5;
```

CVF-180 INFO

- **Category** Procedural
- **Source** EthPrivVault.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
21 uint8 private constant _version = 5;
```

CVF-181 INFO

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

Recommendation The type for these variables should be more specific.

```
20 address public immutable override implementation;
```

```
23 address public override ownMevEscrow;
```

CVF-182 INFO

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

Recommendation The return type should be more specific.

```
43 returns (address vault)
```

CVF-183 INFO

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

Recommendation The key type for these maps should be more specific.

```
16 mapping(address => bool) public override vaults;
```

```
19 mapping(address => bool) public override factories;
```

```
22 mapping(address => bool) public override vaultImpls;
```

CVF-184 INFO

- **Category** Procedural
- **Source** VaultsRegistry.sol

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

```
29 constructor() Ownable(msg.sender) {}
```

CVF-185 INFO

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

Recommendation The argument type should be more specific.

```
32 function addVault(address vault) external override {
```

```
40 function addVaultImpl(address newImpl) external override onlyOwner {
```

```
47 function removeVaultImpl(address impl) external override onlyOwner {
```

```
54 function addFactory(address factory) external override onlyOwner {
```

```
61 function removeFactory(address factory) external override onlyOwner
   ↪ {
```

CVF-186 INFO

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

Recommendation Constants are usually named IN_UPPER_CASE.

```
17 uint256 private constant _signatureLength = 65;
   bytes32 private constant _consolidationsCheckerTypeHash =
```

CVF-187 INFO

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

Recommendation The type for this argument should be “IKeeper”.

```
27 constructor(address keeper) EIP712("ConsolidationsChecker", "1") {
```

CVF-188 INFO

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

Recommendation The type for the “Vault” arguments should be more specific.

```
32 function verifySignatures(address vault, bytes calldata validators,
   ↪ bytes calldata signatures)
```

```
43 function isValidSignatures(address vault, bytes calldata validators,
   ↪  bytes calldata signatures)
```

CVF-189 INFO

- **Category** Suboptimal
- **Source** DepositDataRegistry.sol

Recommendation It would be more efficient to merge these mappings into a single mapping whose keys are vaults, and values are structs encapsulating the values of the original mappings.

```
26 mapping(address => uint256) public override depositDataIndexes;
```

```
29 mapping(address => bytes32) public override depositDataRoots;
```

```
31 mapping(address => address) private _depositDataManagers;
   mapping(address => bool) private _migrated;
```

CVF-190 INFO

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

Recommendation The key type for these mappings should be more specific.

```
26 mapping(address => uint256) public override depositDataIndexes;
```

```
29 mapping(address => bytes32) public override depositDataRoots;
```

```
31 mapping(address => address) private _depositDataManagers;
   mapping(address => bool) private _migrated;
```

CVF-191 INFO

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

Recommendation The type for the “vault” arguments should be more specific.

```
38 modifier onlyValidVault(address vault) {
```

```
54 function getDepositDataManager(address vault) public view override
   ↪ returns (address) {
```

```
60 function setDepositDataManager(address vault, address
   ↪ depositDataManager) external override onlyValidVault(vault) {
```

```
70 function setDepositDataRoot(address vault, bytes32 depositDataRoot)
   ↪ external override onlyValidVault(vault) {
```

```
81 function updateVaultState(address vault, IKeeperRewards.
   ↪ HarvestParams calldata harvestParams) external override {
```

```
87     address vault,
```

```
118    address vault,
```

CVF-192 INFO

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

Recommendation The argument type should be “IVaultsRegistry”.

```
49 constructor(address vaultsRegistry) {
```

CVF-193 INFO

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

Recommendation The return type should be more specific.

```
54 function getDepositDataManager(address vault) public view override
   ↪ returns (address) {
```

CVF-194 INFO

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

Recommendation The type for the “depositDataManager” argument should be more specific.

```
60 function setDepositDataManager(address vault, address
   ↪ depositDataManager) external override onlyValidVault(vault) {
```

```
169 function migrate(bytes32 depositDataRoot, uint256 validatorIndex,
   ↪ address depositDataManager)
```

CVF-195 INFO

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

Description This event is emitted even if nothing actually changed.

```
66 emit DepositDataManagerUpdated(vault, depositDataManager);
```

CVF-196 INFO

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

Description There is no check to ensure “keeperParams.validators.length” is a multiple of “validatorsCount”.

Recommendation Implement such a check.

```
135 uint256 validatorLength = keeperParams.validators.length /
   ↪ validatorsCount;
```

CVF-197 INFO

- **Category** Procedural
- **Source** OsTokenVaultEscrow.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
22 uint256 private constant _maxPercent = 1e18;
   uint256 private constant _wad = 1e18;
   uint256 private constant _hfLiqThreshold = 1e18;
```

CVF-198 INFO

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

Recommendation The key type should be more specific.

```
29 mapping(address vault => mapping(uint256 positionTicket => Position)
   ↪ ) private _positions;
```

CVF-199 INFO

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

Recommendation The type for this argument should be “IOsTokenVaultController”.

```
50 address osTokenVaultController,
```

CVF-200 INFO

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

Recommendation The type for this argument should be “IOsTokenConfig”.

```
51 address osTokenConfig,
```

CVF-201 INFO

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

Recommendation The type for the “vault” argument should be more specific.

```
65  function getPosition(address vault, uint256 positionTicket) external
    ↪  view returns (address, uint256, uint256) {

100 function processExitedAssets(address vault, uint256
    ↪ exitPositionTicket, uint256 timestamp, uint256 exitQueueIndex)

124 function claimExitedAssets(address vault, uint256 exitPositionTicket
    ↪ , uint256 osTokenShares)

166 function liquidateOsToken(address vault, uint256 exitPositionTicket,
    ↪  uint256 osTokenShares, address receiver)

175 function redeemOsToken(address vault, uint256 exitPositionTicket,
    ↪ uint256 osTokenShares, address receiver)

224     address vault,
```

CVF-202 INFO

- **Category** Documentation
- **Source** OsTokenVaultEscrow.sol

Description The semantics of the returned valus is unclear.

Recommendation Give descriptive names to the returned values and/or explain in the documentation comment.

```
65  function getPosition(address vault, uint256 positionTicket) external
    ↪  view returns (address, uint256, uint256) {
```

CVF-203 INFO

- **Category** Suboptimal
- **Source** OsTokenVaultEscrow.sol

Recommendation This check is redundant, as it is anyway possible to pass a dead receiver address.

```
230 if (receiver == address(0)) revert Errors.ZeroAddress();
```

CVF-204 INFO

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

Recommendation Events are usually named via nouns, such as “AssetsReception”.

```
21 event AssetsReceived(address indexed sender, uint256 value);
```

CVF-205 FIXED

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

Recommendation The type for these arguments should be more specific.

```
33 address osTokenVaultController,
   address osTokenConfig,
```

CVF-206 INFO

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

Recommendation The types for these arguments should be more specific.

```
28 address osTokenVaultController,
   address osTokenConfig,
```

CVF-207 INFO

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

Recommendation The type for this argument should be “IERC20”.

```
34 address gnoToken
```

CVF-208 INFO

- **Category** Suboptimal
- **Source** OsToken.sol

Description This check is redundant, as it is anyway possible to pass a dead vault controller address.

Recommendation Remove this check.

```
33 if (vaultController == address(0)) revert Errors.ZeroAddress();
```

CVF-209 INFO

- **Category** Procedural
- **Source** OsTokenConfig.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
16 uint256 private constant _maxPercent = 1e18;
   uint256 private constant _disabledLiqThreshold = type(uint64).max;
```

CVF-210 INFO

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

Recommendation The key type for this mapping should be more specific.

```
24 mapping(address vault => Config config) private _vaultConfigs;
```

CVF-211 INFO

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

Description This check is redundant, as it is anyway possible to pass a dead owner address.

Recommendation Remove this check.

```
33 if (_owner == address(0)) revert Errors.ZeroAddress();
```

CVF-212 INFO

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

Recommendation The argument type should be more specific.

```
40 function getConfig(address vault) external view override returns (
      ↪ Config memory config) {
```

CVF-213 INFO

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

Recommendation The type for the “vault” argument should be more specific.

```
55 function updateConfig(address vault, Config memory config) public
      ↪ override onlyOwner {
```

CVF-214 INFO

- **Category** Procedural
- **Source** OsTokenFlashLoans.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
18 uint256 private constant _maxFlashLoanAmount = 100_000 ether;
```

CVF-215 INFO

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

Recommendation The type for this variable should be “IOsToken”.

```
19 address private immutable _osToken;
```

CVF-216 INFO

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

Recommendation The argument type should be “IERC20”.

```
25 constructor(address osToken) ReentrancyGuard() {
```

CVF-217 INFO

- **Category** Suboptimal
- **Source** OsTokenFlashLoans.sol

Recommendation Conversion of “_osToken” to “address” is redundant, as “_osToken” is already “address”.

```
54 IOsToken(address(_osToken)).burn(address(this), osTokenShares);
```

CVF-218 INFO

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

Recommendation The type for this variable should be more specific.

```
30 address public override positionsManager;
```

CVF-219 INFO

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

Recommendation The type for the “vaultsRegistry” argument should be “IVaultsRegistry”.

```
47 constructor(address vaultsRegistry_, address osToken_, address
   ↪ owner_, uint256 positionsUpdateDelay_)
```

CVF-220 INFO

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

Recommendation The type for the “osToken_” argument should be “IERC20”.

```
47 constructor(address vaultsRegistry_, address osToken_, address
   ↪ owner_, uint256 positionsUpdateDelay_)
```

CVF-221 INFO

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

Recommendation The argument type should be more specific.

```
68 function setPositionsManager(address positionsManager_) external
   ↪ override onlyOwner {
```

CVF-222 INFO

- **Category** Suboptimal
- **Source** OsTokenRedeemer.sol

Description These checks are redundant, as it is anyway possible to pass a dead address.

Recommendation Remove these checks.

```
69 if (positionsManager_ == address(0)) {
```

```
81 if (redeemer_ == address(0)) {
```

CVF-223 INFO

- **Category** Suboptimal
- **Source** OsTokenRedeemer.sol

Description Comparison with zero address looks redundant.

Recommendation Remove comparison with zero address.

```
193 if (position.vault == address(0) || !_vaultsRegistry.vaults(position
    ↪ .vault)) {
```

CVF-224 INFO

- **Category** Procedural
- **Source** OsTokenVaultController.sol

Recommendation Constants are usually named IN_UPPER_CASES.

```
20 uint256 private constant _wad = 1e18;
   uint256 private constant _maxFeePercent = 10_000; // @dev 100.00 %
```

CVF-225 INFO

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

Recommendation The type for these variables should be more specific.

```
23 address private immutable _registry;
   address private immutable _osToken;
```

```
27 address public override keeper;
```

```
36 address public override treasury;
```

CVF-226 INFO

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

Recommendation The type for these arguments should be more specific.

```
58 address _keeper,
   address registry,
60 address osToken,
   address _treasury,
```

CVF-227 INFO

- **Category** Suboptimal
- **Source** OsTokenVaultController.sol

Description This check is redundant, as it is anyway possible to pass a dead treasury address.

Recommendation Remove this check.

```
164 if (_treasury == address(0)) revert Errors.ZeroAddress();
```

CVF-228 INFO

- **Category** Readability
- **Source** OsTokenVaultController.sol

Description The rest of the function looks like it is always executed, while it is actually executed only when “treasuryAssets” is not zero.

Recommendation Put the rest of the function into an explicit “else” branch.

```
250     return;
}
```

CVF-229 INFO

- **Category** Procedural
- **Source** PriceFeed.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
19 uint256 public constant override version = 0;
```

CVF-230 INFO

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

Recommendation The type for this variable should be “IOsTokenVaultController”.

```
21 address public immutable osTokenVaultController;
```

CVF-231 INFO

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

Recommendation The type for the “_osTokenVaultController” argument should be “IOs-TokenVaultController”.

```
31 constructor(address _osTokenVaultController, string memory
   ↪ _description) {
```

CVF-232 INFO

- **Category** Procedural
- **Source** RewardSplitter.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
24 uint256 private constant _wad = 1e18;
```

CVF-233 INFO

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

Recommendation The type for this variable should be more specific.

```
27 address public override vault;
```

CVF-234 INFO

- **Category** Documentation
- **Source** RewardSplitter.sol

Description The semantics of the keys in these mappings is unclear.

Recommendation Give descriprive names to the keys and/or explain in a documentation comment.

```
35 mapping(address => ShareHolder) private _shareHolders;
   mapping(address => uint256) private _unclaimedRewards;
```

CVF-235 INFO

- **Category** Suboptimal
- **Source** RewardSplitter.sol

Recommendation It would be more efficient to merge these two mappings into a single mapping whose keys are accounts and values are structs encapsulating the values of the original mappings.

Client Comment *It's not possible to merge them.*

```
36 mapping(address => uint256) private _unclaimedRewards;
   mapping(uint256 positionTicket => address onBehalf) public override
      ↪ exitPositions;
```

CVF-236 INFO

- **Category** Suboptimal
- **Source** RewardSplitter.sol

Description This check is redundant, as it is anyway possible to pass a dead account address.

Recommendation Remove this check.

```
91 if (account == address(0)) revert InvalidAccount();
```

```
111 if (account == address(0)) revert InvalidAccount();
```

CVF-237 INFO

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

Recommendation The argument type should be more specific.

```
263 function __RewardSplitter_init(address _vault) internal
    ↪ onlyInitializing {
```

CVF-238 INFO

- **Category** Procedural
- **Source** EthRewardSplitter.sol

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

```
20 constructor() RewardSplitter() {}
```

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

CVF-239 INFO

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

Recommendation The argument type should be “IERC20”.

```
23 constructor(address gnoToken) RewardSplitter() {
```

CVF-240 INFO

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

Recommendation The argument type should be more specific.

```
28 function initialize(address _vault) external override initializer {
```

CVF-241 INFO

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

Recommendation The type for this variable should be more specifiic.

```
16 address public immutable override implementation;
```

CVF-242 INFO

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

Recommendation The argument type should be more specific.

```
22 constructor(address _implementation) {
```

```
27 function createRewardSplitter(address vault) external override
   ↪ returns (address rewardSplitter) {
```

CVF-243 INFO

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

Recommendation The return type should be “IRewardSplitter”.

```
27 function createRewardSplitter(address vault) external override
     ↪ returns (address rewardSplitter) {
```

CVF-244 INFO

- **Category** Procedural
- **Source** Errors.sol

Description This library only contains errors.

Recommendation Consider moving the errors to the top level and removing the library.

```
10 library Errors {
```

CVF-245 INFO

- **Category** Suboptimal
- **Source** Errors.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
12 error InvalidShares();
   error InvalidAssets();
```

```
15 error CapacityExceeded();
   error InvalidCapacity();
   error InvalidSecurityDeposit();
   error InvalidFeeRecipient();
   error InvalidFeePercent();
20 error NotHarvested();
   error NotCollateralized();
   error InvalidProof();
   error LowLtv();
   error InvalidPosition();
   error InvalidHealthFactor();
   error InvalidReceivedAssets();
   error InvalidTokenMeta();
   error UpgradeFailed();
   error InvalidValidators();
30 error DeadlineExpired();
   error PermitInvalidSigner();
   error InvalidValidatorsRegistryRoot();
   error InvalidVault();
   error AlreadyAdded();
   error AlreadyRemoved();
   error InvalidOracles();
   error NotEnoughSignatures();
   error InvalidOracle();
   error TooEarlyUpdate();
40 error InvalidAvgRewardPerSecond();
   error InvalidRewardsRoot();
   error HarvestFailed();
   error LiquidationDisabled();
   error InvalidLiqThresholdPercent();
   error InvalidLiqBonusPercent();
   error InvalidLtvPercent();
   error InvalidCheckpointIndex();
   error InvalidCheckpointValue();
   error MaxOraclesExceeded();
```

```
error ExitRequestNotProcessed();
error ValueNotChanged();
error FlashLoanFailed();
error CannotTopUpV1Validators();
error InvalidSignatures();
error EmptySubVaults();
error UnclaimedAssets();
error EjectingVault();
error InvalidCurator();
error RewardsNonceIsHigher();
error InvalidRedeemablePositions();
error RedeemablePositionsProposed();
```

CVF-246 INFO

- **Category** Procedural
- **Source** ExitQueue.sol

Description Relying on business-level constraints for preventing low-level issues, such as underflow, is a very dangerous practice, as business-level inconsistency could open an attack vector.

Recommendation Use checked subtraction.

```
// cannot underflow as prevTotalTickets <= positionTicket
checkpointShares = currTotalTickets - prevTotalTickets;
```

CVF-247 INFO

- **Category** Procedural
- **Source** OsTokenUtils.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
uint256 private constant _wad = 1e18;
uint256 private constant _hfLiqThreshold = 1e18;
uint256 private constant _maxPercent = 1e18;
uint256 private constant _disabledLiqThreshold = type(uint64).max;
```

CVF-248 FIXED

- **Category** Suboptimal
- **Source** OsTokenUtils.sol

Description The condition “data.isLiquidation” is checked twice.

Recommendation Refactor the code to check once.

```
51 if (data.isLiquidation && config.liqThresholdPercent ==
   ↪ _disabledLiqThreshold) {
```

```
56 if (data.isLiquidation) {
```

CVF-249 INFO

- **Category** Procedural
- **Source** ValidatorUtils.sol

Description Consider specifying as “^0.8.0” unless there is something special regarding this particular version.

Recommendation Also relevant for the following files: KeeperValidators.sol, Keeper.sol, KeeperOracles.sol, KeeperRewards.sol, IMulticall.sol, ISubVaultsCurator.sol, ICuratorsRegistry.sol, IBalancerRateProvider.sol, IBalancerVault.sol, IChainlinkAggregator.sol, IChainlinkV3Aggregator.sol, IConsolidationsChecker.sol, IKeeperOracles.sol, IKeeperRewards.sol, IKeeperValidators.sol, IDepositDataRegistry.sol, IVaultAdmin.sol, IVaultBlocklist.sol, IVaultVersion.sol, IVaultFee.sol, IVaultState.sol, IVaultValidators.sol, IVaultEnterExit.sol, IVaultOsToken.sol, IVaultMev.sol, IVaultEthStaking.sol, IEthErc20Vault.sol, IEthBlocklistErc20Vault.sol, IEthVault.sol, IEthBlocklistVault.sol, IEthFoxVault.sol, IEthGenesisVault.sol, IVaultSubVaults.sol, IEthMetaVault.sol, IEthMetaVaultFactory.sol, IEthPoolEscrow.sol, IEthPrivErc20Vault.sol, IEthPrivVault.sol, IEthValidatorsRegistry.sol, IEthVaultFactory.sol, IGnoBlocklistErc20Vault.sol, IVaultGnoStaking.sol, IGnoVault.sol, IGnoBlocklistVault.sol, IGnoErc20Vault.sol, IGnoGenesisVault.sol, IGnoMetaVault.sol, IGnoMetaVaultFactory.sol, IGnoPoolEscrow.sol, IGnoPrivErc20Vault.sol.

```
3 pragma solidity ^0.8.22;
```

CVF-250 INFO

- **Category** Procedural
- **Source** ValidatorUtils.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
17 bytes32 private constant _validatorsManagerTypeHash =
```

```
19 uint256 private constant _validatorV1DepositLength = 176;
20 uint256 private constant _validatorV2DepositLength = 184;
   uint256 private constant _validatorWithdrawalLength = 56;
   uint256 private constant _validatorConsolidationLength = 96;
   uint256 private constant _validatorMinEffectiveBalance = 32 ether;
   uint256 private constant _validatorMaxEffectiveBalance = 2048 ether;
```

CVF-251 INFO

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

Recommendation The type for this argument should be more specific.

```
54 address validatorsManager,
```

CVF-252 INFO

- **Category** Readability
- **Source** ValidatorUtils.sol

Description The code below looks like it is always executed, while it is actually executed only when “isTopUp” is “false”.

Recommendation Put the rest of the loop body into an explicit “else” branch.

```
152 }
```

CVF-253 INFO

- **Category** Procedural
- **Source** KeeperValidators.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
17 bytes32 private constant _registerValidatorsTypeHash = keccak256(
```

```
21 bytes32 private constant _updateExitSigTypeHash =
```

CVF-254 INFO

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

Recommendation The key type should be more specific.

```
27 mapping(address => uint256) public override exitSignaturesNonces;
```

CVF-255 INFO

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

Recommendation The type for this argument should be more specific.

```
78 address vault,
```

CVF-256 INFO

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

Recommendation The type for this argument should be more specific.

```
32 address sharedMevEscrow,
```

CVF-257 INFO

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

Description This check is redundant, as it is anyway possible to pass a dead owner address.

Recommendation Remove this check.

```
46 if (_owner == address(0)) revert Errors.ZeroAddress();
```

CVF-258 INFO

- **Category** Procedural
- **Source** KeeperOracles.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
17 uint256 internal constant _signatureLength = 65;
   uint256 private constant _maxOracles = 30;
```

CVF-259 INFO

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

Recommendation The key type should be more specific.

```
21 mapping(address => bool) public override isOracle;
```

CVF-260 INFO

- **Category** Procedural
- **Source** KeeperOracles.sol

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

```
29 constructor() Ownable(msg.sender) EIP712("KeeperOracles", "1") {}
```

CVF-261 INFO

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

Recommendation The argument type should be more specific.

```
32 function addOracle(address oracle) external override onlyOwner {
```

CVF-262 INFO

- **Category** Suboptimal
- **Source** KeeperOracles.sol

Description This looks overcomplicated.

Recommendation Simplify lije this: unchecked { if (++totalOracles > _maxOracles) revert (...); }

```
35 // SLOAD to memory
   uint256 _totalOracles = totalOracles;
   unchecked {
       // capped with _maxOracles
       _totalOracles += 1;
40 }
   if (_totalOracles > _maxOracles) revert Errors.MaxOraclesExceeded();
```

```
45 totalOracles = _totalOracles;
```

CVF-263 INFO

- **Category** Suboptimal
- **Source** KeeperOracles.sol

Description This looks overcomplicated.

Recommendation Simplify like this: unchecked { totalOracles -= 1; }

```
54 // SLOAD to memory
   uint256 _totalOracles;
   unchecked {
       // cannot underflow
       _totalOracles = totalOracles - 1;
   }
```

```
62 totalOracles = _totalOracles;
```

CVF-264 INFO

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

Description The function name suggests that it updated the contract's state, while actually it just emits an event.

Recommendation Rename to something like "logConfigUpdate".

```
68 function updateConfig(string calldata configIpfsHash) external
   ↪ override onlyOwner {
    emit ConfigUpdated(configIpfsHash);
70 }
```

CVF-265 INFO

- **Category** Suboptimal
- **Source** KeeperOracles.sol

Description The value "startIndex + _signatureLength' is calculated twice.

Recommendation Calculate once and reuse.

```
96  currentOracle = ECDSA.recover(data, signatures[startIndex:startIndex
    ↪  + _signatureLength]);
```

```
106 startIndex += _signatureLength;
```

CVF-266 INFO

- **Category** Procedural
- **Source** KeeperRewards.sol

Recommendation Constants are usually named IN_UPPER_CASE.

```
19 bytes32 private constant _rewardsUpdateTypeHash = keccak256(
```

CVF-267 INFO

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

Recommendation The type for this variable should be more specific.

```
25 address private immutable _sharedMevEscrow;
```

CVF-268 INFO

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

Recommendation The key type should be more specific.

```
35 mapping(address => Reward) public override rewards;
```

```
38 mapping(address => UnlockedMevReward) public override
   ↪ unlockedMevRewards;
```

CVF-269 INFO

- **Category** Suboptimal
- **Source** KeeperRewards.sol

Recommendation It would be more efficient to merge these two mappings into a single mapping whose keys are keeper addresses and values are structs encapsulating the values of the original mappings.

```
35 mapping(address => Reward) public override rewards;
```

```
38 mapping(address => UnlockedMevReward) public override
   ↪ unlockedMevRewards;
```

CVF-270 INFO

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

Recommendation The type for this argument should be more specific.

```
64 address sharedMevEscrow,
```

CVF-271 INFO

- **Category** Documentation
- **Source** KeeperRewards.sol

Description The comment is inaccurate, as “rewardsDelay” is actually “uint256”.

Recommendation Either fix the comment or change the type of the “rewardsDelay” variable.

```
133 // cannot overflow as lastRewardsTimestamp & rewardsDelay are uint64
```

CVF-272 INFO

- **Category** Documentation
- **Source** KeeperRewards.sol

Recommendation The comment doesn’t make sense, as in Solidity a sum of two uint64 values is also uint64, so it may overflow.

```
133 // cannot overflow as lastRewardsTimestamp & rewardsDelay are uint64
```

CVF-273 INFO

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

Recommendation The argument type should be more specific.

```
139 function isHarvestRequired(address vault) external view override
    ↪ returns (bool) {
```

```
150 function canHarvest(address vault) external view override returns (
    ↪ bool) {
```

```
156 function isCollateralized(address vault) public view override
    ↪ returns (bool) {
```

```
238 function _collateralize(address vault) internal {
```

CVF-274 INFO

- **Category** Suboptimal
- **Source** KeeperRewards.sol

Description The assumption that underflow isn't possible is based on a business-level constraint. Relying on business-level constraints for preventing low-level issues, such as underflows, is a very dangerous practice, as an inconsistency in business logic could open an attack vector.

Recommendation Use checked subtraction.

175
```
// cannot underflow as after first merkle root update nonce will be
   ↪ "2"
currentNonce -= 1;
```

CVF-275 INFO

- **Category** Suboptimal
- **Source** KeeperRewards.sol

Description Here a compiler-generate underflow check is used to enforce a business-level constraint. This is a bad practice, as underflow leads to throw without any meaningful error message, that consumes all the remaining gas.

Recommendation Explicitly check that rewards are sufficient and revert with a meaningful error message in case they are not.

196
```
// calculate total assets delta
totalAssetsDelta = params.reward - lastReward.assets;
```

207
```
    unlockedMevDelta = params.unlockedMevReward - unlockedMevRewards
       ↪ [msg.sender].assets;
```

CVF-276 INFO

- **Category** Suboptimal
- **Source** ISubVaultsCurator.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays.

53
```
address[] calldata subVaults,
uint256[] memory balances,
```

CVF-277 INFO

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

Recommendation The type for the “curator” arguments should be more specific.

```
16 event CuratorAdded(address indexed sender, address indexed curator);
```

```
23 event CuratorRemoved(address indexed sender, address indexed curator
   ↪ );
```

```
30 function curators(address curator) external view returns (bool);
```

```
42 function addCurator(address curator) external;
```

```
48 function removeCurator(address curator) external;
```

CVF-278 INFO

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

Recommendation Events are usually named via nouns, such as “Curator” or “CuratorRemoval”.

```
16 event CuratorAdded(address indexed sender, address indexed curator);
```

```
23 event CuratorRemoved(address indexed sender, address indexed curator
   ↪ );
```

CVF-279 FIXED

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

Description Despite the name, the function doesn’t return an array of curators, but rather checks whether a given address is a curator.

Recommendation Rename the function.

```
30 function curators(address curator) external view returns (bool);
```

CVF-280 INFO

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

Recommendation The type for the asset fields should be “IERC20”.

```
29 address assetIn;
30 address assetOut;
```

CVF-281 INFO

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

Description The semantics of the returned value is unclear.

Recommendation Give a descriptive name to the returned value and/or explain in the documentation comment.

```
75 returns (uint256);
```

CVF-282 INFO

- **Category** Procedural
- **Source** IChainlinkAggregator.sol

Recommendation Consider importing directly from the Chainlink repo.

```
8 * @dev Copied from https://github.com/smartcontractkit/chainlink/
    ↪ blob/master/contracts/src/v0.8/interfaces/AggregatorInterface.
    ↪ sol
```

CVF-283 INFO

- **Category** Procedural
- **Source** IChainlinkV3Aggregator.sol

Recommendation Consider importing directly from the Chainlink repo.

```
8 * @dev Copied from https://github.com/smartcontractkit/chainlink/
    ↪ blob/master/contracts/src/v0.8/interfaces/
    ↪ AggregatorV3Interface.sol
```

CVF-284 INFO

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

Recommendation The type for the “vault” arguments should be more specific.

```
19 function verifySignatures(address vault, bytes calldata validators,
   ↪ bytes calldata signatures) external;
```

```
28 function isValidSignatures(address vault, bytes calldata validators,
   ↪  bytes calldata signatures)
```

CVF-285 INFO

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

Recommendation Events are usually named via nouns, such as “Oracle” or “OracleRemoval”.

```
17 event OracleAdded(address indexed oracle);
```

```
23 event OracleRemoved(address indexed oracle);
```

```
29 event ConfigUpdated(string configIpfsHash);
```

CVF-286 INFO

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

Recommendation The type for the “oracle” parameters should be more specific.

```
17 event OracleAdded(address indexed oracle);
```

```
23 event OracleRemoved(address indexed oracle);
```

CVF-287 INFO

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

Recommendation The type for the “oracle” arguments should be more specific.

```
36 function isOracle(address oracle) external view returns (bool);
```

```
48 function addOracle(address oracle) external;
```

```
54 function removeOracle(address oracle) external;
```

CVF-288 INFO

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

Recommendation Events are usually named via nouns, such as “RewardsUpdate” or just “Rewards”.

```
22 event RewardsUpdated(
```

```
38 event Harvested(
```

```
46 event RewardsMinOraclesUpdated(uint256 oracles);
```

CVF-289 INFO

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

Recommendation The type for the “vault” parameter should be more specific.

```
39 address indexed vault, bytes32 indexed rewardsRoot, int256
   ↪ totalAssetsDelta, uint256 unlockedMevDelta
```

CVF-290 INFO

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

Recommendation The type for the “vault” arguments should be more specific.

```
140 function rewards(address vault) external view returns (int192 assets
    ↪ , uint64 nonce);

148 function unlockedMevRewards(address vault) external view returns (
    ↪ uint192 assets, uint64 nonce);

155 function isHarvestRequired(address vault) external view returns (
    ↪ bool);

162 function canHarvest(address vault) external view returns (bool);

175 function isCollateralized(address vault) external view returns (bool
    ↪ );
```

CVF-291 INFO

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

Recommendation The type for the “vault” parameters should be more specific.

```
19 event ValidatorsApproval(address indexed vault, string
    ↪ exitSignaturesIpfsHash);

29     address indexed caller, address indexed vault, uint256 nonce,
        ↪ string exitSignaturesIpfsHash
```

CVF-292 INFO

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

Recommendation Events are usually named via nouns, such as “ExitStrategies” or “ValidatorsMinOracles”.

```
28 event ExitSignaturesUpdated(
```

```
36 event ValidatorsMinOraclesUpdated(uint256 oracles);
```

CVF-293 INFO

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

Recommendation The type for the “vault” arguments should be more specific.

```
43 function exitSignaturesNonces(address vault) external view returns (
   ↪ uint256);
```

```
81     address vault,
```

CVF-294 INFO

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

Recommendation Events are usually named via nouns, such as “DepositDataManager” or “DepositDataRoot”.

```
20 event DepositDataManagerUpdated(address indexed vault, address
   ↪ depositDataManager);
```

```
27 event DepositDataRootUpdated(address indexed vault, bytes32
   ↪ depositDataRoot);
```

```
36 event DepositDataMigrated(
```

CVF-295 INFO

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

Recommendation The type for the “vault” parameters should be more specific.

```
20 event DepositDataManagerUpdated(address indexed vault, address
   ↪ depositDataManager);
```

```
27 event DepositDataRootUpdated(address indexed vault, bytes32
   ↪ depositDataRoot);
```

```
37     address indexed vault, bytes32 depositDataRoot, uint256
          ↪ validatorIndex, address depositDataManager
```

CVF-296 INFO

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

Recommendation The type for the “vault” arguments should be more specific.

```
45 function depositDataIndexes(address vault) external view returns (
   ↪ uint256 validatorIndex);
```

```
52 function depositDataRoots(address vault) external view returns (
   ↪ bytes32 depositDataRoot);
```

```
59 function getDepositDataManager(address vault) external view returns
   ↪ (address);
```

```
66 function setDepositDataManager(address vault, address
   ↪ depositDataManager) external;
```

```
73 function setDepositDataRoot(address vault, bytes32 depositDataRoot)
   ↪ external;
```

```
80 function updateVaultState(address vault, IKeeperRewards.
   ↪ HarvestParams calldata harvestParams) external;
```

```
89     address vault,
```

```
103    address vault,
```

CVF-297 INFO

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

Recommendation Events are usually named via nouns, such as “Metadata” or “Admin”.

```
16 event MetadataUpdated(address indexed caller, string
   ↪ metadataIpfsHash);
```

```
23 event AdminUpdated(address indexed caller, address newAdmin);
```

CVF-298 FIXED

- **Category** Procedural
- **Source** IVaultAdmin.sol

Recommendation The “newAdmin” parameter should be indexed.

```
23 event AdminUpdated(address indexed caller, address newAdmin);
```

CVF-299 INFO

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

Recommendation Events are usually named via nouns, such as “Blocklist” or “Blocklist-Manager”.

```
19 event BlocklistUpdated(address indexed caller, address indexed
   ↪ account, bool isBlocked);
```

```
26 event BlocklistManagerUpdated(address indexed caller, address
   ↪ indexed blocklistManager);
```

CVF-300 INFO

- **Category** Procedural
- **Source** IVaultVersion.sol

Recommendation The return type should be more specific.

```
30 function implementation() external view returns (address);
```

CVF-301 INFO

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

Recommendation Events are usually named via nouns, such as “FeeRecipient” or “FeePercent”.

```
18 event FeeRecipientUpdated(address indexed caller, address indexed
   ↪ feeRecipient);
```

```
25 event FeePercentUpdated(address indexed caller, uint16 feePercent);
```

CVF-302 INFO

- **Category** Documentation
- **Source** IVaultFee.sol

Description The number format of the “feePercent” parameter is unclear.

Recommendation Explain in the documentation comment.

```
23  * @param feePercent The new fee percent
```

```
25 event FeePercentUpdated(address indexed caller, uint16 feePercent);
```

CVF-303 INFO

- **Category** Documentation
- **Source** IVaultFee.sol

Description The number format of the returned value is unclear.

Recommendation Explain in the documentation comment.

```
35  * @return The fee percent applied by the Vault on the rewards
```

```
37 function feePercent() external view returns (uint16);
```

CVF-304 INFO

- **Category** Documentation
- **Source** IVaultFee.sol

Description The number format of the argument is unclear.

Recommendation Explain in the documentation comment.

```
47  * @param _feePercent The new fee percent
```

```
49 function setFeePercent(uint16 _feePercent) external;
```

CVF-305 INFO

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

Recommendation Events are usually named via nouns, such as “Checkpoint” or “FeeSharesMint”.

```
19 event CheckpointCreated(uint256 shares, uint256 assets);
```

```
27 event FeeSharesMinted(address receiver, uint256 shares, uint256
   ↪ assets);
```

```
33 event ExitingAssetsPenalized(uint256 penalty);
```

```
40 event AssetsDonated(address sender, uint256 assets);
```

CVF-306 INFO

- **Category** Procedural
- **Source** IVaultState.sol

Recommendation The “receiver” parameter should be indexed.

```
27 event FeeSharesMinted(address receiver, uint256 shares, uint256
   ↪ assets);
```

CVF-307 INFO

- **Category** Procedural
- **Source** IVaultState.sol

Recommendation The “sender” parameter should be indexed.

```
40 event AssetsDonated(address sender, uint256 assets);
```

CVF-308 INFO

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

Recommendation Events are usually named via nouns, such as “Validator” or “V2Validator”.

```
19 event ValidatorRegistered(bytes publicKey);
```

```
26 event V2ValidatorRegistered(bytes publicKey, uint256 amount);
```

```
34 event ValidatorWithdrawalSubmitted(bytes publicKey, uint256 amount,
   ↪ uint256 feePaid);
```

```
41 event ValidatorFunded(bytes publicKey, uint256 amount);
```

```
49 event ValidatorConsolidationSubmitted(bytes fromPublicKey, bytes
   ↪ toPublicKey, uint256 feePaid);
```

```
56 event KeysManagerUpdated(address indexed caller, address indexed
   ↪ keysManager);
```

```
63 event ValidatorsRootUpdated(address indexed caller, bytes32 indexed
   ↪ validatorsRoot);
```

```
70 event ValidatorsManagerUpdated(address indexed caller, address
   ↪ indexed validatorsManager);
```

CVF-309 INFO

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

Recommendation Events are usually named via nouns, such as “Deposit” or “Redemption”.

```
21 event Deposited(address indexed caller, address indexed receiver,
      ↪ uint256 assets, uint256 shares, address referrer);
```

```
30 event Redeemed(address indexed owner, address indexed receiver,
      ↪ uint256 assets, uint256 shares);
```

```
39 event ExitQueueEntered(address indexed owner, address indexed
      ↪ receiver, uint256 positionTicket, uint256 shares);
```

```
49 event V2ExitQueueEntered(
```

```
60 event ExitedAssetsClaimed(
```

CVF-310 INFO

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

Recommendation Events are usually named via nouns, such as “OsTokenMint” or “OsTokenBurn”.

```
22 event OsTokenMinted(address indexed caller, address receiver,
      ↪ uint256 assets, uint256 shares, address referrer);
```

```
30 event OsTokenBurned(address indexed caller, uint256 assets, uint256
      ↪ shares);
```

```
41 event OsTokenLiquidated(
```

```
59 event OsTokenRedeemed(
```

CVF-311 INFO

- **Category** Procedural
- **Source** IVaultOsToken.sol

Recommendation The “receiver” parameters should be indexed.

```
44 address receiver,
```

```
62 address receiver,
```

CVF-312 INFO

- **Category** Procedural
- **Source** IVaultToken.sol

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

```
16 interface IVaultToken is IERC20Permit, IERC20, IERC20Metadata,
       ↪ IVaultState, IVaultEnterExit {}
```

CVF-313 INFO

- **Category** Bad datatype
- **Source** IEthErc20Vault.sol

Recommendation The type for these fields should be more specific.

```
52 address keeper;
   address vaultsRegistry;
   address validatorsRegistry;
   address validatorsWithdrawals;
   address validatorsConsolidations;
   address consolidationsChecker;
   address osTokenVaultController;
   address osTokenConfig;
60 address osTokenVaultEscrow;
```

```
62 address depositDataRegistry;
```

CVF-314 INFO

- **Category** Procedural
- **Source** IEthBlocklistErc20Vault.sol

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

```
13 interface IEthBlocklistErc20Vault is IEthErc20Vault, IVaultBlocklist
   ↪ {}
```

CVF-315 INFO

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

Recommendation The type for these fields should be more specific.

```
50 address keeper;
   address vaultsRegistry;
   address validatorsRegistry;
   address validatorsWithdrawals;
   address validatorsConsolidations;
   address consolidationsChecker;
   address osTokenVaultController;
   address osTokenConfig;
   address osTokenVaultEscrow;
```

```
60 address depositDataRegistry;
```

CVF-316 INFO

- **Category** Procedural
- **Source** IEthBlocklistVault.sol

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

```
13 interface IEthBlocklistVault is IEthVault, IVaultBlocklist {}
```

CVF-317 INFO

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

Recommendation The type for these fields should be more specific.

```
47 address keeper;
   address vaultsRegistry;
   address validatorsRegistry;
50 address validatorsWithdrawals;
   address validatorsConsolidations;
   address consolidationsChecker;
```

```
54 address depositDataRegistry;
```

CVF-318 INFO

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

Recommendation Events are usually named via nouns, such as “UserEjection” or “EthFoxVault”.

```
63 event UserEjected(address user, uint256 shares);
```

```
89 event EthFoxVaultCreated(
```

CVF-319 INFO

- **Category** Procedural
- **Source** IEthFoxVault.sol

Recommendation The “user” parameter should be indexed.

```
63 event UserEjected(address user, uint256 shares);
```

CVF-320 INFO

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

Recommendation Events are usually named via nouns, such as “Migration” or “GenesisVault”.

```
19 event Migrated(address receiver, uint256 assets, uint256 shares);
```

```
28 event GenesisVaultCreated(address admin, uint256 capacity, uint16
   ↪ feePercent, string metadataIpfsHash);
```

CVF-321 INFO

- **Category** Procedural
- **Source** IEthGenesisVault.sol

Recommendation The “receiver” parameter should be indexed.

```
19 event Migrated(address receiver, uint256 assets, uint256 shares);
```

CVF-322 INFO

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

Recommendation The type for this field should be more specific.

```
29 address vault;
```

CVF-323 INFO

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

Recommendation Events are usually named via nouns, such as “RewardNonce” or “SubVaultHarvest”.

```
37 event RewardsNonceUpdated(uint256 rewardsNonce);
```

```
43 event SubVaultsHarvested(int256 totalAssetsDelta);
```

```
50 event SubVaultAdded(address indexed caller, address indexed vault);
```

```
64 event SubVaultEjected(address indexed caller, address indexed vault)
   ↪ ;
```

```
71 event SubVaultsCuratorUpdated(address indexed caller, address
   ↪ indexed curator);
```

CVF-324 INFO

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

Recommendation The type for the “vault” parameters should be more specific.

```
50 event SubVaultAdded(address indexed caller, address indexed vault);
```

```
57 event SubVaultEjecting(address indexed caller, address indexed vault
   ↪ );
```

```
64 event SubVaultEjected(address indexed caller, address indexed vault)
   ↪ ;
```

CVF-325 INFO

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

Recommendation The return types should be more specific.

```
77 function subVaultsCurator() external view returns (address);
```

```
83 function ejectingSubVault() external view returns (address);
```

```
89 function getSubVaults() external view returns (address[] memory);
```

CVF-326 INFO

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

Recommendation The argument type should be more specific.

```
96 function subVaultsStates(address vault) external view returns (
   ↪ SubVaultState memory);
```

```
102 function setSubVaultsCurator(address curator) external;
```

```
108 function addSubVault(address vault) external;
```

```
115 function ejectSubVault(address vault) external;
```

CVF-327 INFO

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

Recommendation The type for these fields should be more specific.

```
41 address keeper;
   address vaultsRegistry;
   address osTokenVaultController;
   address osTokenConfig;
   address osTokenVaultEscrow;
   address curatorsRegistry;
```

CVF-328 INFO

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

Recommendation The type for this field should be more specific.

```
58 address subVaultsCurator;
```

CVF-329 INFO

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

Recommendation Events are usually named via nouns, such as “MetaVault”.

```
18 event MetaVaultCreated(address indexed caller, address indexed admin
   ↪ , address indexed vault, bytes params);
```

CVF-330 INFO

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

Recommendation The return type should be more specific.

```
24 function implementation() external view returns (address);
```

```
37 function createVault(address admin, bytes calldata params) external
   ↪ payable returns (address vault);
```

CVF-331 INFO

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

Recommendation Events are usually named via nouns, such as “Withdrawal” or “OwnershipTransferCommit”.

```
18 event Withdrawn(address indexed sender, address indexed payee,
   ↪ uint256 amount);
```

```
25 event OwnershipTransferCommitted(address indexed currentOwner,
   ↪ address indexed futureOwner);
```

```
32 event OwnershipTransferApplied(address indexed previousOwner,
   ↪ address indexed newOwner);
```

CVF-332 INFO

- **Category** Procedural
- **Source** IEthPrivErc20Vault.sol

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

```
13 interface IEthPrivErc20Vault is IEthErc20Vault, IVaultWhitelist {}
```

CVF-333 INFO

- **Category** Procedural
- **Source** IEthPrivVault.sol

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

```
13 interface IEthPrivVault is IEthVault, IVaultWhitelist {}
```

CVF-334 INFO

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

Recommendation Events are usually named via nouns, such as “Vault”.

```
18 event VaultCreated(address indexed admin, address indexed vault,
   ↪ address ownMevEscrow, bytes params);
```

CVF-335 INFO

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

Recommendation The type for the “vault” parameter should be indexed.

```
18 event VaultCreated(address indexed admin, address indexed vault,
   ↪ address ownMevEscrow, bytes params);
```

CVF-336 INFO

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

Recommendation The type for the “ownMevEscrow” parameter should be more specific.

```
18 event VaultCreated(address indexed admin, address indexed vault,
      ↪ address ownMevEscrow, bytes params);
```

CVF-337 INFO

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

Recommendation The return type should be more specific.

```
24 function implementation() external view returns (address);
```

```
30 function ownMevEscrow() external view returns (address);
```

```
44 function createVault(bytes calldata params, bool isOwnMevEscrow)
      ↪ external payable returns (address vault);
```

CVF-338 INFO

- **Category** Procedural
- **Source** IGnoBlocklistErc20Vault.sol

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

```
13 interface IGnoBlocklistErc20Vault is IGnoErc20Vault, IVaultBlocklist
      ↪  {}
```

CVF-339 INFO

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

Recommendation Events are usually named via nouns, such as “XdaiSwap”.

```
19 event XdaiSwapped(uint256 amount, uint256 assets);
```

CVF-340 INFO

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

Recommendation The type for these fields should be more specific.

```
51  address keeper;
    address vaultsRegistry;
    address validatorsRegistry;
    address validatorsWithdrawals;
    address validatorsConsolidations;
    address consolidationsChecker;
    address osTokenVaultController;
    address osTokenConfig;
    address osTokenVaultEscrow;
60  address sharedMevEscrow;
    address depositDataRegistry;
    address gnoToken;
    address tokensConverterFactory;
```

CVF-341 INFO

- **Category** Procedural
- **Source** IGnoBlocklistVault.sol

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

```
13  interface IGnoBlocklistVault is IGnoVault, IVaultBlocklist {}
```

CVF-342 INFO

- **Category** Bad datatype
- **Source** IGnoErc20Vault.sol

Recommendation The type for these fields should be more specific.

```
53 address keeper;
   address vaultsRegistry;
   address validatorsRegistry;
   address validatorsWithdrawals;
   address validatorsConsolidations;
   address consolidationsChecker;
   address osTokenVaultController;
60 address osTokenConfig;
   address osTokenVaultEscrow;
   address sharedMevEscrow;
   address depositDataRegistry;
   address gnoToken;
   address tokensConverterFactory;
```

CVF-343 INFO

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

Recommendation Events are usually named via nouns, such as “Migration”, or “GenesisVault”.

```
19 event Migrated(address receiver, uint256 assets, uint256 shares);
```

```
28 event GenesisVaultCreated(address admin, uint256 capacity, uint16
      ↪ feePercent, string metadataIpfsHash);
```

CVF-344 INFO

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

Recommendation The type for these fields should be more specific.

```
41 address keeper;
   address vaultsRegistry;
   address osTokenVaultController;
   address osTokenConfig;
   address osTokenVaultEscrow;
   address curatorsRegistry;
   address gnoToken;
```

CVF-345 INFO

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

Recommendation Events are usually named via nouns, such as “MetaVault”.

```
18 event MetaVaultCreated(address indexed caller, address indexed admin
   ↪ , address indexed vault, bytes params);
```

CVF-346 INFO

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

Recommendation The type for the “vault” parameter should be more specific.

```
18 event MetaVaultCreated(address indexed caller, address indexed admin
   ↪ , address indexed vault, bytes params);
```

CVF-347 INFO

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

Recommendation The return type should be more specific.

```
24 function implementation() external view returns (address);
```

```
37 function createVault(address admin, bytes calldata params) external
   ↪ returns (address vault);
```

CVF-348 INFO

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

Recommendation Events are usually named via nouns, such as “Withdrawal” or “OwnershipTransferCommit”.

```
18 event Withdrawn(address indexed sender, address indexed payee,
   ↪ uint256 amount);
```

```
25 event OwnershipTransferCommitted(address indexed currentOwner,
   ↪ address indexed futureOwner);
```

```
32 event OwnershipTransferApplied(address indexed previousOwner,
   ↪ address indexed newOwner);
```

CVF-349 INFO

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

Recommendation The type for the “token” argument should be more specific.

```
62 function withdrawTokens(address token, address payee, uint256 amount
   ↪ ) external;
```

CVF-350 INFO

- **Category** Procedural
- **Source** IGnoPrivErc20Vault.sol

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

```
13 interface IGnoPrivErc20Vault is IGnoErc20Vault, IVaultWhitelist {}
```

CVF-351 INFO

- **Category** Procedural
- **Source** IGnoPrivVault.sol

Description Consider specifying as "^0.8.0" unless there is something special regarding this particular version.

Recommendation Also relevant for the following files: IGnoTokensConverter.sol, IGnoValidatorsRegistry.sol, IValidatorsRegistry.sol, IGnoVaultFactory.sol, IKeeper.sol, IMerkleDistributor.sol, IOsTokenFlashLoanRecipient.sol, IOsTokenFlashLoans.sol, IOsTokenConfig.sol, IOsTokenRedeemer.sol, IOsToken.sol, IOsTokenVaultController.sol, IOsTokenVaultEscrow.sol, IOsTokenVaultEscrowAuth.sol, IOwnMevEscrow.sol, IRewardEthToken.sol, IRewardGnoToken.sol, IRewardSplitter.sol, IRewardSplitterFactory.sol, ISavingsXDaiAdapter.sol, ISharedMevEscrow.sol, ITokensConverterFactory.sol, IValidatorsChecker.sol, IVaultsRegistry.sol, IVaultWhitelist.sol, BalancedCurator.sol, CuratorsRegistry.sol, ERC20Upgradeable.sol, Multicall.sol.

```
3 pragma solidity ^0.8.22;
```

CVF-352 INFO

- **Category** Procedural
- **Source** IGnoPrivVault.sol

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

```
13 interface IGnoPrivVault is IGnoVault, IVaultWhitelist {}
```

CVF-353 INFO

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

Recommendation Events are usually named via nouns, such as “Vault”.

```
18 event VaultCreated(address indexed admin, address indexed vault,
   ↪ address ownMevEscrow, bytes params);
```

CVF-354 INFO

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

Recommendation The type for the “vault” parameter should be more specific.

```
18 event VaultCreated(address indexed admin, address indexed vault,
   ↪ address ownMevEscrow, bytes params);
```

CVF-355 INFO

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

Recommendation The return type should be more specific.

```
24 function implementation() external view returns (address);
```

```
30 function ownMevEscrow() external view returns (address);
```

```
44 function createVault(bytes calldata params, bool isOwnMevEscrow)
   ↪ external returns (address vault);
```

CVF-356 INFO

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

Recommendation The type for the “token” argument should be more specific.

```
18 function distributeOneTime(address token, uint256 amount, string
   ↪ calldata rewardsIpfsHash, bytes calldata extraData)
```

CVF-357 INFO

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

Recommendation Events are usually named via nouns, such as “OsTokenConfig” or “Redeemer”.

```
18 event OsTokenConfigUpdated(address vault, uint128 liqBonusPercent,
   ↪ uint64 liqThresholdPercent, uint64 ltvPercent);
```

```
24 event RedeemerUpdated(address newRedeemer);
```

CVF-358 INFO

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

Recommendation The type for the “vault” parameter should be more specific.

```
18 event OsTokenConfigUpdated(address vault, uint128 liqBonusPercent,
   ↪ uint64 liqThresholdPercent, uint64 ltvPercent);
```

CVF-359 INFO

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

Recommendation The type for the “newRedeemer” parameter should be more specific.

```
24 event RedeemerUpdated(address newRedeemer);
```

CVF-360 INFO

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

Recommendation The return type should be more specific.

```
42 function redeemer() external view returns (address);
```

CVF-361 INFO

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

Recommendation The argument type should be more specific.

```
55 function setRedeemer(address newRedeemer) external;
```

CVF-362 INFO

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

Recommendation The type for the “vault” argument should be more specific.

```
62 function updateConfig(address vault, Config memory config) external;
```

CVF-363 INFO

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

Recommendation The type for this field should be more specific.

```
29 address vault;
```

CVF-364 INFO

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

Recommendation Events are usually named via noun s, such as “Redeemer” or “PositionManager”.

```
39 event RedeemerUpdated(address newRedeemer);
```

```
45 event PositionsManagerUpdated(address positionsManager);
```

```
52 event RedeemablePositionsProposed(bytes32 merkleRoot, string
   ↪ ipfsHash);
```

```
59 event RedeemablePositionsAccepted(bytes32 merkleRoot, string
   ↪ ipfsHash);
```

```
66 event RedeemablePositionsDenied(bytes32 merkleRoot, string ipfsHash)
   ↪ ;
```

```
73 event RedeemablePositionsRemoved(bytes32 merkleRoot, string ipfsHash
   ↪ );
```

CVF-365 INFO

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

Recommendation The parameter type should be more specific.

```
39 event RedeemerUpdated(address newRedeemer);
```

```
45 event PositionsManagerUpdated(address positionsManager);
```

CVF-366 INFO

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

Recommendation The return type should be indexed.

```
79 function redeemer() external view returns (address);
```

```
85 function positionsManager() external view returns (address);
```

CVF-367 INFO

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

Recommendation The argument type should be more specific.

```
105 function setRedeemer(address redeemer_) external;
```

```
111 function setPositionsManager(address positionsManager_) external;
```

CVF-368 INFO

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

Recommendation Events are usually named via nouns, such as “ControllerUpdate”.

```
21 event ControllerUpdated(address indexed controller, bool registered)
     ↪ ;
```

CVF-369 INFO

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

Recommendation The type for the “controller” parameter should be more specific.

```
21 event ControllerUpdated(address indexed controller, bool registered)
     ↪ ;
```

CVF-370 INFO

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

Recommendation The type for the “controller” arguments should be more specific.

```
28 function controllers(address controller) external view returns (bool
   ↪ );
```

```
49 function setController(address controller, bool registered) external
   ↪ ;
```

CVF-371 INFO

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

Recommendation The type for the “vault” parameter should be more specific.

```
18 event Mint(address indexed vault, address indexed receiver, uint256
   ↪ assets, uint256 shares);
```

```
27 event Burn(address indexed vault, address indexed owner, uint256
   ↪ assets, uint256 shares);
```

CVF-372 INFO

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

Recommendation Events are usually named via nouns, such as “State” or “Capacity”.

```
35 event StateUpdated(uint256 profitAccrued, uint256 treasuryShares,
   ↪ uint256 treasuryAssets);
```

```
41 event CapacityUpdated(uint256 capacity);
```

```
47 event TreasuryUpdated(address indexed treasury);
```

```
53 event FeePercentUpdated(uint16 feePercent);
```

```
59 event AvgRewardPerSecondUpdated(uint256 avgRewardPerSecond);
```

```
65 event KeeperUpdated(address keeper);
```

CVF-373 INFO

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

Recommendation The type for the “treasury” parameter should be more specific.

```
47 event TreasuryUpdated(address indexed treasury);
```

CVF-374 INFO

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

Recommendation The type for the “keeper” parameter should be more specific.

```
65 event KeeperUpdated(address keeper);
```

CVF-375 INFO

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

Recommendation The return type should be more specific.

```
77 function treasury() external view returns (address);
```

```
89 function keeper() external view returns (address);
```

CVF-376 INFO

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

Recommendation The argument type should be more specific.

```
154 function setTreasury(address _treasury) external;
```

```
172 function setKeeper(address _keeper) external;
```

CVF-377 INFO

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

Recommendation Events are usually named via nouns, such as “Position” or “ExitedAsset”.

```
35 event PositionCreated(
```

```
50 event ExitedAssetsProcessed(
```

```
63 event OsTokenLiquidated(
```

```
81 event OsTokenRedeemed(
```

```
98 event ExitedAssetsClaimed(
```

```
111 event LiqConfigUpdated(uint64 liqThresholdPercent, uint256
    ↪ liqBonusPercent);
```

```
117 event AuthenticatorUpdated(address newAuthenticator);
```

CVF-378 INFO

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

Recommendation The type for the “vault” parameters should be more specific.

```
36 address indexed vault,
```

```
51 address indexed vault, address indexed caller, uint256 indexed
    ↪ exitPositionTicket, uint256 exitedAssets
```

```
65 address indexed vault,
```

```
83 address indexed vault,
```

```
100 address indexed vault,
```

CVF-379 INFO

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

Description The return values actually don't have names mentioned in the documentation comment.

Recommendation Give names to the return values.

```
141  * @return owner The address of the assets owner
     * @return exitedAssets The amount of assets exited and ready to be
        ↪ claimed
     * @return osTokenShares The amount of osToken shares
145  function getPosition(address vault, uint256 positionTicket) external
        ↪  view returns (address, uint256, uint256);
```

CVF-380 INFO

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

Recommendation The type for the "vault" arguments should be more specific.

```
145  function getPosition(address vault, uint256 positionTicket) external
        ↪  view returns (address, uint256, uint256);
164  function processExitedAssets(address vault, uint256
        ↪ exitPositionTicket, uint256 timestamp, uint256 exitQueueIndex)
174  function claimExitedAssets(address vault, uint256 exitPositionTicket
        ↪ , uint256 osTokenShares)
185  function liquidateOsToken(address vault, uint256 exitPositionTicket,
        ↪  uint256 osTokenShares, address receiver)
195  function redeemOsToken(address vault, uint256 exitPositionTicket,
        ↪ uint256 osTokenShares, address receiver)
```

CVF-381 INFO

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

Recommendation The type for the “vault” argument should be more specific.

```
19 function canRegister(address vault, address owner, uint256
     ↪ exitPositionTicket, uint256 osTokenShares)
```

CVF-382 INFO

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

Recommendation Events are usually named via nouns, such as “Mev” or “Harvest”.

```
15 event MevReceived(uint256 assets);
```

```
21 event Harvested(uint256 assets);
```

CVF-383 INFO

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

Recommendation The returned value should be more specific.

```
27 function vault() external view returns (address payable);
```

CVF-384 INFO

- **Category** Suboptimal
- **Source** IRewardSplitter.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
16 error InvalidAccount();
   error InvalidAmount();
```

CVF-385 INFO

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

Recommendation Events are usually named via nouns, such as “Claimer” or “SharesIncrease”.

```
34 event ClaimerUpdated(address caller, address claimer);
```

```
41 event SharesIncreased(address indexed account, uint256 amount);
```

```
48 event SharesDecreased(address indexed account, uint256 amount);
```

```
55 event RewardsSynced(uint256 totalRewards, uint256 rewardPerShare);
```

```
62 event RewardsWithdrawn(address indexed account, uint256 amount);
```

```
70 event ExitQueueEnteredOnBehalf(address indexed onBehalf, uint256
   ↪ positionTicket, uint256 amount);
```

```
78 event ExitedAssetsClaimedOnBehalf(address indexed onBehalf, uint256
   ↪ positionTicket, uint256 amount);
```

CVF-386 FIXED

- **Category** Procedural
- **Source** IRewardSplitter.sol

Recommendation Both parameters should be indexed.

```
34 event ClaimerUpdated(address caller, address claimer);
```

CVF-387 INFO

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

Recommendation The return type should be more specifiic.

```
84 function vault() external view returns (address);
```

CVF-388 INFO

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

Recommendation Events are usually named via nous, such as “RewardSplitter”.

```
17 event RewardSplitterCreated(address owner, address vault, address
   ↪ rewardSplitter);
```

CVF-389 INFO

- **Category** Procedural
- **Source** IRewardSplitterFactory.sol

Recommendation All three parameters should be indexed.

```
17 event RewardSplitterCreated(address owner, address vault, address
   ↪ rewardSplitter);
```

CVF-390 INFO

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

Recommendation The return type should be more specific.

```
23 function implementation() external view returns (address);
```

```
30 function createRewardSplitter(address vault) external returns (
   ↪ address rewardSplitter);
```

CVF-391 INFO

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

Recommendation The argument type should be more specific.

```
30 function createRewardSplitter(address vault) external returns (
   ↪ address rewardSplitter);
```

CVF-393 INFO

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

Recommendation Events are usually named via nouse, such as “Mev” or “Harvest”.

```
15 event MevReceived(uint256 assets);
```

```
22 event Harvested(address indexed caller, uint256 assets);
```

CVF-392 INFO

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

Description The semantics of the returned value is unclear.

Recommendation Give a descriptive name to the returned value.

```
16 function depositXDAI(address receiver) external payable returns (
   ↪ uint256);
```

CVF-394 INFO

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

Recommendation The argument type should be more specific.

```
16 function createConverter(address vault) external returns (address
   ↪ converter);
```

```
23 function getTokensConverter(address vault) external view returns (
   ↪ address converter);
```

CVF-395 INFO

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

Recommendation The return type should be more specific.

```
16 function createConverter(address vault) external returns (address
   ↪ converter);
```

```
23 function getTokensConverter(address vault) external view returns (
   ↪ address converter);
```

CVF-396 INFO

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

Recommendation The type for this field should be more specific.

```
36 address vault;
```

CVF-397 INFO

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

Recommendation The type for the “vault” arguments should be more specific.

```
49 function updateVaultState(address vault, IKeeperRewards.
   ↪ HarvestParams calldata harvestParams) external;
```

```
56 function getExitQueueCumulativeTickets(address vault) external view
   ↪ returns (uint256);
```

```
65 function getExitQueueMissingAssets(address vault, uint256
   ↪ withdrawingAssets, uint256 targetCumulativeTickets)
```

```
80     address vault,
```

CVF-398 INFO

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

Description The semantics of the returned value is unclear.

Recommendation Give a descriptive name to the returned value.

```
56 function getExitQueueCumulativeTickets(address vault) external view
   ↪ returns (uint256);
```

CVF-399 INFO

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

Recommendation Events are usually named via nouns, such as “Vault” or “VaultImpl”.

```
16 event VaultAdded(address indexed caller, address indexed vault);
```

```
22 event VaultImplAdded(address indexed impl);
```

```
28 event VaultImplRemoved(address indexed impl);
```

```
34 event FactoryAdded(address indexed factory);
```

```
40 event FactoryRemoved(address indexed factory);
```

CVF-400 INFO

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

Recommendation The type for the “vault” parameter should be more specific.

```
16 event VaultAdded(address indexed caller, address indexed vault);
```

CVF-401 INFO

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

Recommendation The type for the “impl” parameters should be more specific.

```
22 event VaultImplAdded(address indexed impl);
```

```
28 event VaultImplRemoved(address indexed impl);
```

CVF-402 INFO

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

Recommendation The type for the “factory” parameters should be more specific.

```
34 event FactoryAdded(address indexed factory);
```

```
40 event FactoryRemoved(address indexed factory);
```

CVF-403 INFO

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

Recommendation The argument type should be more specific.

```
47 function vaults(address vault) external view returns (bool);
```

```
54 function vaultImpls(address impl) external view returns (bool);
```

```
61 function factories(address factory) external view returns (bool);
```

```
67 function addVault(address vault) external;
```

```
73 function addVaultImpl(address newImpl) external;
```

```
79 function removeVaultImpl(address impl) external;
```

```
85 function addFactory(address factory) external;
```

```
91 function removeFactory(address factory) external;
```

CVF-404 INFO

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

Recommendation Events are usually named via nouns, such as “WhitelistUpdate” or “WhitelisterUpdate”.

```
19 event WhitelistUpdated(address indexed caller, address indexed
   ↪ account, bool approved);
```

```
26 event WhitelisterUpdated(address indexed caller, address indexed
   ↪ whitelister);
```

CVF-405 FIXED

- **Category** Suboptimal
- **Source** BalancedCurator.sol

Description The latter assignment has no effect, as both assignments are performed by reference.

Recommendation Remove the latter assignment.

```
79 exitRequest = exitRequests[i];
```

```
82 exitRequests[i] = exitRequest;
```

CVF-406 INFO

- **Category** Procedural
- **Source** CuratorsRegistry.sol

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

```
23 constructor() Ownable(msg.sender) {}
```

CVF-407 FIXED

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

Description These events are emitted even if nothing actually changed.

```
28 emit CuratorAdded(msg.sender, curator);
```

```
34 emit CuratorRemoved(msg.sender, curator);
```

CVF-408 INFO

- **Category** Suboptimal
- **Source** CuratorsRegistry.sol

Description This check is redundant, as it is anyway possible to pass a dead owner address.

Recommendation Remove this check.

```
39 if (_owner == address(0)) revert Errors.ZeroAddress();
```

CVF-409 INFO

- **Category** Readability
- **Source** ERC20Upgradeable.sol

Recommendation Consider giving names to the keys and values.

```
31 mapping(address => mapping(address => uint256)) public override
      ↪ allowance;
```

```
34 mapping(address => uint256) public override nonces;
```

CVF-410 INFO

- **Category** Documentation
- **Source** ERC20Upgradeable.sol

Description It is unclear what variable this comment refers to.

Recommendation Put this comment next to the variable it refers to.

```
43 * @dev Since the immutable variable value is stored in the bytecode,
   *      its value would be shared among all proxies pointing to a
      ↪ given contract instead of each 'proxys storage.
```

CVF-411 INFO

- **Category** Suboptimal
- **Source** ERC20Upgradeable.sol

Description This check is redundant, as there is nothing bad with approving to the zero address.

Recommendation Remove this check.

```
55 if (spender == address(0)) revert Errors.ZeroAddress();
```

```
85 if (spender == address(0)) revert Errors.ZeroAddress();
```

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