Report

v. 2.0

Customer

Othernet Global PTE LTD

Smart Contract Audit

Salty.IO

5th January 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	05.01.24	A. Zveryanskaya	Initial Draft
0.2	05.01.24	A. Zveryanskaya	Minor revision
1.0	05.01.24	A. Zveryanskaya	Release
1.1	05.01.24	A. Zveryanskaya	CVF-10 client comment removed, project name fixed
2.0	05.01.24	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.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

dao/			
	DAO.sol	DAOConfig.sol	Parameters.sol
	Proposals.sol		
/			
	Upkeep.sol	SigningTools.sol	ExchangeConfig.sol
	AccessManager.sol		
dao/interfaces/			
	ICalledContract.sol	IDAO.sol	IDAOConfig.sol
	IProposals.sol		
interfaces/			
	IAccessManager.sol	IExchangeConfig.sol	IManagedWallet.sol
	ISalt.sol	IUpkeep.sol	
launch/			
	Airdrop.sol	BootstrapBallot.sol	InitialDistribution.sol
launch/interfaces/			
	IAirdrop.sol	IBootstrapBallot.sol	IInitialDistribution.sol

arbitrage/		
ArbitrageSearch.sol		
pools/		
PoolUtils.sol	PoolStats.sol	PoolsConfig.sol
Pools.sol	PoolMath.sol	
pools/interfaces/		
IPoolStats.sol	IPools.sol	IPoolsConfig.sol
price_feed/interfaces/		
IPriceAggregator.sol	IPriceFeed.sol	
price_feed/		
CoreChainlinkFeed.sol	CoreSaltyFeed.sol	CoreUniswapFeed.sol
PriceAggregator.sol		
rewards/		
Emissions.sol	RewardsConfig.sol	RewardsEmitter.sol
SaltRewards.sol		
rewards/interfaces/		
IEmissions.sol	IRewardsConfig.sol	IRewardsEmitter.sol
ISaltRewards.sol		
stable/		
USDS.sol	StableConfig.sol	Liquidizer.sol
CollateralAnd Liquidity.sol		
stable/interfaces/		
IUSDS.sol	IStableConfig.sol	ILiquidizer.sol
ICollateralAnd Liquidity.sol		
staking/		
StakingRewards.sol	StakingConfig.sol	Staking.sol
Liquidity.sol		
staking/interfaces/		
IStakingRewards.sol	IStakingConfig.sol	IStaking.sol
ILiquidity.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.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 9 major, and a few less important issues. All identified Major issues have been fixed.

	Info	Fixed
Major	0	9
Moderate	31	1
Minor	125	62

Fixed 72 out of 228 issues

6 Major Issues

CVF-3. FIXED

- **Category** Suboptimal
- **Source** Liquidity.sol

Description Each token balance is obtained twice.

Recommendation Consider obtaining once and reusing.

```
116 if ( tokenA.balanceOf( address(this) ) > startingBalanceA )
        tokenA.safeTransfer( msg.sender, tokenA.balanceOf( address(
          ↪ this) ) - startingBalanceA );
```

```
119 if ( tokenB.balanceOf( address(this) ) > startingBalanceB )
120         tokenB.safeTransfer( msg.sender, tokenB.balanceOf( address(
              ↪ this) ) - startingBalanceB );
```

CVF-4. FIXED

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

Description Using balance changes to tell how much tokens were (or were not) transferred is a bad practice as it is prone to reentrancy attacks.

Recommendation Consider just calculating the differences between the original max amounts and the actual added amounts.

```
116 if ( tokenA.balanceOf( address(this) ) > startingBalanceA )
```

```
119 if ( tokenB.balanceOf( address(this) ) > startingBalanceB )
```

CVF-14. FIXED

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

Description If the ”profitsForPools” array is longer than the ”poolIDs” array, the remaining values in the former array will be silently ignored.

Recommendation Consider reverting in case the arrays are of different length.

```
63 function _sendLiquidityRewards( uint256 liquidityRewardsAmount,
      ↪ uint256 directRewardsForSaltUSDS, bytes32[] memory poolIDs,
      ↪ uint256[] memory profitsForPools, uint256 totalProfits )
      ↪ internal
```

```
109 function sendInitialSaltRewards( uint256 liquidityBootstrapAmount,
      ↪ bytes32[] calldata poolIDs ) external
```

```
120 function performUpkeep( bytes32[] calldata poolIDs, uint256[]
      ↪ calldata profitsForPools ) external
```

CVF-15. FIXED

- **Category** Documentation
- **Source** PoolMath.sol

Description The constraint seems incorrect, as it leads to z0 = r0.

Client Comment *Documentation updated.*

```
12 // Assuming z0 in excess, determine how much z0 should be swapped to
      ↪  z1 first for z0/z1 = r0/z1 so that liquidity can be added
```

CVF-16. FIXED

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

Recommendation Calculations would be simpler and less likely to overflow, if the following values would be used: A = 1 B = 2r0 C = r0(r0z1 - r1z0)/(r1 + z1)

```
101 A = r1 + z1
    B = 2r0(r1 + z1)
    C = r0(r0z1 - r1z0)
```

CVF-17. FIXED

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

Description Reducing precision to a fixed number of decimals is a bad idea, as there could be very cheap tokens or very expensive tokens.

Recommendation Consider normalizing the input values to certain number of bits:128 bits should work fine. Calculate the index of the most significant bit in max(reserve0, zapAmount0). Then shift both, reserve0 and zapAmount0, so their maximum is exactly 128 bits wide. Do the same with reserve1 and zapAmount1. After calculations, shift the result back.

Client Comment *Fixed as suggested - though I needed to normalize to 80 bits to prevent overflow.*

```
113 uint256 constant private REDUCED_DECIMALS = 6;
```

CVF-18. FIXED

- **Category** Suboptimal
- **Source** PoolMath.sol

Recommendation Using signed types is redundant. The only line where negative value could appear is this one, but taking into account the logic inside "_determineZapSwapAmount", C is guaranteed to be negative. So, just calculate -C instead as an unsigned integer and use it appropriately: uint256 C = r0 * (r1 * z0 - r0 * z1); uint256 discriminant = B * B + 4 * A * C;

```
182 int256 C = r0 * ( r0 * z1 - r1 * z0 );
```

CVF-19. FIXED

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

Description In case of none-existing pool, this function returns zero, which is a valid pool index. This could make error investigations harder.

Recommendation Consider reverting in such a case.

Client Comment *A constant INVALID_POOL_ID value of uint64.max is now used.*

```
61 function _poolIndex( IERC20 tokenA, IERC20 tokenB, bytes32[] memory
   ↪ poolIDs ) internal pure returns (uint64 index)
```

CVF-23. FIXED

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

Description This shouldn't be calculated unless isWhitelisted(poolID1) is false.

```
150 bytes32 poolID2 = PoolUtils._poolIDOnly( token, weth );
```

7 Moderate Issues

CVF-1. INFO

- **Category** Flaw
- **Source** PriceAggregator.sol

Description In case one price is zero, it is possible that this zero price could participate in the smallest price diff, which could lead to a weird result.

Recommendation Consider the following values: price1 = 0, price2 = 2, price3 = 4. Then, priceA will be 0 and priceB will be 2. Consider never returning a zero price.

Client Comment *It is not possible for a zero price to mistakingly be used by the price aggregator in resulting aggregated price.*
The price aggregator requires that the two closest prices are within a default 3% of each other (bound between 1% and 7%) for the price to be valid.
Under that restriction there is no possibility that valid priceB and priceC values will be closer to zero than to each other.
*The only way zero could be closer to priceB or priceC would be if priceC is at least 2 * priceB (or visa versa). That then is well beyond the default 3% requirement.*
Under that situation the price aggregator revers with an invalid price - as it should.

```
132 if ( ( diff12 <= diff13 ) && ( diff12 <= diff23 ) )
        (priceA, priceB) = (price1, price2);
    else if ( ( diff13 <= diff12 ) && ( diff13 <= diff23 ) )
        (priceA, priceB) = (price1, price3);
    else if ( ( diff23 <= diff12 ) && ( diff23 <= diff13 ) )
        (priceA, priceB) = (price2, price3);
```

CVF-2. INFO

- **Category** Suboptimal
- **Source** IStakingRewards.sol

Description This function is identical to the function declared in the “IRewardsEmitter” interface.

Recommendation Consider refactoring to avoid such duplications.

```
22 function addSALTRewards( AddedReward[] calldata addedRewards )
     ↪ external;
```

CVF-5. INFO

- **Category** Suboptimal
- **Source** StakingConfig.sol

Description These events are emitted even if nothing actually changed.

Client Comment *Event emission when parameters are already at the edge of the allowed range is acceptable.*

```
47 emit MinUnstakeWeeksChanged(minUnstakeWeeks);
64 emit MaxUnstakeWeeksChanged(maxUnstakeWeeks);
81 emit MinUnstakePercentChanged(minUnstakePercent);
98 emit ModificationCooldownChanged(modificationCooldown);
```

CVF-6. INFO

- **Category** Suboptimal
- **Source** StakingConfig.sol

Description Here values, just written into the storage, are read back.

Recommendation Consider reusing the written values.

Client Comment *Acceptable considering infrequency of the call.*

```
47 emit MinUnstakeWeeksChanged(minUnstakeWeeks);
64 emit MaxUnstakeWeeksChanged(maxUnstakeWeeks);
81 emit MinUnstakePercentChanged(minUnstakePercent);
98 emit ModificationCooldownChanged(modificationCooldown);
```

CVF-7. INFO

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

Description These events are emitted even if nothing actually changed.

Client Comment *Event emission when parameters are already at the edge of the allowed range is acceptable.*

```
63 emit RewardPercentForCallingLiquidationChanged(
   ↪ rewardPercentForCallingLiquidation);

80 emit MaxRewardValueForCallingLiquidationChanged(
   ↪ maxRewardValueForCallingLiquidation);

97 emit MinimumCollateralValueForBorrowingChanged(
   ↪ minimumCollateralValueForBorrowing);

114 emit InitialCollateralRatioPercentChanged(
   ↪ initialCollateralRatioPercent);

134 emit MinimumCollateralRatioPercentChanged(
   ↪ minimumCollateralRatioPercent);

151 emit PercentArbitrageProfitsForStablePOLChanged(
   ↪ percentArbitrageProfitsForStablePOL);
```

CVF-8. INFO

- **Category** Suboptimal
- **Source** StableConfig.sol

Description Here a value just written into the storage is read back.

Recommendation Consider reusing the written value.

Client Comment *Acceptable considering infrequency of the call.*

```
63 emit RewardPercentForCallingLiquidationChanged(
      ↪ rewardPercentForCallingLiquidation);
```

```
80 emit MaxRewardValueForCallingLiquidationChanged(
      ↪ maxRewardValueForCallingLiquidation);
```

```
97 emit MinimumCollateralValueForBorrowingChanged(
      ↪ minimumCollateralValueForBorrowing);
```

```
114 emit InitialCollateralRatioPercentChanged(
       ↪ initialCollateralRatioPercent);
```

```
134 emit MinimumCollateralRatioPercentChanged(
       ↪ minimumCollateralRatioPercent);
```

```
151 emit PercentArbitrageProfitsForStablePOLChanged(
       ↪ percentArbitrageProfitsForStablePOL);
```

CVF-9. INFO

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

Description Usually, such functionality is implemented off-chain, by analyzing contract's event log and storage. Having it on-chain seems weird.

Recommendation Consider removing this functionality from smart contract to reduce its size.

Client Comment *Off-chain liquidatable user search can be added later as well if deemed more efficient.*

```
314 function findLiquidatableUsers( uint256 startIndex, uint256 endIndex
    ↪  ) public view returns (address[] memory)
```

```
348 function findLiquidatableUsers() external view returns (address[]
    ↪ memory)
```

CVF-10. INFO

- **Category** Suboptimal
- **Source** CollateralAndLiquidity.sol

Recommendation It is possible to reduce the length of an in-memory array without allocating a new array. Just overwrite the length via an assembly block.

```
340 address[] memory resizedLiquidatableUsers = new address[](count);
    for ( uint256 i = 0; i < count; i++ )
            resizedLiquidatableUsers[i] = liquidatableUsers[i];
```

CVF-11. INFO

- **Category** Suboptimal
- **Source** IRewardsEmitter.sol

Description This function is identical to the function declared in the "IStakingRewards" interface.

Recommendation Consider refactoring to avoid such duplications.

```
9 function addSALTRewards( AddedReward[] calldata addedRewards )
    ↪ external;
```

CVF-12. INFO

- **Category** Suboptimal
- **Source** RewardsConfig.sol

Description These events are emitted even if nothing actually changed.

Client Comment *Event emission when parameters are already at the edge of the allowed range is acceptable.*

```
49 emit RewardsEmitterDailyPercentChanged(
      ↪ rewardsEmitterDailyPercentTimes1000);
```

```
65 emit EmissionsWeeklyPercentChanged(emissionsWeeklyPercentTimes1000);
```

```
82 emit StakingRewardsPercentChanged(stakingRewardsPercent);
```

```
99 emit PercentRewardsSaltUSDSChanged(percentRewardsSaltUSDS);
```

CVF-13. INFO

- **Category** Suboptimal
- **Source** RewardsConfig.sol

Description Here values, just written into the storage are read back again.

Recommendation Consider reusing the written values.

Client Comment *Acceptable considering infrequency of the call.*

```
49 emit RewardsEmitterDailyPercentChanged(
      ↪ rewardsEmitterDailyPercentTimes1000);
```

```
65 emit EmissionsWeeklyPercentChanged(emissionsWeeklyPercentTimes1000);
```

```
82 emit StakingRewardsPercentChanged(stakingRewardsPercent);
```

```
99 emit PercentRewardsSaltUSDSChanged(percentRewardsSaltUSDS);
```

CVF-20. INFO

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

Description The event is emitted even if the pool wasn't whitelisted.

Client Comment *Only the DAO can unwhitelist, and proposing a token unwhitelisting within the DAO first checks that the token is whitelisted (see Proposals.proposeTokenUnwhitelisting) - meaning this function should only be called with unwhitelisted tokens.*

```
73 emit PoolUnwhitelisted(address(tokenA), address(tokenB));
```

CVF-21. INFO

- **Category** Suboptimal
- **Source** PoolsConfig.sol

Description These events are emitted even if nothing actually changed.

Client Comment *Event emission when parameters are already at the edge of the allowed range is acceptable.*

```
90 emit MaximumWhitelistedPoolsChanged(maximumWhitelistedPools);
```

```
107 emit maximumInternalSwapPercentTimes1000Changed(
      ↪ maximumInternalSwapPercentTimes1000);
```

CVF-22. INFO

- **Category** Suboptimal
- **Source** PoolsConfig.sol

Description Here the values just written into the storage are read back again.

Recommendation Consider reusing the written values.

Client Comment *Acceptable considering infrequency of the call.*

```
90 emit MaximumWhitelistedPoolsChanged(maximumWhitelistedPools);
```

```
107 emit maximumInternalSwapPercentTimes1000Changed(
      ↪ maximumInternalSwapPercentTimes1000);
```

CVF-24. INFO

- **Category** Suboptimal
- **Source** PoolUtils.sol

Description This is not a value, but rather a token amount. The real value depends on the token price, and for extremely expensive tokens the value could be significant. Also, while most of the tokens use 18 decimals, there are very popular tokens with much less decimal places, such as USDT.

Recommendation Consider using different dust thresholds for different tokens or not using the dust concept at all.

Client Comment *The limitation is acceptable. With USDT's 6 decimals DUST has a value of $0.0001. With WBTC's 8 decimals, a BTC price of $1 million would yield a DUST value of $1 - which is also acceptable.*

```
11 // Token reserves less than dust are treated as if they don't exist
      ↪ at all.
   // With the 18 decimals that are used for most tokens, DUST has a
      ↪ value of 0.0000000000000001
   uint256 constant public DUST = 100;
```

CVF-25. INFO

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

Description Phantom overflow is possible here.

Recommendation Consider using the "mulDiv" function.

Client Comment *Pool reserves are stored as uint128, so the multiplication won't overflow as maximumInternalSwapPercentTimes1000 has a maximum allowable value of 2000.*

```
61 uint256 maxAmountIn = reservesIn *
      ↪ maximumInternalSwapPercentTimes1000 / (100000);
```

CVF-26. INFO

- **Category** Suboptimal
- **Source** Parameters.sol

Description A linear sequence of conditional operator is inefficient (O(n) complexity).

Recommendation Consider implementing a binary decision tree (O(ln n) complexity).

Client Comment *Acceptable considering the infrequency of the call.*

```
60 if ( parameterType == ParameterTypes.maximumWhitelistedPools )

62 else if ( parameterType == ParameterTypes.
      ↪ maximumInternalSwapPercentTimes1000 )

66 else if ( parameterType == ParameterTypes.minUnstakeWeeks )

68 else if ( parameterType == ParameterTypes.maxUnstakeWeeks )

70 else if ( parameterType == ParameterTypes.minUnstakePercent )

72 else if ( parameterType == ParameterTypes.modificationCooldown )

76 else if ( parameterType == ParameterTypes.
      ↪ rewardsEmitterDailyPercentTimes1000 )

78 else if ( parameterType == ParameterTypes.
      ↪ emissionsWeeklyPercentTimes1000 )

80 else if ( parameterType == ParameterTypes.stakingRewardsPercent )

82 else if ( parameterType == ParameterTypes.percentRewardsSaltUSDS )

86 else if ( parameterType == ParameterTypes.
      ↪ rewardPercentForCallingLiquidation )

88 else if ( parameterType == ParameterTypes.
      ↪ maxRewardValueForCallingLiquidation )

90 else if ( parameterType == ParameterTypes.
      ↪ minimumCollateralValueForBorrowing )
```

(92, 94, 96, 100, 102, 104, 106, 108, 110, 112, 114, 118, 120)

CVF-27. INFO

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

Description In case of unknown ballot type this function silently does nothing.

Recommendation Consider reverting in such a case.

Client Comment *Acceptable as a NOOP so that the finalized ballot can be removed from the queue. Reversion would cause incorrect ballots to remain stuck.*

166
```
function _executeApproval( Ballot memory ballot ) internal
```

CVF-28. INFO

- **Category** Suboptimal
- **Source** Proposals.sol

Description This check is performed even when ”ballotName” already ends with ”_confirm”, which doesn’t make much sense.

Recommendation Consider not performing the check in such a case.

Client Comment *Ballots cannot end in ”_confirm” unless they are created in createConfirmationProposal. This check is important to prevent duplication of a ballot that is already being confirmed.*

108
```
require( openBallotsByName[ string.concat(ballotName, "_confirm")]
    ↪ == 0, "Cannot␣create␣a␣proposal␣for␣a␣ballot␣with␣a␣secondary␣
    ↪ confirmation" );
```

CVF-29. INFO

- **Category** Suboptimal
- **Source** Proposals.sol

Description This check doesn't make much sense, as total supply of a token may change over time.

Recommendation Consider removing the check.

Client Comment *While the total supply may change over time it still provides some measure of protection for tokens that already have an excessively large supply.*

```
170 require( token.totalSupply() < type(uint112).max, "Token␣supply␣
    ↪ cannot␣exceed␣uint112.max" ); // 5 quadrillion max supply with
    ↪  18 decimals of precision
```

CVF-30. INFO

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

Description Using the denominator 100 leads to quite coarse precision. It allows specifying reward of 5% or 6%, but not 5.5%.

Recommendation Consider using bigger denominator, such as 10000 (for basis points) or 1e18.

Client Comment *For upkeepRewardPercent, 1% to 10% with an adjustment of 1% is acceptable.*
For percentArbitrageProfitsForStablePOL, 1% to 10% with an adjustment of 1% is acceptable.
For arbitrageProfitsPercentPOL, 15% to 45% with an adjustment of 5% is acceptable.

```
130 uint256 rewardAmount = withdrawnAmount * daoConfig.
    ↪ upkeepRewardPercent() / 100;
```

```
163 uint256 amountOfWETH = wethBalance * stableConfig.
    ↪ percentArbitrageProfitsForStablePOL() / 100;
```

```
176 uint256 amountOfWETH = wethBalance * daoConfig.
    ↪ arbitrageProfitsPercentPOL() / 100;
```

```
300 return daoWETH * daoConfig.upkeepRewardPercent() / 100;
```

CVF-31. INFO

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

Description Overflows and underflows are possible inside these unchecked block.

Recommendation Consider either using checked math or clearly explaining for each operation why it is safe even within unchecked block.

Client Comment *The code can safely be made unchecked as the functionality for the found bestArbAmountIn is duplicated exactly in Pools._arbitrage with overflow checks kept in place.*
If any overflows occur as a result of the calculations in the unchecked ArbitrageSearch code they will revert in the Pools._arbitrage code.

```
67 unchecked
```

```
105 unchecked
```

CVF-32. INFO

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

Description Overflow is possible when converting to ”uint128”.

Recommendation Consider using safe conversion.

Client Comment *Acceptable as the virtualRewards token is SALT and the maximum supply of SALT is 100 million. virtualRewards can never exceed that.*

```
87         user.virtualRewards += uint128(virtualRewardsToAdd);
totalRewards[poolID] += uint128(virtualRewardsToAdd);
```

```
92 user.userShare += uint128(increaseShareAmount);
```

CVF-33. INFO

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

Description Overflow is possible when converting to ”uint128”.

Recommendation Consider using safe conversion.

Client Comment *Acceptable as the virtualRewards token is SALT and the maximum supply of SALT is 100 million. virtualRewards can never exceed that.*

```
129 user.userShare -= uint128(decreaseShareAmount);
130 user.virtualRewards -= uint128(virtualRewardsToRemove);
```

CVF-34. INFO

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

Description Overflow is possible when converting to ”uint56” and then to ”int56”.

Recommendation Consider using safe conversion.

Client Comment *Acceptable as the virtualRewards token is SALT and the maximum supply of SALT is 100 million. virtualRewards can never exceed that.*
Liquidity is measured as amountToken1 + amountToken2 and practically will not exceed uint128.

```
67 int24 tick = int24((tickCumulatives[1] - tickCumulatives[0]) / int56
   ↪ (uint56(twapInterval)));
```

CVF-35. INFO

- **Category** Documentation
- **Source** PriceAggregator.sol

Description The check can be bypassed if '_priceFeed1' is 0.

Recommendation Consider documenting this feature or disable it.

Client Comment *It's only valid to set the initial feeds if they have never been set before. This is treated as a sanity check to make sure that the price feeds can't all be set at once.*

```
37 function setInitialFeeds( IPriceFeed _priceFeed1, IPriceFeed
      ↪ _priceFeed2, IPriceFeed _priceFeed3 ) public onlyOwner
```

```
39         require( address(priceFeed1) == address(0), "setInitialFeeds
              ↪ ()␣can␣only␣be␣called␣once" );
```

CVF-36. INFO

- **Category** Flaw
- **Source** PriceAggregator.sol

Description In case of an invalid "priceFeedNum" value, this function doesn't set any price feeds, but still updates the "setPriceFeedCooldownExpiration" variable and emits the "PriceFeedSet" event.

Recommendation Consider reverting in such a case.

Client Comment *Practically speaking, this method can only be called with the valid indicies as it is only callable by DAO._executeSetContract() can the correct arguments are made in those calls.*

```
47 function setPriceFeed( uint256 priceFeedNum, IPriceFeed newPriceFeed
      ↪  ) public onlyOwner
```

```
60         setPriceFeedCooldownExpiration = block.timestamp +
              ↪ setPriceFeedCooldown;
           emit PriceFeedSet(priceFeedNum, address(newPriceFeed));
```

CVF-37. INFO

- **Category** Documentation
- **Source** Pools.sol

Recommendation Situation when one of the reserves is less than dust, but the other isn't should either be handled separately, of there should be a clear explanation, why it could be treated as an empty pool.

Client Comment *Reserves can only be less than DUST when no liquidity has been previously added. Afterwards swaps and remove liquidity both revert if either of the resulting reserves are less than DUST. As such, the reserves can simply be checked to see if either is zero (rather than less than DUST).*

```
99  // If either reserve is less than dust then consider the pool to be
       ↪ empty and that the added liquidity will become the initial
       ↪ token ratio
100 if ( ( reserve0 < PoolUtils.DUST ) || ( reserve1 < PoolUtils.DUST )
       ↪ )
```

CVF-38. INFO

- **Category** Flaw
- **Source** PoolsConfig.sol

Description There is no check to ensure the pool isn't already whitelisted.

Recommendation Consider adding such a check.

Client Comment *Only the DAO can call whitelistPool, and Proposals.proposeTokenWhitelisting first checks to make sure the token is not already whitelisted.*

```
53  _whitelist.add(poolID);
```

CVF-39. INFO

- **Category** Procedural
- **Source** BootstrapBallot.sol

Description The signed message doesn't include the contract address as well as the purpose of the signed message. This could make replay attack possible.

Recommendation Consider properly implementing ERC-712.

Client Comment *Acceptable as the contract address is immutable and there is only one signed function used by the contract.*

```
53 bytes32 messageHash = keccak256(abi.encodePacked(block.chainid, msg.
   ↪ sender));
```

CVF-40. INFO

- **Category** Flaw
- **Source** Parameters.sol

Description For an unknown parameter type this function silently does nothing.

Recommendation Consider reverting in such a case.

Client Comment *Acceptable as a NOOP so that the finalized ballot can be removed from the queue. Reversion would cause incorrect ballots to remain stuck.*

```
57 function _executeParameterChange( ParameterTypes parameterType, bool
   ↪  increase, IPoolsConfig poolsConfig, IStakingConfig
   ↪ stakingConfig, IRewardsConfig rewardsConfig, IStableConfig
   ↪ stableConfig, IDAOConfig daoConfig, IPriceAggregator
   ↪ priceAggregator ) internal
```

CVF-41. FIXED

- **Category** Suboptimal
- **Source** Proposals.sol

Description This value should be rounded up, rather than down. Currently, even non-zero "totalStaked" could lead to zero "requiredXSalt".

Client Comment *Non-zero check now made on requiredXSalt rather than totalStaked.*

```
97 uint256 requiredXSalt = ( totalStaked * daoConfig.
   ↪ requiredProposalPercentStakeTimes1000() ) / ( 100 * 1000 );
```

8 Minor Issues

CVF-43. INFO

- **Category** Procedural
- **Source** ArbitrageSearch.sol

Description Specifying a particular compiler version makes it harder migrating to newer versions.

Recommendation Consider specifying as "^0.8.0". Also relevant for:IStakingRewards.sol, ILiquidity.sol, IStaking.sol, IStakingConfig.sol, StakingRewards.sol, Liquidity.sol, Staking.sol, StakingConfig.sol, ILiquidizer.sol, ICollateralAndLiquidity.sol, IUSDS.sol, IStableConfig.sol, Liquidizer.sol, StableConfig.sol, USDS.sol, CollateralAndLiquidity.sol, IRewardsEmitter.sol, ISaltRewards.sol, IEmissions.sol, IRewardsConfig.sol, Emissions.sol, RewardsConfig.sol, RewardsEmitter.sol, SaltRewards.sol, IPriceFeed.sol, IPriceAggregator.sol, CoreChainlinkFeed.sol, CoreSaltyFeed.sol, CoreUniswapFeed.sol, PriceAggregator.sol, IPoolStats.sol, IPools.sol, IPoolsConfig.sol, PoolMath.sol, PoolStats.sol, Pools.sol, PoolsConfig.sol, PoolUtils.sol, IBootstrapBallot.sol, IInitialDistribution.sol, IAirdrop.sol, Airdrop.sol, BootstrapBallot.sol, InitialDistribution.sol, IAccessManager.sol, ISalt.sol, IUpkeep.sol, IManagedWallet.sol, IExchangeConfig.sol, ICalledContract.sol, IDAO.sol, IDAOConfig.sol, IProposals.sol, Parameters.sol, DAO.sol, DAOConfig.sol, Upkeep.sol, SigningTools.sol, AccessManager.sol, ExchangeConfig.sol.

Client Comment *Exact compiler version specified as an indication that this is what was actually extensively tested - in the event that future versions introduce unforeseen inconsistencies.*

```
2 pragma solidity =0.8.22;
```

CVF-44. FIXED

- **Category** Readability
- **Source** ArbitrageSearch.sol

Recommendation This value could be rendered as "0.001e18".

```
16 uint256 constant public MIDPOINT_PRECISION = 10**15; // .001 ETH
     ↪ precision for arb search
```

CVF-45. INFO

- **Category** Suboptimal
- **Source** ArbitrageSearch.sol

Description Here the same contract is called three times in a row, which is inefficient.

Recommendation Consider merging these three calls into a single call.

Client Comment *Acceptable as it's only done once in the constructor.*

```
24 wbtc = _exchangeConfig.wbtc();
   weth = _exchangeConfig.weth();
   salt = _exchangeConfig.salt();
```

CVF-46. INFO

- **Category** Suboptimal
- **Source** ArbitrageSearch.sol

Recommendation if (swapTokenOut == weth) { if (swapTokenIn == wbtc) return (wbtc, salt); else return (swapTokenIn, wbtc); }

Client Comment *Kept as is for clarity.*

```
37 if ( address(swapTokenIn) == address(wbtc))
   if ( address(swapTokenOut) == address(weth))
           return (wbtc, salt);
```

```
54 if ( address(swapTokenOut) == address(weth))
           return (swapTokenIn, wbtc);
```

CVF-47. INFO

- **Category** Suboptimal
- **Source** ArbitrageSearch.sol

Recommendation This could be optimized and simplified as: if (swapTokenIn == weth) { if (swapTokenOut == wbtc) return (salt, wbtc); else return (wbtc, swapTokenOut); }

Client Comment *Kept as is for clarity.*

```
43 if ( address(swapTokenIn) == address(weth))
   if ( address(swapTokenOut) == address(wbtc))
           return (salt, wbtc);
```

```
49 if ( address(swapTokenIn) == address(weth))
50         return (wbtc, swapTokenOut);
```

CVF-48. INFO

- **Category** Procedural
- **Source** ArbitrageSearch.sol

Recommendation Brackets around multiplication are redundant.

Client Comment *Kept as is for clarity.*

```
70 uint256 amountOut = (reservesA1 * midpoint) / (reservesA0 + midpoint
      ↪ );
   amountOut = (reservesB1 * amountOut) / (reservesB0 + amountOut);
   amountOut = (reservesC1 * amountOut) / (reservesC0 + amountOut);
```

```
84 amountOut = (reservesA1 * midpoint) / (reservesA0 + midpoint);
   amountOut = (reservesB1 * amountOut) / (reservesB0 + amountOut);
   amountOut = (reservesC1 * amountOut) / (reservesC0 + amountOut);
```

```
129 uint256 amountOut = (reservesA1 * bestArbAmountIn) / (reservesA0 +
       ↪ bestArbAmountIn);
130 amountOut = (reservesB1 * amountOut) / (reservesB0 + amountOut);
    amountOut = (reservesC1 * amountOut) / (reservesC0 + amountOut);
```

CVF-49. FIXED

- **Category** Documentation
- **Source** ArbitrageSearch.sol

Description This is not a binary search, but rather a bisection method.

```
95 // Perform a modified binary search to search for the
   ↪ bestArbAmountIn in a range of 1/128th to 125% of
   ↪ swapAmountInValueInETH.
```

CVF-50. INFO

- **Category** Procedural
- **Source** IStakingRewards.sol

Description Defining a top-level structure in a file named after an interface makes it harder navigating through code.

Recommendation Consider either moving the structure definition into the interface or moving it into a separate file.

```
5 struct AddedReward
```

```
11 struct UserShareInfo
```

CVF-51. INFO

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

Recommendation This function should emit some event and this event should be declared in this interface.

Client Comment *SaltRewardsAdded is already emitted for each pool receiving rewards.*

```
22 function addSALTRewards( AddedReward[] calldata addedRewards )
   ↪ external;
```

CVF-52. INFO

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

Recommendation These function should emit some events and these events should be declared in this interface.

Client Comment *LiquidityDeposited and LiquidityWithdrawn are already emitted.*

```
10 function depositLiquidityAndIncreaseShare( IERC20 tokenA, IERC20
   ↪ tokenB, uint256 maxAmountA, uint256 maxAmountB, uint256
   ↪ minLiquidityReceived, uint256 deadline, bool useZapping )
   ↪ external returns (uint256 addedAmountA, uint256 addedAmountB,
   ↪ uint256 addedLiquidity);
function withdrawLiquidityAndClaim( IERC20 tokenA, IERC20 tokenB,
   ↪ uint256 liquidityToWithdraw, uint256 minReclaimedA, uint256
   ↪ minReclaimedB, uint256 deadline ) external returns (uint256
   ↪ reclaimedA, uint256 reclaimedB);
```

CVF-53. INFO

- **Category** Procedural
- **Source** IStaking.sol

Description Defining top-level types in a file named after an interface makes it harder to navigate through code.

Recommendation Consider either moving the type definitions into the interface or moving them into a separate file.

```
12 enum UnstakeState { NONE, PENDING, CANCELLED, CLAIMED }
```

```
14  struct Unstake
```

CVF-54. INFO

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *Events are already emitted.*

```
27 function stakeSALT( uint256 amountToStake ) external;
   function unstake( uint256 amountUnstaked, uint256 numWeeks )
      ↪ external returns (uint256 unstakeID);
   function cancelUnstake( uint256 unstakeID ) external;
30 function recoverSALT( uint256 unstakeID ) external;
   function transferStakedSaltFromAirdropToUser(address wallet, uint256
      ↪  amount) external;
```

CVF-55. INFO

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *Events are already emitted.*

```
 7 function changeMinUnstakeWeeks(bool increase) external; // onlyOwner
   function changeMaxUnstakeWeeks(bool increase) external; // onlyOwner
   function changeMinUnstakePercent(bool increase) external; //
      ↪ onlyOwner
10 function changeModificationCooldown(bool increase) external; //
      ↪ onlyOwner
```

CVF-56. INFO

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

Recommendation Events are usually named via nouns, such as ”UserShareIncrease” or ”RewardsClaim”.

```
23 event UserShareIncreased(address indexed wallet, bytes32 indexed
      ↪ poolID, uint256 amountIncreased);
   event UserShareDecreased(address indexed wallet, bytes32 indexed
      ↪ poolID, uint256 amountDecreased, uint256 claimedRewards);
   event RewardsClaimed(address indexed wallet, uint256 claimedRewards)
      ↪ ;
   event SaltRewardsAdded(bytes32 indexed poolID, uint256 amountAdded);
```

CVF-57. FIXED

- **Category** Suboptimal
- **Source** StakingRewards.sol

Description These checks are redundant, as dead addresses could anyway be passed.

Recommendation Consider removing these checks.

```
48 require( address(_exchangeConfig) != address(0), "_exchangeConfig␣
      ↪ cannot␣be␣address(0)" );
   require( address(_poolsConfig) != address(0), "_poolsConfig␣cannot␣
      ↪ be␣address(0)" );
50 require( address(_stakingConfig) != address(0), "_stakingConfig␣
      ↪ cannot␣be␣address(0)" );
```

CVF-58. INFO

- **Category** Procedural
- **Source** StakingRewards.sol

Recommendation Brackets around multiplications are redundant.

Client Comment *Kept as is for clarity.*

```
118 uint256 rewardsForAmount = ( totalRewards[poolID] *
    ↪ decreaseShareAmount ) / totalShares[poolID];
```

```
122 uint256 virtualRewardsToRemove = (user.virtualRewards *
    ↪ decreaseShareAmount) / user.userShare;
```

```
243 uint256 rewardsShare = ( totalRewards[poolID] * user.userShare ) /
    ↪ totalShares[poolID];
```

CVF-59. INFO

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

Recommendation Events are usually named via nouns, such as ”LiquidityDeposit” or ”LiquidityWithdrawal”.

```
20 event LiquidityDeposited(address indexed user, address indexed
    ↪ tokenA, address indexed tokenB, uint256 amountA, uint256
    ↪ amountB, uint256 addedLiquidity);
   event LiquidityWithdrawn(address indexed user, address indexed
    ↪ tokenA, address indexed tokenB, uint256 amountA, uint256
    ↪ amountB, uint256 withdrawnLiquidity);
```

CVF-60. FIXED

- **Category** Suboptimal
- **Source** Liquidity.sol

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

Recommendation Consider removing this check.

```
34 require( address(_pools) != address(0), "_pools␣cannot␣be␣address(0)
    ↪ " );
```

CVF-61. FIXED

- **Category** Readability
- **Source** Liquidity.sol

Recommendation Should be ”else if”.

```
68 if ( swapAmountB > 0)
```

CVF-62. INFO

- **Category** Procedural
- **Source** Liquidity.sol

Description Here the arguments ”maxAmountA” and ”maxAmountB” are used as variables, which is a bad practice that makes code harder to read.

Recommendation Consider using local variables instead.

```
99 (maxAmountA, maxAmountB) = _dualZapInLiquidity(tokenA, tokenB,
      ↪ maxAmountA, maxAmountB );
```

CVF-63. INFO

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

Recommendation Events are usually named via nouns, such as ”SALTStake” or ”UnstakeInitiation”.

```
17 event SALTStaked(address indexed user, uint256 amountStaked);
   event UnstakeInitiated(address indexed user, uint256 indexed
      ↪ unstakeID, uint256 amountUnstaked, uint256 claimableSALT,
      ↪ uint256 numWeeks);
   event UnstakeCancelled(address indexed user, uint256 indexed
      ↪ unstakeID);
20 event SALTRecovered(address indexed user, uint256 indexed unstakeID,
      ↪  uint256 saltRecovered, uint256 expeditedUnstakeFee);
   event XSALTTransferredFromAirdrop(address indexed toUser, uint256
      ↪ amountTransferred);
```

CVF-64. INFO

- **Category** Procedural
- **Source** Staking.sol

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

35
```
{
}
```

CVF-65. INFO

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

Recommendation The value ”100” should be a named constant.

207
```
        uint256 percentAboveMinimum = 100 - minUnstakePercent;
```

211
```
return numerator / ( 100 * unstakeRange );
```

CVF-66. INFO

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

Recommendation Events are usually named via nouns, such as ”MinUnstakeWeeks” or ”MinUnstakePercent”.

11
```
event MinUnstakeWeeksChanged(uint256 newMinUnstakeWeeks);
event MaxUnstakeWeeksChanged(uint256 newMaxUnstakeWeeks);
event MinUnstakePercentChanged(uint256 newMinUnstakePercent);
event ModificationCooldownChanged(uint256 newModificationCooldown);
```

CVF-67. INFO

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

Recommendation The default values should be named constants.

```
18 uint256 public minUnstakeWeeks = 2;  // minUnstakePercent returned
     ↪ for unstaking this number of weeks

22 uint256 public maxUnstakeWeeks = 52;

26 uint256 public minUnstakePercent = 20;

31 uint256 public modificationCooldown = 1 hours;
```

CVF-68. INFO

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

Recommendation These values should be named constants.

```
38 if (minUnstakeWeeks < 12)
       minUnstakeWeeks += 1;

43 if (minUnstakeWeeks > 1)
       minUnstakeWeeks -= 1;

55 if (maxUnstakeWeeks < 108)
       maxUnstakeWeeks += 8;

60 if (maxUnstakeWeeks > 20)
       maxUnstakeWeeks -= 8;

72 if (minUnstakePercent < 50)
       minUnstakePercent += 5;

77 if (minUnstakePercent > 10)
       minUnstakePercent -= 5;

89 if (modificationCooldown < 6 hours)
90     modificationCooldown += 15 minutes;

94 if (modificationCooldown > 15 minutes)
       modificationCooldown -= 15 minutes;
```

CVF-69. INFO

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

Recommendation This function should emit some event and this event should be declared in this interface.

```
11 function setContracts(ICollateralAndLiquidity
   ↪ _collateralAndLiquidity, IPools _pools, IDAO _dao) external;
   ↪ // onlyOwner
```

CVF-70. FIXED

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

Description The name looks like a “view” function, which is not the case.

Recommendation Consider renaming.

```
12 function shouldBurnMoreUSDS( uint256 usdsToBurn ) external;
```

CVF-71. INFO

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *Events are already emitted.*

```
10 function depositCollateralAndIncreaseShare( uint256 maxAmountWBTC,
   ↪ uint256 maxAmountWETH, uint256 minLiquidityReceived, uint256
   ↪ deadline, bool useZapping ) external returns (uint256
   ↪ addedAmountWBTC, uint256 addedAmountWETH, uint256
   ↪ addedLiquidity);
function withdrawCollateralAndClaim( uint256 collateralToWithdraw,
   ↪ uint256 minReclaimedWBTC, uint256 minReclaimedWETH, uint256
   ↪ deadline ) external returns (uint256 reclaimedWBTC, uint256
   ↪ reclaimedWETH);
function borrowUSDS( uint256 amountBorrowed ) external;
function repayUSDS( uint256 amountRepaid ) external;
function liquidateUser( address wallet ) external;
```

CVF-72. INFO

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

Recommendation This function should emit some event and this events should be declared in this interface.

Client Comment *There events are already emitted.*

```
10 function setCollateralAndLiquidity( ICollateralAndLiquidity
   ↪ _collateralAndLiquidity ) external; // onlyOwner
```

CVF-73. FIXED

- **Category** Procedural
- **Source** IStableConfig.sol

Description This import is not used.

Recommendation Consider removing it.

```
4 import "../../price_feed/interfaces/IPriceFeed.sol";
```

CVF-74. INFO

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *Events are already emitted.*

```
9  function changeRewardPercentForCallingLiquidation(bool increase)
       ↪ external; // onlyOwner
10 function changeMaxRewardValueForCallingLiquidation(bool increase)
       ↪ external; // onlyOwner
   function changeMinimumCollateralValueForBorrowing(bool increase)
       ↪ external; // onlyOwner
   function changeInitialCollateralRatioPercent(bool increase) external
       ↪ ; // onlyOwner
   function changeMinimumCollateralRatioPercent(bool increase) external
       ↪ ; // onlyOwner
   function changePercentArbitrageProfitsForStablePOL(bool increase)
       ↪ external; // onlyOwner
```

CVF-75. FIXED

- **Category** Suboptimal
- **Source** Liquidizer.sol

Recommendation These checks are redundant as it is anyway possible to pass dead addresses.

```
51 require( address(_exchangeConfig) != address(0), "_exchangeConfig␣
       ↪ cannot␣be␣address(0)" );
   require( address(_poolsConfig) != address(0), "_poolsConfig␣cannot␣
       ↪ be␣address(0)" );
```

CVF-76. INFO

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

Recommendation This function should emit some event.

Client Comment *Function only called once on deployment.*

```
66 function setContracts( ICollateralAndLiquidity
      ↪ _collateralAndLiquidity, IPools _pools, IDAO _dao) external
      ↪ onlyOwner
```

CVF-77. FIXED

- **Category** Suboptimal
- **Source** Liquidizer.sol

Recommendation These checks are redundant as it is anyway possible to pass dead addresses.

```
68 require( address(_collateralAndLiquidity) != address(0), "
      ↪ _collateralAndLiquidity␣cannot␣be␣address(0)" );
   require( address(_pools) != address(0), "_pools␣cannot␣be␣address(0)
      ↪ " );
70 require( address(_dao) != address(0), "_dao␣cannot␣be␣address(0)" );
```

CVF-78. FIXED

- **Category** Readability
- **Source** Liquidizer.sol

Description The code below looks like it is always executed, while actually it is executed only when ”usdsBalance” is insufficient.

Recommendation Consider putting the rest of the function into an explicit ”else” branch.

```
123 }
```

CVF-79. INFO

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

Recommendation Events are usually named via nouns, such as ”RewardPercentForCallingLiquidation” or ”MaxRewardValueForCallingLiquidation”.

```
11 event RewardPercentForCallingLiquidationChanged(uint256
       ↪ newRewardPercent);
   event MaxRewardValueForCallingLiquidationChanged(uint256
       ↪ newMaxRewardValue);
   event MinimumCollateralValueForBorrowingChanged(uint256
       ↪ newMinimumCollateralValue);
   event InitialCollateralRatioPercentChanged(uint256
       ↪ newInitialCollateralRatioPercent);
   event MinimumCollateralRatioPercentChanged(uint256
       ↪ newMinimumCollateralRatioPercent);
   event PercentArbitrageProfitsForStablePOLChanged(uint256
       ↪ newPercentArbitrageProfits);
```

CVF-80. INFO

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

Recommendation The default values should be named constants.

```
20 uint256 public rewardPercentForCallingLiquidation = 5;
```

```
24 uint256 public maxRewardValueForCallingLiquidation = 500 ether;
```

```
29 uint256 public minimumCollateralValueForBorrowing = 2500 ether;
```

```
34 uint256 public initialCollateralRatioPercent = 200;
```

```
39     uint256 public minimumCollateralRatioPercent = 110;
```

```
44     uint256 public percentArbitrageProfitsForStablePOL = 5;
```

CVF-81. INFO

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

Recommendation These values should be named constants.

```
54  if (remainingRatioAfterReward >= 105 &&
    ↪ rewardPercentForCallingLiquidation < 10)
        rewardPercentForCallingLiquidation += 1;

59  if (rewardPercentForCallingLiquidation > 5)
60      rewardPercentForCallingLiquidation -= 1;

71  if (maxRewardValueForCallingLiquidation < 1000 ether)
        maxRewardValueForCallingLiquidation += 100 ether;

76  if (maxRewardValueForCallingLiquidation > 100 ether)
        maxRewardValueForCallingLiquidation -= 100 ether;

88  if (minimumCollateralValueForBorrowing < 5000 ether)
        minimumCollateralValueForBorrowing += 500 ether;

93  if (minimumCollateralValueForBorrowing > 1000 ether)
        minimumCollateralValueForBorrowing -= 500 ether;

105 if (initialCollateralRatioPercent < 300)
        initialCollateralRatioPercent += 25;

110 if (initialCollateralRatioPercent > 150)
        initialCollateralRatioPercent -= 25;

122 if (minimumCollateralRatioPercent < 120)
        minimumCollateralRatioPercent += 1;

130 if (remainingRatioAfterReward >= 105 &&
    ↪ minimumCollateralRatioPercent > 110)
        minimumCollateralRatioPercent -= 1;

142 if (percentArbitrageProfitsForStablePOL < 10)
        percentArbitrageProfitsForStablePOL += 1;

147 if (percentArbitrageProfitsForStablePOL > 1)
        percentArbitrageProfitsForStablePOL -= 1;
```

CVF-82. INFO

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

Recommendation Events are usually named via nouns, such as ”Mint” or ”Burn”.

```
16 event USDSMinted(address indexed to, uint256 amount);
   event USDSTokensBurned(uint256 amount);
```

CVF-83. INFO

- **Category** Procedural
- **Source** USDS.sol

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

```
24         {
   }
```

CVF-84. INFO

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

Recommendation This function should emit some event.

Client Comment *Only called on deployment and not callable afterwards.*

```
29 function setCollateralAndLiquidity( ICollateralAndLiquidity
      ↪ _collateralAndLiquidity ) external onlyOwner
```

CVF-85. INFO

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

Recommendation Events are usually named via nouns, such as ”CollateralDeposit” or ”CollateralWithdrawal”.

```
23 event CollateralDeposited(address indexed depositor, uint256
      ↪ amountWBTC, uint256 amountWETH, uint256 liquidity);
   event CollateralWithdrawn(address indexed withdrawer, uint256
      ↪ collateralWithdrawn, uint256 reclaimedWBTC, uint256
      ↪ reclaimedWETH);
```

CVF-86. FIXED

- **Category** Suboptimal
- **Source** CollateralAndLiquidity.sol

Recommendation These checks are redundant as dead addresses could anyway be passed.

```
54 require( address(_stableConfig) != address(0), "_stableConfig␣cannot
      ↪ ␣be␣address(0)" );
   require( address(_priceAggregator) != address(0), "_priceAggregator␣
      ↪ cannot␣be␣address(0)" );
   require( address(_liquidizer) != address(0), "_liquidizer␣cannot␣be␣
      ↪ address(0)" );
```

CVF-87. FIXED

- **Category** Suboptimal
- **Source** CollateralAndLiquidity.sol

Description Double type conversion here looks weird.

Recommendation Consider converting to ”IERC20Metadata” instead.

```
66 wbtcDecimals = ERC20(address(wbtc)).decimals();
   wethDecimals = ERC20(address(weth)).decimals();
```

CVF-88. INFO

- **Category** Procedural
- **Source** CollateralAndLiquidity.sol

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

Recommendation Consider treating all token amounts as integers.

Client Comment *The decimals are needed to compute the value of the specified tokens. Without knowing the decimals the underlying value cannot be determined.*

```
66 wbtcDecimals = ERC20(address(wbtc)).decimals();
   wethDecimals = ERC20(address(weth)).decimals();
```

CVF-89. INFO

- **Category** Suboptimal
- **Source** CollateralAndLiquidity.sol

Recommendation Brackets around multiplication are redundant.

Client Comment *Kept as is for clarity.*

```
162 uint256 rewardedWBTC = (reclaimedWBTC * rewardPercent) / 100;
    uint256 rewardedWETH = (reclaimedWETH * rewardPercent) / 100;

170         rewardedWBTC = (rewardedWBTC * maxRewardValue) / rewardValue
              ↪ ;
            rewardedWETH = (rewardedWETH * maxRewardValue) / rewardValue
              ↪ ;

205 uint256 btcValue = ( amountBTC * btcPrice ) / (10 ** wbtcDecimals );
    uint256 ethValue = ( amountETH * ethPrice ) / (10 ** wethDecimals );

235 uint256 userWBTC = (reservesWBTC * userCollateralAmount ) /
      ↪ totalCollateralShares;
    uint256 userWETH = (reservesWETH * userCollateralAmount ) /
      ↪ totalCollateralShares;

253 uint256 requiredCollateralValueAfterWithdrawal = (
      ↪ usdsBorrowedByUsers[wallet] * stableConfig.
      ↪ initialCollateralRatioPercent() ) / 100;

310 return (( userCollateralValue * 100 ) / usdsBorrowedAmount) <
      ↪ stableConfig.minimumCollateralRatioPercent();

329                 uint256 minCollateralValue = (usdsBorrowedByUsers[
                      ↪ wallet] * stableConfig.
                      ↪ minimumCollateralRatioPercent()) / 100;

332                 uint256 minCollateral = (minCollateralValue *
                      ↪ totalCollateralShares) / totalCollateralValue;
```

CVF-90. INFO

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

Recommendation The value ”100” should be a named constant.

```
162 uint256 rewardedWBTC = (reclaimedWBTC * rewardPercent) / 100;
    uint256 rewardedWETH = (reclaimedWETH * rewardPercent) / 100;
```

```
253 uint256 requiredCollateralValueAfterWithdrawal = (
       ↪ usdsBorrowedByUsers[wallet] * stableConfig.
       ↪ initialCollateralRatioPercent() ) / 100;
```

```
283 uint256 maxBorrowableAmount = ( userCollateralValue * 100 ) /
       ↪ stableConfig.initialCollateralRatioPercent();
```

```
310 return (( userCollateralValue * 100 ) / usdsBorrowedAmount) <
       ↪ stableConfig.minimumCollateralRatioPercent();
```

```
329                 uint256 minCollateralValue = (usdsBorrowedByUsers[
                       ↪ wallet] * stableConfig.
                       ↪ minimumCollateralRatioPercent()) / 100;
```

CVF-91. INFO

- **Category** Suboptimal
- **Source** CollateralAndLiquidity.sol

Description The denominator of 100 seems to low, as it doesn't allow specifying fractions of a percent.

Recommendation Consider using higher denominator.

Client Comment *rewardPercentForCallingLiquidation: 5 to 10% with an adjustment of 1% is acceptable*
initialCollateralRatioPercent: 150 to 300% with an adjustment of 25% is acceptable
minimumCollateralRatioPercent: 110 to 120% with an adjustment of 1% is acceptable

```
162 uint256 rewardedWBTC = (reclaimedWBTC * rewardPercent) / 100;
    uint256 rewardedWETH = (reclaimedWETH * rewardPercent) / 100;
```

```
253 uint256 requiredCollateralValueAfterWithdrawal = (
      ↪ usdsBorrowedByUsers[wallet] * stableConfig.
      ↪ initialCollateralRatioPercent() ) / 100;
```

```
283 uint256 maxBorrowableAmount = ( userCollateralValue * 100 ) /
      ↪ stableConfig.initialCollateralRatioPercent();
```

```
310 return (( userCollateralValue * 100 ) / usdsBorrowedAmount) <
      ↪ stableConfig.minimumCollateralRatioPercent();
```

```
329                 uint256 minCollateralValue = (usdsBorrowedByUsers[
                      ↪ wallet] * stableConfig.
                      ↪ minimumCollateralRatioPercent()) / 100;
```

CVF-92. FIXED

- **Category** Procedural
- **Source** CollateralAndLiquidity.sol

Recommendation The values "10 ** wbtcDecimals" and "10 ** wethDecimals" should be precomputed and stored in immutable variables.

```
205 uint256 btcValue = ( amountBTC * btcPrice ) / (10 ** wbtcDecimals );
    uint256 ethValue = ( amountETH * ethPrice ) / (10 ** wethDecimals );
```

CVF-93. INFO

- **Category** Procedural
- **Source** CollateralAndLiquidity.sol

Recommendation Brackets are redundant.

Client Comment *Kept as is for clarity.*

```
310 return (( userCollateralValue * 100 ) / usdsBorrowedAmount) <
    ↪ stableConfig.minimumCollateralRatioPercent();
```

CVF-94. INFO

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

Recommendation This function should emit some event and this event should be declared in this interface.

Client Comment *Event already emitted.*

```
9 function addSALTRewards( AddedReward[] calldata addedRewards )
    ↪ external;
```

CVF-95. INFO

- **Category** Suboptimal
- **Source** ISaltRewards.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays, This would also make the length check unnecessary.

```
10 function performUpkeep( bytes32[] calldata poolIDs, uint256[]
    ↪ calldata profitsForPools ) external;
```

CVF-96. INFO

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *Events are already emitted.*

```
7  function changeRewardsEmitterDailyPercent(bool increase) external;
       ↪ // onlyOwner
   function changeEmissionsWeeklyPercent(bool increase) external; //
       ↪ onlyOwner
   function changeStakingRewardsPercent(bool increase) external; //
       ↪ onlyOwner
10 function changePercentRewardsSaltUSDS(bool increase) external; //
       ↪ onlyOwner
```

CVF-97. FIXED

- **Category** Suboptimal
- **Source** Emissions.sol

Description These checks are redundant, as dead addresses could anyway be passed.

Recommendation Consider removing these checks.

```
29 require( address(_saltRewards) != address(0), "_saltRewards␣cannot␣
       ↪ be␣address(0)" );
30 require( address(_exchangeConfig) != address(0), "_exchangeConfig␣
       ↪ cannot␣be␣address(0)" );
   require( address(_rewardsConfig) != address(0), "_rewardsConfig␣
       ↪ cannot␣be␣address(0)" );
```

CVF-98. INFO

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

Recommendation The value ”100 * 1000 weeks” should be a named constant.

```
59 uint256 saltToSend = ( saltBalance * timeSinceLastUpkeep *
       ↪ rewardsConfig.emissionsWeeklyPercentTimes1000() ) / ( 100 *
       ↪ 1000 weeks );
```

CVF-99. INFO

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

Recommendation The value ”100 * 1000” could be rendered as ”1e5” for readability.

```
59 uint256 saltToSend = ( saltBalance * timeSinceLastUpkeep *
   ↪ rewardsConfig.emissionsWeeklyPercentTimes1000() ) / ( 100 *
   ↪ 1000 weeks );
```

CVF-100. INFO

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

Recommendation Events are usually named via nouns, such as ”RewardsEmitterDailyPercent” or ”EmissionsWeeklyPercent”.

```
11 event RewardsEmitterDailyPercentChanged(uint256
   ↪ newRewardsEmitterDailyPercent);
event EmissionsWeeklyPercentChanged(uint256
   ↪ newEmissionsWeeklyPercent);
event StakingRewardsPercentChanged(uint256 newStakingRewardsPercent)
   ↪ ;
event PercentRewardsSaltUSDSChanged(uint256
   ↪ newPercentRewardsSaltUSDS);
```

CVF-101. INFO

- **Category** Suboptimal
- **Source** RewardsConfig.sol

Recommendation The prefix ”new” in parameter names is redundant, as there are no corresponding ”old” parameters.

```
11 event RewardsEmitterDailyPercentChanged(uint256
   ↪ newRewardsEmitterDailyPercent);
event EmissionsWeeklyPercentChanged(uint256
   ↪ newEmissionsWeeklyPercent);
event StakingRewardsPercentChanged(uint256 newStakingRewardsPercent)
   ↪ ;
event PercentRewardsSaltUSDSChanged(uint256
   ↪ newPercentRewardsSaltUSDS);
```

CVF-102. INFO

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

Recommendation The default values should be named constants.

```
19    uint256 public rewardsEmitterDailyPercentTimes1000 = 1000;  //
         ↪ Defaults to 1.0% with a 1000x multiplier
```

```
23    uint256 public emissionsWeeklyPercentTimes1000 = 500;  //
         ↪ Defaults to 0.50% with a 1000x multiplier
```

```
27 uint256 public stakingRewardsPercent = 50;
```

```
33 uint256 public percentRewardsSaltUSDS = 10;
```

CVF-103. INFO

- **Category** Readability
- **Source** RewardsConfig.sol

Recommendation This value could be rendered as ”0.01e5”.

```
19 uint256 public rewardsEmitterDailyPercentTimes1000 = 1000;  //
   ↪ Defaults to 1.0% with a 1000x multiplier
```

CVF-104. INFO

- **Category** Readability
- **Source** RewardsConfig.sol

Recommendation This value could be rendered as ”0.005e5”.

```
23 uint256 public emissionsWeeklyPercentTimes1000 = 500;  // Defaults
   ↪ to 0.50% with a 1000x multiplier
```

CVF-105. INFO

- **Category** Readability
- **Source** RewardsConfig.sol

Recommendation This value could be rendered as ”0.5e2”.

```
27 uint256 public stakingRewardsPercent = 50;
```

CVF-106. INFO

- **Category** Readability
- **Source** RewardsConfig.sol

Recommendation This value could be rendered as ”0.1e2”.

```
33 uint256 public percentRewardsSaltUSDS = 10;
```

CVF-107. INFO

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

Recommendation These values should be named constants.

```
40 if (rewardsEmitterDailyPercentTimes1000 < 2500)
       rewardsEmitterDailyPercentTimes1000 =
          ↪ rewardsEmitterDailyPercentTimes1000 + 250;

45 if (rewardsEmitterDailyPercentTimes1000 > 250)
       rewardsEmitterDailyPercentTimes1000 =
          ↪ rewardsEmitterDailyPercentTimes1000 - 250;

56 if (emissionsWeeklyPercentTimes1000 < 1000)
       emissionsWeeklyPercentTimes1000 =
          ↪ emissionsWeeklyPercentTimes1000 + 250;

61 if (emissionsWeeklyPercentTimes1000 > 250)
       emissionsWeeklyPercentTimes1000 =
          ↪ emissionsWeeklyPercentTimes1000 - 250;

73 if (stakingRewardsPercent < 75)
       stakingRewardsPercent = stakingRewardsPercent + 5;

78 if (stakingRewardsPercent > 25)
       stakingRewardsPercent = stakingRewardsPercent - 5;

90 if (percentRewardsSaltUSDS < 25)
       percentRewardsSaltUSDS = percentRewardsSaltUSDS + 5;

95 if (percentRewardsSaltUSDS > 5)
       percentRewardsSaltUSDS = percentRewardsSaltUSDS - 5;
```

CVF-108. FIXED

- **Category** Suboptimal
- **Source** RewardsEmitter.sol

Recommendation These checks are redundant as dead addresses could anyway be passed.

```
41 require( address(_stakingRewards) != address(0), "_stakingRewards␣
      ↪ cannot␣be␣address(0)" );
   require( address(_exchangeConfig) != address(0), "_exchangeConfig␣
      ↪ cannot␣be␣address(0)" );
   require( address(_poolsConfig) != address(0), "_poolsConfig␣cannot␣
      ↪ be␣address(0)" );
   require( address(_rewardsConfig) != address(0), "_rewardsConfig␣
      ↪ cannot␣be␣address(0)" );
```

CVF-109. INFO

- **Category** Suboptimal
- **Source** RewardsEmitter.sol

Recommendation The value ”1 days * 100000” could be rendered as ”1e5 days”.

```
118 uint256 denominatorMult = 1 days * 100000; // simplification of
       ↪ numberSecondsInOneDay * (100 percent) * 1000
```

CVF-110. INFO

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

Recommendation The value ”1 days * 100000” should be a named constant.

```
118 uint256 denominatorMult = 1 days * 100000; // simplification of
       ↪ numberSecondsInOneDay * (100 percent) * 1000
```

CVF-111. FIXED

- **Category** Suboptimal
- **Source** SaltRewards.sol

Description These checks are redundant as dead addresses could anyway be passed.

Recommendation Consider removing these checks.

```
30 require( address(_stakingRewardsEmitter) != address(0), "
      ↪ _stakingRewardsEmitter␣cannot␣be␣address(0)" );
   require( address(_liquidityRewardsEmitter) != address(0), "
      ↪ _liquidityRewardsEmitter␣cannot␣be␣address(0)" );
   require( address(_exchangeConfig) != address(0), "_exchangeConfig␣
      ↪ cannot␣be␣address(0)" );
   require( address(_rewardsConfig) != address(0), "_rewardsConfig␣
      ↪ cannot␣be␣address(0)" );
```

CVF-112. INFO

- **Category** Suboptimal
- **Source** SaltRewards.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

```
63 function _sendLiquidityRewards( uint256 liquidityRewardsAmount,
      ↪ uint256 directRewardsForSaltUSDS, bytes32[] memory poolIDs,
      ↪ uint256[] memory profitsForPools, uint256 totalProfits )
      ↪ internal
```

CVF-113. INFO

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

Recommendation The value ”100” should be a named constant.

```
141 uint256 directRewardsForSaltUSDS = ( saltRewardsToDistribute *
       ↪ rewardsConfig.percentRewardsSaltUSDS() ) / 100;
```

```
145 uint256 stakingRewardsAmount = ( remainingRewards * rewardsConfig.
       ↪ stakingRewardsPercent() ) / 100;
```

CVF-114. INFO

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

Description The interface name is quite generic and is asset-neutral, while actually it contains functions for two particular assets.

Recommendation Consider either making this interface more generic, or making its name asset-specific.

```
5 interface IPriceFeed
```

CVF-115. FIXED

- **Category** Documentation
- **Source** IPriceFeed.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

```
7 function getPriceBTC() external view returns (uint256);
  function getPriceETH() external view returns (uint256);
```

CVF-116. FIXED

- **Category** Documentation
- **Source** IPriceFeed.sol

Description The base currency for the prices is unclear.

Recommendation Consider documenting.

```
7 function getPriceBTC() external view returns (uint256);
  function getPriceETH() external view returns (uint256);
```

CVF-117. INFO

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *Events are already emitted.*

```
9  function setInitialFeeds( IPriceFeed _priceFeed1, IPriceFeed
       ↪ _priceFeed2, IPriceFeed _priceFeed3 ) external;
10 function setPriceFeed( uint256 priceFeedNum, IPriceFeed newPriceFeed
       ↪  ) external; // onlyOwner
   function changeMaximumPriceFeedPercentDifferenceTimes1000(bool
       ↪ increase) external; // onlyOwner
   function changeSetPriceFeedCooldown(bool increase) external; //
       ↪ onlyOwner
```

CVF-118. FIXED

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

Description The name looks like a setter, while actually this is a ”view” function.

Recommendation Consider renaming.

```
16 function setPriceFeedCooldown() external view returns (uint256);
```

CVF-119. FIXED

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

Recommendation The type for these variables should be ”AggregatorV3Interface”.

```
13     address immutable public CHAINLINK_BTC_USD;
   address immutable public CHAINLINK_ETH_USD;
```

CVF-120. FIXED

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

Recommendation The type for arguments should be ”AggregatorV3Interface”.

```
17 constructor( address _CHAINLINK_BTC_USD, address _CHAINLINK_ETH_USD
   ↪ )
```

CVF-121. FIXED

- **Category** Suboptimal
- **Source** CoreChainlinkFeed.sol

Recommendation These checks are redundant as dead addresses could anyway be passed.

```
19 require( _CHAINLINK_BTC_USD != address(0), "_CHAINLINK_BTC_USD␣
   ↪ cannot␣be␣address(0)" );
20 require( _CHAINLINK_ETH_USD != address(0), "_CHAINLINK_ETH_USD␣
   ↪ cannot␣be␣address(0)" );
```

CVF-122. FIXED

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

Recommendation The argument type should be ”AggregatorV3Interface”.

```
29 function latestChainlinkPrice(address _chainlinkFeed) public view
   ↪ returns (uint256)
```

CVF-123. FIXED

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

Description This should be done only if answerDelay <= 60 minutes.

```
44 price = _price;
```

CVF-124. FIXED

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

Recommendation This threshold should be a named constant.

```
50 if ( answerDelay > 60 minutes )
```

CVF-125. FIXED

- **Category** Suboptimal
- **Source** CoreSaltyFeed.sol

Description These checks are redundant as dead addressed could anyway be passed.

```
24 require( address(_pools) != address(0), "_pools cannot be address(0)
     ↪ " );
   require( address(_exchangeConfig) != address(0), "_exchangeConfig
     ↪ cannot be address(0)" );
```

CVF-126. INFO

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

Recommendation The values ”10**8” and ”10**18” should be named constants.

```
43 return ( reservesUSDS * 10**8 ) / reservesWBTC;
```

```
55 return ( reservesUSDS * 10**18 ) / reservesWETH;
```

CVF-127. INFO

- **Category** Suboptimal
- **Source** CoreSaltyFeed.sol

Recommendation These values could be rendered as ”1e8” and ”1e18”.

```
43 return ( reservesUSDS * 10**8 ) / reservesWBTC;
```

```
55 return ( reservesUSDS * 10**18 ) / reservesWETH;
```

CVF-128. FIXED

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

Recommendation The type for these variables should be ”IUniswapV3Pool”.

```
21 address immutable public UNISWAP_V3_WBTC_WETH;
       address immutable public UNISWAP_V3_WETH_USDC;
```

CVF-129. FIXED

- **Category** Suboptimal
- **Source** CoreUniswapFeed.sol

Description These checks are redundant as dead addresses could anyway be passed.

```
34 require( address(_wbtc) != address(0), "_wbtc␣cannot␣be␣address(0)"
      ↪ );
   require( address(_weth) != address(0), "_weth␣cannot␣be␣address(0)"
      ↪ );
   require( address(_usdc) != address(0), "_usdc␣cannot␣be␣address(0)"
      ↪ );
   require( _UNISWAP_V3_WBTC_WETH != address(0), "_UNISWAP_V3_WBTC_WETH
      ↪ ␣cannot␣be␣address(0)" );
   require( _UNISWAP_V3_WETH_USDC != address(0), "_UNISWAP_V3_WETH_USDC
      ↪ ␣cannot␣be␣address(0)" );
```

CVF-130. FIXED

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

Recommendation The type for the ”pool” argument should be ”IUniswapV3Pool”.

```
56 function _getUniswapTwapWei( address pool, uint256 twapInterval )
      ↪ public view returns (uint256)
```

```
87 function getUniswapTwapWei( address pool, uint256 twapInterval )
      ↪ public virtual view returns (uint256)
```

CVF-131. INFO

- **Category** Suboptimal
- **Source** CoreUniswapFeed.sol

Recommendation Consider splitting this branch into two branches: one when decimals0 - decimals1 > 18 and another when decimals0 - decimals1 <= 18. This would allow making code simpler and more efficient.

```
77 if ( decimals0 > decimals1 )
       return ( FixedPoint96.Q96 * ( 10 ** 18 ) ) / ( p * ( 10 ** (
          ↪  decimals0 - decimals1 ) ) );
```

CVF-132. INFO

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

Description This shouldn't be executed in case uniswapWBTC_WETH is zero.

```
108 uint256 uniswapWETH_USDC = getUniswapTwapWei( UNISWAP_V3_WETH_USDC,
      ↪ twapInterval );
```

CVF-133. INFO

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

Recommendation Events are usually named via nouns, such as "PriceFeed" or "MaximumPriceFeedPercentDifference".

```
15 event PriceFeedSet(uint256 indexed priceFeedNum, address indexed
      ↪ newPriceFeed);
   event MaximumPriceFeedPercentDifferenceChanged(uint256
      ↪ newMaxDifference);
   event SetPriceFeedCooldownChanged(uint256 newCooldown);
```

CVF-134. FIXED

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

Recommendation The type for the ”newPriceFeed” parameter should be ”IPriceFeed”.

```
15 event PriceFeedSet(uint256 indexed priceFeedNum, address indexed
   ↪ newPriceFeed);
```

CVF-135. INFO

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

Recommendation The default values should be named constants.

```
29 uint256 public maximumPriceFeedPercentDifferenceTimes1000 = 3000; //
   ↪  Defaults to 3.0% with a 1000x multiplier
```

```
34 uint256 public setPriceFeedCooldown = 35 days;
```

CVF-136. INFO

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

Description This function should emit some event.

Client Comment *Events are already emitted.*

```
37 function setInitialFeeds( IPriceFeed _priceFeed1, IPriceFeed
   ↪ _priceFeed2, IPriceFeed _priceFeed3 ) public onlyOwner
```

CVF-137. FIXED

- **Category** Suboptimal
- **Source** PriceAggregator.sol

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

Client Comment *Removed*

```
39 require( address(priceFeed1) == address(0), "setInitialFeeds()␣can␣
   ↪ only␣be␣called␣once" );
```

CVF-138. INFO

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

Recommendation These values should be named constants.

```
69 if (maximumPriceFeedPercentDifferenceTimes1000 < 7000)
70     maximumPriceFeedPercentDifferenceTimes1000 += 500;
```

```
74 if (maximumPriceFeedPercentDifferenceTimes1000 > 1000)
       maximumPriceFeedPercentDifferenceTimes1000 -= 500;
```

```
86 if (setPriceFeedCooldown < 45 days)
       setPriceFeedCooldown += 5 days;
```

```
91 if (setPriceFeedCooldown > 30 days)
       setPriceFeedCooldown -= 5 days;
```

CVF-139. INFO

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

Recommendation The value ”100000” should be a named constant.

```
142 if (  (_absoluteDifference(priceA, priceB) * 100000) / averagePrice
    ↪ > maximumPriceFeedPercentDifferenceTimes1000 )
```

CVF-140. INFO

- **Category** Readability
- **Source** PriceAggregator.sol

Recommendation The value ”100000” could be rendered as ”1e5”.

```
142 if (  (_absoluteDifference(priceA, priceB) * 100000) / averagePrice
    ↪ > maximumPriceFeedPercentDifferenceTimes1000 )
```

CVF-141. INFO

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *Events are already emitted.*

```
13 function setContracts( IDAO _dao, ICollateralAndLiquidity
    ↪ _collateralAndLiquidity ) external; // onlyOwner
```

```
15 function addLiquidity( IERC20 tokenA, IERC20 tokenB, uint256
    ↪ maxAmountA, uint256 maxAmountB, uint256 minLiquidityReceived,
    ↪ uint256 totalLiquidity ) external returns (uint256
    ↪ addedAmountA, uint256 addedAmountB, uint256 addedLiquidity);
function removeLiquidity( IERC20 tokenA, IERC20 tokenB, uint256
    ↪ liquidityToRemove, uint256 minReclaimedA, uint256
    ↪ minReclaimedB, uint256 totalLiquidity ) external returns (
    ↪ uint256 reclaimedA, uint256 reclaimedB);
```

```
18 function deposit( IERC20 token, uint256 amount ) external;
function withdraw( IERC20 token, uint256 amount ) external;
20 function swap( IERC20 swapTokenIn, IERC20 swapTokenOut, uint256
    ↪ swapAmountIn, uint256 minAmountOut, uint256 deadline )
    ↪ external returns (uint256 swapAmountOut);
function depositSwapWithdraw(IERC20 swapTokenIn, IERC20 swapTokenOut
    ↪ , uint256 swapAmountIn, uint256 minAmountOut, uint256 deadline
    ↪  ) external returns (uint256 swapAmountOut);
function depositDoubleSwapWithdraw( IERC20 swapTokenIn, IERC20
    ↪ swapTokenMiddle, IERC20 swapTokenOut, uint256 swapAmountIn,
    ↪ uint256 minAmountOut, uint256 deadline ) external returns (
    ↪ uint256 swapAmountOut);
```

CVF-142. INFO

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *Events are already emitted.*

```
10 function whitelistPool(  IPools pools, IERC20 tokenA, IERC20 tokenB
      ↪ ) external; // onlyOwner
   function unwhitelistPool( IPools pools, IERC20 tokenA, IERC20 tokenB
      ↪  ) external; // onlyOwner
   function changeMaximumWhitelistedPools(bool increase) external; //
      ↪ onlyOwner
   function changeMaximumInternalSwapPercentTimes1000(bool increase)
      ↪ external; // onlyOwner
```

CVF-143. FIXED

- **Category** Documentation
- **Source** IPoolsConfig.sol

Description The ”wbtc” and ”weth” arguments seem odd.

Recommendation Consider clearly explaining their role.

Client Comment *Comment added.*

```
23 function tokenHasBeenWhitelisted( IERC20 token, IERC20 wbtc, IERC20
      ↪ weth ) external view returns (bool);
```

CVF-144. FIXED

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

Recommendation The semantics of the returned value is unclear, taking into account that there are three tokens passed as arguments.

Client Comment *Comment added.*

```
23 function tokenHasBeenWhitelisted( IERC20 token, IERC20 wbtc, IERC20
      ↪ weth ) external view returns (bool);
```

CVF-145. FIXED

- **Category** Suboptimal
- **Source** PoolMath.sol

Description The same checks and calculation are performed on both, decreasing and restoring precision.

Recommendation Consider refactoring to reuse their results.

Client Comment *Replaced with suggested normalized bit method.*

```
124 if ( decimals == REDUCED_DECIMALS )
```

```
128 if ( decimals < REDUCED_DECIMALS )
        return int256( n * 10**(REDUCED_DECIMALS - decimals) );
```

```
132 return int256( n / 10**(decimals - REDUCED_DECIMALS) );
```

```
144 if ( decimals == REDUCED_DECIMALS )
```

```
148 if ( decimals < REDUCED_DECIMALS )
        return uint256(n) / 10**(REDUCED_DECIMALS - decimals);
```

```
152 return uint256(n) * 10**(decimals - REDUCED_DECIMALS);
```

CVF-146. FIXED

- **Category** Suboptimal
- **Source** PoolMath.sol

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

Recommendation Consider treating all token amounts as integers.

Client Comment *Removed.*

```
207 uint8 decimalsA = ERC20(address(tokenA)).decimals();
    uint8 decimalsB = ERC20(address(tokenB)).decimals();
```

CVF-147. FIXED

- **Category** Suboptimal
- **Source** PoolStats.sol

Description These checks are redundant as dead addresses could anyway be passed.

Recommendation Consider removing these checks.

```
27 require( address(_exchangeConfig) != address(0), "_exchangeConfig␣
      ↪ cannot␣be␣address(0)" );
   require( address(_poolsConfig) != address(0), "_poolsConfig␣cannot␣
      ↪ be␣address(0)" );
```

CVF-148. INFO

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

Recommendation The return type should be ”uint256” to avoid unsafe type conversion.

Client Comment *Pool indicies are stored as uint64 within the ArbitrageIndicies structure. The maximum number of pools is 100 so uint64 should practically never overflow.*

```
61 function _poolIndex( IERC20 tokenA, IERC20 tokenB, bytes32[] memory
      ↪ poolIDs ) internal pure returns (uint64 index)
```

CVF-149. INFO

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

Recommendation Events are usually named via nouns, such as ”AddedLiquiditty” or ”RemovedLiquidity”.

```
25 event LiquidityAdded(address indexed tokenA, address indexed tokenB,
   ↪  uint256 addedAmountA, uint256 addedAmountB, uint256
   ↪ addedLiquidity);
   event LiquidityRemoved(address indexed tokenA, address indexed
   ↪ tokenB, uint256 reclaimedA, uint256 reclaimedB, uint256
   ↪ removedLiquidity);
   event TokenDeposit(address indexed user, IERC20 indexed token,
   ↪ uint256 amount);
   event TokenWithdrawal(address indexed user, IERC20 indexed token,
   ↪ uint256 amount);
   event SwapAndArbitrage(address indexed user, IERC20 indexed
   ↪ swapTokenIn, IERC20 indexed swapTokenOut, uint256 swapAmountIn
   ↪ , uint256 swapAmountOut, uint256 arbitrageProfit);
```

CVF-150. FIXED

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

Recommendation The type for the token parameters should be ”IERC20”.

```
25 event LiquidityAdded(address indexed tokenA, address indexed tokenB,
   ↪  uint256 addedAmountA, uint256 addedAmountB, uint256
   ↪ addedLiquidity);
   event LiquidityRemoved(address indexed tokenA, address indexed
   ↪ tokenB, uint256 reclaimedA, uint256 reclaimedB, uint256
   ↪ removedLiquidity);
```

CVF-151. INFO

- **Category** Procedural
- **Source** Pools.sol

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

```
55 {
   }
```

CVF-152. FIXED

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

Description These functions should emit some events.

Client Comment *setContracts() is only called once on deployment. BallotFinalized event added.*

```
60 function setContracts( IDAO _dao, ICollateralAndLiquidity
      ↪ _collateralAndLiquidity ) external onlyOwner
```

```
73 function startExchangeApproved() external nonReentrant
```

CVF-153. FIXED

- **Category** Suboptimal
- **Source** Pools.sol

Description These checks are redundant as dead addresses could anyway be passed.

Recommendation Consider removing these checks.

```
62 require( address(_dao) != address(0), "_dao␣cannot␣be␣address(0)" );
   require( address(_collateralAndLiquidity) != address(0), "
      ↪ _collateralAndLiquidity␣cannot␣be␣address(0)" );
```

CVF-154. INFO

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

Recommendation Events are usually named via nouns, such as ”WhitelistedPool” or ”MaximumWhitelistedPools”.

```
13 event PoolWhitelisted(address indexed tokenA, address indexed tokenB
      ↪ );
   event PoolUnwhitelisted(address indexed tokenA, address indexed
      ↪ tokenB);
   event MaximumWhitelistedPoolsChanged(uint256 newMaxPools);
   event maximumInternalSwapPercentTimes1000Changed(uint256 newMaxSwap)
      ↪ ;
```

CVF-155. FIXED

- **Category** Documentation
- **Source** PoolsConfig.sol

Description The semantics of the values in this set is unclear.

Recommendation Consider documenting.

Client Comment *Comment modified.*

```
29 EnumerableSet.Bytes32Set private _whitelist;
```

CVF-156. INFO

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

Recommendation The default values should be named constants.

```
37 uint256 public maximumWhitelistedPools = 50;
```

```
41 uint256 public maximumInternalSwapPercentTimes1000 = 1000;  //
      ↪ Defaults to 1.0% with a 1000x multiplier
```

CVF-157. INFO

- **Category** Readability
- **Source** PoolsConfig.sol

Recommendation This value could be rendered as ”0.01e5”.

```
41 uint256 public maximumInternalSwapPercentTimes1000 = 1000;  //
     ↪ Defaults to 1.0% with a 1000x multiplier
```

CVF-158. INFO

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

Recommendation The values ”100”, ”10”, and ”20” should be named constants.

```
81 if (maximumWhitelistedPools < 100)
       maximumWhitelistedPools += 10;
```

```
86 if (maximumWhitelistedPools > 20)
       maximumWhitelistedPools -= 10;
```

CVF-159. INFO

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

Recommendation The values ”2000”, ”250”, and ”250” should be named constants.

```
98 if (maximumInternalSwapPercentTimes1000 < 2000)
       maximumInternalSwapPercentTimes1000 += 250;
```

```
103 if (maximumInternalSwapPercentTimes1000 > 250)
        maximumInternalSwapPercentTimes1000 -= 250;
```

CVF-160. FIXED

- **Category** Suboptimal
- **Source** PoolsConfig.sol

Recommendation This could be simplified as: return poolID == PoolUtils.STAKED_SALT || _whitelist.contains (poolID);

Client Comment

```
122 if ( poolID == PoolUtils.STAKED_SALT )
        return true;
```

```
125 return _whitelist.contains( poolID );
```

CVF-161. FIXED

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

Description The name llooks ike a modifier.

Recommendation Consider renaming to "_poolID".

Client Comment

```
21 function _poolIDOnly( IERC20 tokenA, IERC20 tokenB ) internal pure
   ↪ returns (bytes32 poolID)
```

CVF-162. FIXED

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

Description The name doesn't reflect the fact that the function returns additional information along with pool ID.

Recommendation Consider renaming to something like "_poolIDandFlipFlag".

Client Comment *Renamed as _poolIDAndFlipped*

```
32 function _poolID( IERC20 tokenA, IERC20 tokenB ) internal pure
   ↪ returns (bytes32 poolID, bool flipped)
```

CVF-163. INFO

- **Category** Procedural
- **Source** PoolUtils.sol

Recommendation Brackets are redundant here.

Client Comment *Kept as is for clarity.*

```
61 uint256 maxAmountIn = reservesIn *
   ↪ maximumInternalSwapPercentTimes1000 / (100000);
```

CVF-164. INFO

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

Recommendation The denominator should be a named constant.

```
61 uint256 maxAmountIn = reservesIn *
   ↪ maximumInternalSwapPercentTimes1000 / (100000);
```

CVF-165. FIXED

- **Category** Suboptimal
- **Source** PoolUtils.sol

Recommendation Using 10^5 as a denominator is quite unusual. More common denominators are 10^4 (basis points) or 10^6.

Client Comment *Clarified as 100 * 1000*

```
61 uint256 maxAmountIn = reservesIn *
   ↪ maximumInternalSwapPercentTimes1000 / (100000);
```

CVF-166. FIXED

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

Recommendation These function should emit some events and these events should be declared in this interface.

Client Comment *BallotFinalized event added*

```
7 function vote( bool voteStartExchangeYes, bytes memory signature  )
    ↪ external;
  function finalizeBallot() external;
```

CVF-167. INFO

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

Recommendation This function should emit some events and this event should be declared in this interface.

```
9 function distributionApproved() external;
```

CVF-168. FIXED

- **Category** Suboptimal
- **Source** IAirdrop.sol

Description This function doesn't scale.

Recommendation Consider implementing an ability to obtain the list of authorized wallets in chunks.

Client Comment *Function removed*

```
18 function authorizedWallets() external view returns (address[]
    ↪ calldata);
```

CVF-169. INFO

- **Category** Suboptimal
- **Source** Airdrop.sol

Recommendation This mapping is redundant. Just remove claimed addresses from "_authorizedUsers".

Client Comment *This functionality is not redundant. The UI alerts the user when they try to claim the airdrop, but have already done so.*

```
29 mapping(address=>bool) public claimed;
```

CVF-170. FIXED

- **Category** Suboptimal
- **Source** BootstrapBallot.sol

Recommendation These checks are redundant as it is anyway possible to pass dead addresses.

Client Comment

```
34 require( address(_exchangeConfig) != address(0), "_exchangeConfig␣
   ↪ cannot␣be␣address(0)" );
   require( address(_airdrop) != address(0), "_airdrop␣cannot␣be␣
   ↪ address(0)" );
```

CVF-171. FIXED

- **Category** Suboptimal
- **Source** InitialDistribution.sol

Recommendation These checks are redundant as it is anyway possible to pass dead addresses.

Client Comment

```
36 require( address(_salt) != address(0), "_salt␣cannot␣be␣address(0)"
     ↪ );
   require( address(_poolsConfig) != address(0), "_poolsConfig␣cannot␣
     ↪ be␣address(0)" );
   require( address(_emissions) != address(0), "_emissions␣cannot␣be␣
     ↪ address(0)" );
   require( address(_bootstrapBallot) != address(0), "_bootstrapBallot␣
     ↪ cannot␣be␣address(0)" );
40 require( address(_dao) != address(0), "_dao␣cannot␣be␣address(0)" );
   require( address(_daoVestingWallet) != address(0), "
     ↪ _daoVestingWallet␣cannot␣be␣address(0)" );
   require( address(_teamVestingWallet) != address(0), "
     ↪ _teamVestingWallet␣cannot␣be␣address(0)" );
   require( address(_airdrop) != address(0), "_airdrop␣cannot␣be␣
     ↪ address(0)" );
   require( address(_saltRewards) != address(0), "_saltRewards␣cannot␣
     ↪ be␣address(0)" );
   require( address(_collateralAndLiquidity) != address(0), "
     ↪ _collateralAndLiquidity␣cannot␣be␣address(0)" );
```

CVF-172. INFO

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

Recommendation These values should be named constants.

```
64 require( salt.balanceOf(address(this)) == 100 * MILLION_ETHER, "SALT
      ↪ ␣has␣already␣been␣sent␣from␣the␣contract" );

67 salt.safeTransfer( address(emissions), 52 * MILLION_ETHER );

70 salt.safeTransfer( address(daoVestingWallet), 25 * MILLION_ETHER );

73 salt.safeTransfer( address(teamVestingWallet), 10 * MILLION_ETHER );

76 salt.safeTransfer( address(airdrop), 5 * MILLION_ETHER );

83 salt.safeTransfer( address(saltRewards), 8 * MILLION_ETHER );
   saltRewards.sendInitialSaltRewards(5 * MILLION_ETHER, poolIDs );
```

CVF-173. INFO

- **Category** Procedural
- **Source** InitialDistribution.sol

Description Hardcoding constant values in comments is a bad practice as in case the values will ever change, it will be hard to update them properly everywhere.

Recommendation Consider referring to the constants by names rather than by values.

```
66 // 52 million          Emissions

69     // 25 million              DAO Reserve Vesting Wallet

72     // 10 million              Initial Development Team Vesting
          ↪ Wallet

75     // 5 million               Airdrop Participants

81     // 5 million               Liquidity Bootstrapping
       // 3 million               Staking Bootstrapping
```

CVF-174. FIXED

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

Recommendation This function should emit some event and this event should be declared in this interface.

Client Comment *AccessGranted event added.*

```
8 function grantAccess(bytes calldata signature) external;
```

CVF-175. INFO

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *setAccessManager already emits the SetContract event. setContracts is only called once at deployment time.*

```
21 function setContracts( IDAO _dao, IUpkeep _upkeep,
   ↪ IInitialDistribution _initialDistribution, IAirdrop _airdrop,
   ↪ VestingWallet _teamVestingWallet, VestingWallet
   ↪ _daoVestingWallet ) external; // onlyOwner
function setAccessManager( IAccessManager _accessManager ) external;
   ↪  // onlyOwner
```

CVF-176. INFO

- **Category** Documentation
- **Source** ICalledContract.sol

Description The semantics of the argument is unclear.

Recommendation Consider giving a descriptive name to the argument and/or documenting.

Client Comment *The argument has no predefined meaning or purpose. The meaning is contingent on the contract being called.*

```
7 function callFromDAO(uint256) external;
```

CVF-177. INFO

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *The POLWithdrawn event is already emitted. finalizeBallot already emits events dependent on the type of ballot being finalized.*

```
11 function finalizeBallot( uint256 ballotID ) external;
```

```
17 function withdrawPOL( IERC20 tokenA, IERC20 tokenB, uint256
   ↪ percentToLiquidate ) external;
```

CVF-178. INFO

- **Category** Procedural
- **Source** IProposals.sol

Description Defining top-level types in a file named after an interface makes it harder to navigate through code.

Recommendation Consider either moving the type definitions into the interface, or moving them into a separate file.

```
7 enum Vote { INCREASE, DECREASE, NO_CHANGE, YES, NO }
enum BallotType { PARAMETER, WHITELIST_TOKEN, UNWHITELIST_TOKEN,
   ↪ SEND_SALT, CALL_CONTRACT, INCLUDE_COUNTRY, EXCLUDE_COUNTRY,
   ↪ SET_CONTRACT, SET_WEBSITE_URL, CONFIRM_SET_CONTRACT,
   ↪ CONFIRM_SET_WEBSITE_URL }
```

```
10 struct UserVote
```

```
16 struct Ballot
```

CVF-179. INFO

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

Recommendation These functions should return the ID of the created ballot.

Client Comment *Unnecessary as all required functionality is already within _possiblyCreateProposal.*

```
35 function createConfirmationProposal( string calldata ballotName,
      ↪ BallotType ballotType, address address1, string calldata
      ↪ string1, string calldata description ) external;
```

```
38 function proposeParameterBallot( uint256 parameterType, string
      ↪ calldata description ) external;
   function proposeTokenWhitelisting( IERC20 token, string calldata
      ↪ tokenIconURL, string calldata description ) external;
40 function proposeTokenUnwhitelisting( IERC20 token, string calldata
      ↪ tokenIconURL, string calldata description ) external;
   function proposeSendSALT( address wallet, uint256 amount, string
      ↪ calldata description ) external;
   function proposeCallContract( address contractAddress, uint256
      ↪ number, string calldata description ) external;
   function proposeCountryInclusion( string calldata country, string
      ↪ calldata description ) external;
   function proposeCountryExclusion( string calldata country, string
      ↪ calldata description ) external;
   function proposeSetContractAddress( string calldata contractName,
      ↪ address newAddress, string calldata description ) external;
   function proposeWebsiteUpdate( string calldata newWebsiteURL, string
      ↪  calldata description ) external;
```

CVF-180. FIXED

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

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment

```
35 function createConfirmationProposal( string calldata ballotName,
      ↪ BallotType ballotType, address address1, string calldata
      ↪ string1, string calldata description ) external;
   function markBallotAsFinalized( uint256 ballotID ) external;
```

```
48 function castVote( uint256 ballotID, Vote vote ) external;
```

CVF-181. INFO

- **Category** Suboptimal
- **Source** IProposals.sol

Description Passing a string is inefficient.

Recommendation Consider passing the ballot name hash instead.

Client Comment *Acceptable as this is simply a getter into the mapping already in place.*

```
52 function openBallotsByName( string calldata name ) external returns
      ↪ (uint256);
```

CVF-182. INFO

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

Recommendation Events are usually named via nouns, such as ”BootstrappingRewards” or ”PercentPolRewardsBurned”.

```
11 event BootstrappingRewardsChanged(uint256 newBootstrappingRewards);
   event PercentPolRewardsBurnedChanged(uint256
      ↪ newPercentPolRewardsBurned);
   event BaseBallotQuorumPercentChanged(uint256
      ↪ newBaseBallotQuorumPercentTimes1000);
   event BallotDurationChanged(uint256 newBallotDuration);
   event RequiredProposalPercentStakeChanged(uint256
      ↪ newRequiredProposalPercentStakeTimes1000);
   event MaxPendingTokensForWhitelistingChanged(uint256
      ↪ newMaxPendingTokensForWhitelisting);
   event ArbitrageProfitsPercentPOLChanged(uint256
      ↪ newArbitrageProfitsPercentPOL);
   event UpkeepRewardPercentChanged(uint256 newUpkeepRewardPercent);
```

CVF-183. INFO

- **Category** Suboptimal
- **Source** DAOConfig.sol

Recommendation The prefix ”new” in parameter names is redundant, as there are no corresponding ”old” parameters.

```
11 event BootstrappingRewardsChanged(uint256 newBootstrappingRewards);
   event PercentPolRewardsBurnedChanged(uint256
      ↪ newPercentPolRewardsBurned);
   event BaseBallotQuorumPercentChanged(uint256
      ↪ newBaseBallotQuorumPercentTimes1000);
   event BallotDurationChanged(uint256 newBallotDuration);
   event RequiredProposalPercentStakeChanged(uint256
      ↪ newRequiredProposalPercentStakeTimes1000);
   event MaxPendingTokensForWhitelistingChanged(uint256
      ↪ newMaxPendingTokensForWhitelisting);
   event ArbitrageProfitsPercentPOLChanged(uint256
      ↪ newArbitrageProfitsPercentPOL);
   event UpkeepRewardPercentChanged(uint256 newUpkeepRewardPercent);
```

CVF-184. INFO

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

Recommendation The default values should be named constants.

```
24      uint256 public bootstrappingRewards = 200000 ether;
```

```
28  uint256 public percentPolRewardsBurned = 50;
```

```
40      uint256 public baseBallotQuorumPercentTimes1000 = 10 * 1000; //
           ↪ Default 10% of the total amount of SALT staked with a 1000
           ↪ x multiplier
```

```
45      uint256 public ballotMinimumDuration = 10 days;
```

```
49      uint256 public requiredProposalPercentStakeTimes1000 = 500;  //
           ↪ Defaults to 0.50% with a 1000x multiplier
```

```
53      uint256 public maxPendingTokensForWhitelisting = 5;
```

```
57      uint256 public arbitrageProfitsPercentPOL = 20;
```

```
61      uint256 public upkeepRewardPercent = 5;
```

CVF-185. INFO

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation This value could be rendered as ”0.5e2”.

```
28  uint256 public percentPolRewardsBurned = 50;
```

CVF-186. INFO

- **Category** Suboptimal
- **Source** DAOConfig.sol

Description Using different denominator for different parameters makes code harder to read and more error-prone.

Recommendation Consider using the same denominator everywhere.

```
28 uint256 public percentPolRewardsBurned = 50;
```

```
40     uint256 public baseBallotQuorumPercentTimes1000 = 10 * 1000; //
          ↪ Default 10% of the total amount of SALT staked with a 1000
          ↪ x multiplier
```

```
49     uint256 public requiredProposalPercentStakeTimes1000 = 500;  //
          ↪ Defaults to 0.50% with a 1000x multiplier
```

```
57     uint256 public arbitrageProfitsPercentPOL = 20;
```

```
61     uint256 public upkeepRewardPercent = 5;
```

CVF-187. INFO

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation This value could be rendered as "0.1e5".

```
40 uint256 public baseBallotQuorumPercentTimes1000 = 10 * 1000; //
      ↪ Default 10% of the total amount of SALT staked with a 1000x
      ↪ multiplier
```

CVF-188. INFO

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation This value could be rendered as "0.005e5".

```
49 uint256 public requiredProposalPercentStakeTimes1000 = 500;  //
      ↪ Defaults to 0.50% with a 1000x multiplier
```

CVF-189. INFO

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation This value could be rendered as ”0.2e2”.

```
57 uint256 public arbitrageProfitsPercentPOL = 20;
```

CVF-190. INFO

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation This value could be rendered as ”0.05e2”.

```
61 uint256 public upkeepRewardPercent = 5;
```

CVF-191. INFO

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

Recommendation These values should be named constants.

```
68  if (bootstrappingRewards < 500000 * 1 ether)
        bootstrappingRewards += 50000 * 1 ether;
```

```
73  if (bootstrappingRewards > 50000 * 1 ether)
        bootstrappingRewards -= 50000 * 1 ether;
```

```
85              if (percentPolRewardsBurned < 75)
                    percentPolRewardsBurned += 5;
```

```
90              if (percentPolRewardsBurned > 25)
                    percentPolRewardsBurned -= 5;
```

```
102             if (baseBallotQuorumPercentTimes1000 < 20 * 1000)
                    baseBallotQuorumPercentTimes1000 += 1000;
```

```
107             if (baseBallotQuorumPercentTimes1000 > 5 * 1000 )
                    baseBallotQuorumPercentTimes1000 -= 1000;
```

```
119 if (ballotMinimumDuration < 14 days)
120     ballotMinimumDuration += 1 days;
```

```
124 if (ballotMinimumDuration > 3 days)
        ballotMinimumDuration -= 1 days;
```

```
136             if (requiredProposalPercentStakeTimes1000 < 2000) //
                   ↪ Maximum 2%
                    requiredProposalPercentStakeTimes1000 += 100; //
                       ↪  Increase by 0.10%
```

```
141             if (requiredProposalPercentStakeTimes1000 > 100) //
                   ↪ Minimum 0.10%
                    requiredProposalPercentStakeTimes1000 -= 100; //
                       ↪ Decrease by 0.10%
```

```
153             if (maxPendingTokensForWhitelisting < 12)
                    maxPendingTokensForWhitelisting += 1;
```

(158, 170, 175, 187, 192)

CVF-192. FIXED

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation These values could be rendered as ”500000 ether” and ”50000 ether”.

Client Comment

```
68 if (bootstrappingRewards < 500000 * 1 ether)
       bootstrappingRewards += 50000 * 1 ether;
```

```
73 if (bootstrappingRewards > 50000 * 1 ether)
       bootstrappingRewards -= 50000 * 1 ether;
```

CVF-193. INFO

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation Theses values could be rendered as ”0.75e2”, ”0.05e2”, and ”0.25e2”.

```
85 if (percentPolRewardsBurned < 75)
           percentPolRewardsBurned += 5;
```

```
90 if (percentPolRewardsBurned > 25)
           percentPolRewardsBurned -= 5;
```

CVF-194. INFO

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation These values could be rendered as ”0.2e5”, ”0.01e5”, and ”0.05e5”.

```
102 if (baseBallotQuorumPercentTimes1000 < 20 * 1000)
           baseBallotQuorumPercentTimes1000 += 1000;
```

```
107 if (baseBallotQuorumPercentTimes1000 > 5 * 1000 )
           baseBallotQuorumPercentTimes1000 -= 1000;
```

CVF-195. INFO

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation There values could be rendered as ”0.02e5” and ”0.001e5”.

```
136 if (requiredProposalPercentStakeTimes1000 < 2000) // Maximum 2%
        requiredProposalPercentStakeTimes1000 += 100; // Increase by
          ↪  0.10%
```

```
141 if (requiredProposalPercentStakeTimes1000 > 100) // Minimum 0.10%
    requiredProposalPercentStakeTimes1000 -= 100; // Decrease by
       ↪ 0.10%
```

CVF-196. INFO

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation These values could be rendered as ”0.45e2”, ”0.05e2” and ”0.15e2”.

```
170 if (arbitrageProfitsPercentPOL < 45)
        arbitrageProfitsPercentPOL += 5;
```

```
175 if (arbitrageProfitsPercentPOL > 15)
        arbitrageProfitsPercentPOL -= 5;
```

CVF-197. INFO

- **Category** Readability
- **Source** DAOConfig.sol

Recommendation These values could be rendered as ”0.1e2” and ”0.01e2”.

```
187 if (upkeepRewardPercent < 10)
        upkeepRewardPercent += 1;
```

```
192 if (upkeepRewardPercent > 1)
        upkeepRewardPercent -= 1;
```

CVF-198. INFO

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

Recommendation Events are usually named via nouns, such as ”FinalizedBallot” or ”Contract”.

```
25      event BallotFinalized(uint256 indexed ballotID);
event SetContract(string indexed ballotName, address indexed
   ↪ contractAddress);
event SetWebsiteURL(string newURL);
event WhitelistToken(address indexed token);
event UnwhitelistToken(address indexed token);
30 event GeoExclusionUpdated(string country, bool excluded);
event ArbitrageProfitsWithdrawn(address indexed upkeepContract,
   ↪ IERC20 indexed weth, uint256 withdrawnAmount);
event SaltSent(address indexed to, uint256 amount);
event ContractCalled(address indexed contractAddress, uint256
   ↪ indexed intArg);
event TeamRewardsTransferred(uint256 teamAmount);
```

```
36 event POLFormed(address indexed tokenA, address indexed tokenB,
   ↪ uint256 amountA, uint256 amountB);
event POLProcessed(uint256 claimedSALT);
event POLWithdrawn(address indexed tokenA, address indexed tokenB,
   ↪ uint256 withdrawnA, uint256 withdrawnB);
```

CVF-199. FIXED

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

Recommendation The type of the token parameters should be ”IERC20”.

Client Comment

```
28 event WhitelistToken(address indexed token);
event UnwhitelistToken(address indexed token);
```

```
36 event POLFormed(address indexed tokenA, address indexed tokenB,
   ↪ uint256 amountA, uint256 amountB);
```

```
38 event POLWithdrawn(address indexed tokenA, address indexed tokenB,
   ↪ uint256 withdrawnA, uint256 withdrawnB);
```

CVF-200. FIXED

- **Category** Documentation
- **Source** DAO.sol

Description The format of the keys is unclear.

Recommendation Consider documenting.

Client Comment

```
66 mapping(string=>bool) public excludedCountries;
```

CVF-201. FIXED

- **Category** Suboptimal
- **Source** DAO.sol

Recommendation These checks are redundant as it is anyway possible to pass dead addresses.

Client Comment

```
71 require( address(_pools) != address(0), "_pools␣cannot␣be␣address(0)
   ↪ " );
   require( address(_proposals) != address(0), "_proposals␣cannot␣be␣
   ↪ address(0)" );
   require( address(_exchangeConfig) != address(0), "_exchangeConfig␣
   ↪ cannot␣be␣address(0)" );
   require( address(_poolsConfig) != address(0), "_poolsConfig␣cannot␣
   ↪ be␣address(0)" );
   require( address(_stakingConfig) != address(0), "_stakingConfig␣
   ↪ cannot␣be␣address(0)" );
   require( address(_rewardsConfig) != address(0), "_rewardsConfig␣
   ↪ cannot␣be␣address(0)" );
   require( address(_stableConfig) != address(0), "_stableConfig␣cannot
   ↪ ␣be␣address(0)" );
   require( address(_daoConfig) != address(0), "_daoConfig␣cannot␣be␣
   ↪ address(0)" );
   require( address(_priceAggregator) != address(0), "_priceAggregator␣
   ↪ cannot␣be␣address(0)" );
80 require( address(_liquidityRewardsEmitter) != address(0), "
   ↪ _liquidityRewardsEmitter␣cannot␣be␣address(0)" );
   require( address(_collateralAndLiquidity) != address(0), "
   ↪ _collateralAndLiquidity␣cannot␣be␣address(0)" );
```

CVF-202. INFO

- **Category** Suboptimal
- **Source** DAO.sol

Description Hardcoding the default exclusions looks odd.

Recommendation Consider passing the list of default exclusions as a constructor argument.

Client Comment *Added explicitly to the contract for the purposes of transparency - rather than included as an argument on deployment which would be harder to see.*

```
105 // Excluded by default: United States, Canada, United Kingdom, China
       ↪ , India, Pakistan, Russian, Afghanistan, Cuba, Iran, North
       ↪ Korea, Syria, Venezuela
```

```
107 excludedCountries["US"] = true;
    excludedCountries["CA"] = true;
    excludedCountries["GB"] = true;
110 excludedCountries["CN"] = true;
    excludedCountries["IN"] = true;
    excludedCountries["PK"] = true;
    excludedCountries["RU"] = true;
    excludedCountries["AF"] = true;
    excludedCountries["CU"] = true;
    excludedCountries["IR"] = true;
    excludedCountries["KP"] = true;
    excludedCountries["SY"] = true;
    excludedCountries["VE"] = true;
```

CVF-203. FIXED

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

Recommendation This event should contain the winning vote as a parameter.

Client Comment *Fixed as suggested*

```
138 emit BallotFinalized(ballotID);
```

CVF-204. INFO

- **Category** Suboptimal
- **Source** DAO.sol

Description This event is emitted even for unknown ballot names when no changes were actually made.

Recommendation Consider emitting no event in such cases.

Client Comment *Acceptable to record that the SetContract method was called even without changing addresses.*

```
155 emit SetContract(ballot.ballotName, ballot.address1);
```

CVF-205. INFO

- **Category** Suboptimal
- **Source** DAO.sol

Description There could be other reasons for a token transfer to fail. Performing the balance check before a transfer consumes gas without significant benefit.

Recommendation Consider catching errors from transfer calls instead.

```
180 // This should not happen but is here just in case - to prevent
    ↪ approved proposals from reverting on finalization.
if ( exchangeConfig.salt().balanceOf(address(this)) >= ballot.
    ↪ number1 )
```

CVF-206. INFO

- **Category** Suboptimal
- **Source** DAO.sol

Description Using denominator of 100 means quite coarse precision. It allows specifying percentages like 3% or 4%, but not 3.5%.

Recommendation Consider using bigger denominator.

Client Comment *Acceptable as percentToLiquidate is fixed at 1% - as called from Liquidizer.*

```
380 uint256 liquidityToWithdraw = (liquidityHeld * percentToLiquidate) /
    ↪  100;
```

CVF-207. INFO

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

Recommendation Events are usually named via nouns, such as ”Proposal” or ”Finalized-Ballot”.

```
23 event ProposalCreated(uint256 indexed ballotID, BallotType
   ↪ ballotType, string ballotName);
   event BallotFinalized(uint256 indexed ballotID);
   event VoteCast(address indexed voter, uint256 indexed ballotID, Vote
   ↪  vote, uint256 votingPower);
```

CVF-208. INFO

- **Category** Suboptimal
- **Source** Proposals.sol

Recommendation These two variables could be replaces with a single variable of type ”Ballot[]”.

```
41 mapping(uint256=>Ballot) public ballots;
   uint256 public nextBallotID = 1;
```

CVF-209. INFO

- **Category** Suboptimal
- **Source** Proposals.sol

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

```
41 mapping(uint256=>Ballot) public ballots;
```

```
51 mapping(uint256=>mapping(Vote=>uint256)) private _votesCastForBallot
   ↪ ;
```

```
55 mapping(uint256=>mapping(address=>UserVote)) private
   ↪ _lastUserVoteForBallot;
```

```
63 mapping(uint256=>address) private _usersThatProposedBallots;
```

CVF-210. INFO

- **Category** Suboptimal
- **Source** Proposals.sol

Description Hardcoding particular constant values in comments and error messages is a bad practice, as in case the value will ever change it would be hard to properly update it everywhere.

Recommendation Consider referring to constants by their names rather than values.

65
```
// The time at which the first proposal can be made (45 days after
   ↪ deployment).
```

88
```
        require( block.timestamp >= firstPossibleProposalTimestamp,
           ↪ "Cannot␣propose␣ballots␣within␣the␣first␣45␣days␣of␣
           ↪ deployment" );
```

198
```
// Only one sendSALT Ballot can be open at a time and the sending
   ↪ limit is 5% of the current SALT balance of the DAO.
```

203
```
        // Limit to 5% of current balance
```

206
```
        require( amount <= maxSendable, "Cannot␣send␣more␣than␣5%␣of
           ↪ ␣the␣DAO␣SALT␣balance" );
```

318
```
// The required quorum is normally a default 10% of the amount of
   ↪ SALT staked.
// There is though a minimum of 0.50% of SALT.totalSupply (in the
   ↪ case that the amount of staked SALT is low - at launch for
   ↪ instance).
```

334
```
        // Make sure that the requiredQuorum is at least 0.50% of
           ↪ the total SALT supply.
        // Circulating supply after the first 45 days of emissions
           ↪ will be about 3 million - so this would require about
           ↪ 16% of the circulating
        // SALT to be staked and voting to pass a proposal (
           ↪ including whitelisting) 45 days after deployment..
```

CVF-211. INFO

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

Recommendation The value ”45 days” should be a named constant.

```
67 uint256 immutable firstPossibleProposalTimestamp = block.timestamp +
   ↪  45 days;
```

CVF-212. FIXED

- **Category** Suboptimal
- **Source** Proposals.sol

Recommendation These checks are redundant, as it is anyway possible to pass dead addresses.

Client Comment

```
72 require( address(_staking) != address(0), "_staking␣cannot␣be␣
   ↪ address(0)" );
require( address(_exchangeConfig) != address(0), "_exchangeConfig␣
   ↪ cannot␣be␣address(0)" );
require( address(_poolsConfig) != address(0), "_poolsConfig␣cannot␣
   ↪ be␣address(0)" );
require( address(_daoConfig) != address(0), "_daoConfig␣cannot␣be␣
   ↪ address(0)" );
```

CVF-213. INFO

- **Category** Procedural
- **Source** Proposals.sol

Recommendation Brackets around the numerator are redundant.

Client Comment *Kept as is for clarity.*

```
97 uint256 requiredXSalt = ( totalStaked * daoConfig.
   ↪ requiredProposalPercentStakeTimes1000() ) / ( 100 * 1000 );
```

CVF-214. INFO

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

Recommendation The value ”100 * 1000” should be a named constant.

```
97 uint256 requiredXSalt = ( totalStaked * daoConfig.
      ↪ requiredProposalPercentStakeTimes1000() ) / ( 100 * 1000 );
```

CVF-215. INFO

- **Category** Readability
- **Source** Proposals.sol

Recommendation This value ”100 * 1000” could be rendered as ”1e5”.

```
97 uint256 requiredXSalt = ( totalStaked * daoConfig.
      ↪ requiredProposalPercentStakeTimes1000() ) / ( 100 * 1000 );
```

```
327 requiredQuorum = ( 1 * totalStaked * daoConfig.
      ↪ baseBallotQuorumPercentTimes1000()) / ( 100 * 1000 );
```

```
329 requiredQuorum = ( 2 * totalStaked * daoConfig.
      ↪ baseBallotQuorumPercentTimes1000()) / ( 100 * 1000 );
```

```
332 requiredQuorum = ( 3 * totalStaked * daoConfig.
      ↪ baseBallotQuorumPercentTimes1000()) / ( 100 * 1000 );
```

CVF-216. INFO

- **Category** Suboptimal
- **Source** Proposals.sol

Recommendation This expression could be simplified as ”balance / 20”.

Client Comment *Kept as is for consistency with underlying functionality.*

```
205 uint256 maxSendable = balance * 5 / 100;
```

CVF-217. INFO

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

Recommendation The values ”5” and ”100” should be named constants.

```
205 uint256 maxSendable = balance * 5 / 100;
```

CVF-218. INFO

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

Recommendation The values ”1”, ”2”, ”3”, and ”100 * 1000” should be named constants.

```
327 requiredQuorum = ( 1 * totalStaked * daoConfig.
    ↪ baseBallotQuorumPercentTimes1000()) / ( 100 * 1000 );
```

```
329 requiredQuorum = ( 2 * totalStaked * daoConfig.
    ↪ baseBallotQuorumPercentTimes1000()) / ( 100 * 1000 );
```

```
332 requiredQuorum = ( 3 * totalStaked * daoConfig.
    ↪ baseBallotQuorumPercentTimes1000()) / ( 100 * 1000 );
```

CVF-219. INFO

- **Category** Readability
- **Source** Proposals.sol

Recommendation This expression could be simplified as ”totalSupply / 200”.

Client Comment *Kept as is for consistency with underlying functionality.*

```
338 uint256 minimumQuorum = totalSupply * 5 / 1000;
```

CVF-220. INFO

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

Recommendation The values ”5” and ”1000” should be named constants.

```
338 uint256 minimumQuorum = totalSupply * 5 / 1000;
```

CVF-221. INFO

- **Category** Suboptimal
- **Source** Proposals.sol

Recommendation These conditions are mutually exclusive, so the latter condition shouldn’t be checked in case the former condition is true.

```
375 if ( increaseTotal > decreaseTotal )
```

```
379 if ( decreaseTotal > increaseTotal )
```

CVF-222. FIXED

- **Category** Suboptimal
- **Source** Upkeep.sol

Recommendation These checks are redundant, as it is anyway possible to pass dead addresses.

Client Comment

```
69 require( address(_pools) != address(0), "_pools cannot be address(0)
      ↪ " );
70 require( address(_exchangeConfig) != address(0), "_exchangeConfig
      ↪ cannot be address(0)" );
   require( address(_poolsConfig) != address(0), "_poolsConfig cannot
      ↪ be address(0)" );
   require( address(_daoConfig) != address(0), "_daoConfig cannot be
      ↪ address(0)" );
   require( address(_stableConfig) != address(0), "_stableConfig cannot
      ↪ be address(0)" );
   require( address(_priceAggregator) != address(0), "_priceAggregator
      ↪ cannot be address(0)" );
   require( address(_saltRewards) != address(0), "_saltRewards cannot
      ↪ be address(0)" );
   require( address(_collateralAndLiquidity) != address(0), "
      ↪ _collateralAndLiquidity cannot be address(0)" );
   require( address(_emissions) != address(0), "_emissions cannot be
      ↪ address(0)" );
   require( address(_dao) != address(0), "_dao cannot be address(0)" );
```

CVF-223. INFO

- **Category** Procedural
- **Source** Upkeep.sol

Description Mentioning exact constant values in comments is a bad idea, as if constant values will be changed, it would be very easy to forget updating relevant comments.

Recommendation Consider giving constant names instead.

```
122 // 2. Withdraw existing WETH arbitrage profits from the Pools
       ↪ contract and reward the caller of performUpkeep() with default
       ↪  5% of the withdrawn amount.
```

```
129         // Default 5% of the arbitrage profits for the caller of
               ↪ performUpkeep()
```

CVF-224. INFO

- **Category** Suboptimal
- **Source** SigningTools.sol

Description Hardcoding mainnet addresses makes testing harder.

Recommendation Consider passing the expected signer address as a constructor argument and storing in an immutable variable.

```
7 address constant public EXPECTED_SIGNER = 0
    ↪ x1234519DCA2ef23207E1CA7fd70b96f281893bAa;
```

CVF-225. FIXED

- **Category** Suboptimal
- **Source** SigningTools.sol

Description This function is very inefficient.

Recommendation Consider optimizing like this: require (index <= array.length - 32); assembly { result := mload (add (array, add (0×20, index))) }

Client Comment *Using the suggested assembly variant below instead.*

```
10 function _slice32(bytes memory array, uint index) internal pure
    ↪ returns (bytes32 result)
```

CVF-226. FIXED

- **Category** Suboptimal
- **Source** SigningTools.sol

Recommendation This could be optimized as: assembly { r := mload (add (signature, 0×20)) s := mload (add (signature, 0×40)) v := mload (add (signature, 0×41)) }

Client Comment

```
27 bytes32 r = _slice32(signature, 0);
bytes32 s = _slice32(signature, 32);
uint8 v = uint8(signature[64]);
```

CVF-227. FIXED

- **Category** Suboptimal
- **Source** AccessManager.sol

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

Client Comment

```
32 require( address(_dao) != address(0), "_dao␣cannot␣be␣address(0)" );
```

CVF-228. FIXED

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

Description These functions should emit some events.

Client Comment *GeoExclusionUpdated now includes the geoVersion. AccessGranted event added.*

```
41 function excludedCountriesUpdated() external
```

```
61 function grantAccess(bytes calldata signature) external
```

CVF-229. FIXED

- **Category** Suboptimal
- **Source** ExchangeConfig.sol

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

Recommendation Consider removing these checks.

Client Comment

```
38 require( address(_salt) != address(0), "_salt cannot be address(0)"
      ↪ );
   require( address(_wbtc) != address(0), "_wbtc cannot be address(0)"
      ↪ );
40 require( address(_weth) != address(0), "_weth cannot be address(0)"
      ↪ );
   require( address(_dai) != address(0), "_dai cannot be address(0)" );
   require( address(_usds) != address(0), "_usds cannot be address(0)"
      ↪ );
   require( address(_managedTeamWallet) != address(0), "
      ↪ _managedTeamWallet cannot be address(0)" );
```

```
58 require( address(dao) == address(0), "setContracts can only be
      ↪ called once" );
```

```
60 require( address(_dao) != address(0), "_dao cannot be address(0)" );
   require( address(_upkeep) != address(0), "_upkeep cannot be address
      ↪ (0)" );
   require( address(_initialDistribution) != address(0), "
      ↪ _initialDistribution cannot be address(0)" );
   require( address(_airdrop) != address(0), "_airdrop cannot be
      ↪ address(0)" );
   require( address(_teamVestingWallet) != address(0), "
      ↪ _teamVestingWallet cannot be address(0)" );
   require( address(_daoVestingWallet) != address(0), "
      ↪ _daoVestingWallet cannot be address(0)" );
```

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