Report

v. 1.0

Customer

Chi

Smart Contract Audit

Chi Protocol

8th December 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	08.12.23	A. Zveryanskaya	Initial Draft
0.2	08.12.23	A. Zveryanskaya	Minor revision
1.0	08.12.23	A. Zveryanskaya	Release

2 Introduction

All modifications to this document are prohibited. Violators will be prosecuted to the full extent of the U.S. law.

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.

Chi is an Ethereum-based protocol issuing a decentralized and capital-efficient stablecoin, known as USC, designed to bring stability, scalability and sustainable economic incentives to the world of decentralized finance. USC is the first stablecoin issued by Chi Protocol. LSTs are used as collateral to back it, and it relies on a dual stability mechanism to maintain the price at $1.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/			
	Arbitrage.sol	ReserveHolder.sol	
common/			
	Mintable.sol		
dso/			
	LPRewards.sol	OCHI.sol	PoolHelper.sol
interfaces/			
	IArbitrage.sol	IArbitrageERC20.sol	IBurnableERC20.sol
	ICHI.sol	IChiLocking.sol	IChiStaking.sol
	IChiVesting.sol	ILPRewards.sol	IMintable.sol
	IMintableBurnable.sol	IMintableERC20.sol	IOCHI.sol
	IOracle.sol	IPriceFeedAggregator.sol	IReserveHolder.sol
	IRewardController.sol	IStaking.sol	IStakingWithEpochs.sol
	ISTETH.sol	ITimeWeighted Bonding.sol	IToken.sol
	IUniswapV2Twap Oracle.sol	IUSC.sol	IUSCStaking.sol
	IWETH.sol		
library/			
	ExternalContract Addresses.sol		

oracles/			
	ChainlinkOracle.sol	PriceFeedAggregator.sol	UniswapV2Twap Oracle.sol
sale/			
	TimeWeighted Bonding.sol		
staking/			
	ChiLocking.sol	ChiStaking.sol	ChiVesting.sol
	RewardController.sol	StakingWithEpochs.sol	USCStaking.sol
tokens/			
	CHI.sol	USC.sol	veCHI.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 15 major, and a few less important issues. All identified Major issues have been fixed.

Issues	Info	Fixed
Severity		
Critical	**0**	**0**
Major	**0**	**15**
Moderate	**8**	**9**
Minor	**52**	**97**

Fixed 121 out of 181 issues

6 Major Issues

CVF-2. FIXED

- **Category** Flaw
- **Source** ChiLocking.sol

Description There are no range checks for the arguments.

Recommendation Consider adding appropriate checks. For example, consider checking that both ”amount” and ”duration” a not zero.

112
```
function lockChi(address account, uint256 amount, uint256 duration)
   ↪ external onlyChiLockers {
```

CVF-3. FIXED

- **Category** Suboptimal
- **Source** ChiStaking.sol

Description This first transfers tokens to ”msg.sender” and then to this contract. Such approach seems cumbersome and inefficient.

Recommendation Consider refactoring.

29
```
function setChiLocking(IChiLocking _chiLocking) external onlyOwner {
```

35
```
function setRewardController(address _rewardController) external
   ↪ onlyOwner {
```

80
```
    unstake(amount);
    stakeToken.safeTransferFrom(msg.sender, address(chiLocking),
       ↪ amount);
```

CVF-4. FIXED

- **Category** Suboptimal
- **Source** ChiStaking.sol

Description These checks are redundant, as they a superseded by the final return statement.

Recommendation Consider removing these checks.

```
106 if (token == RewardToken.USC) return false;
    if (token == RewardToken.CHI) return false;
```

CVF-5. FIXED

- **Category** Suboptimal
- **Source** StakingWithEpochs.sol

Description The second part of the condition is redundant, as ”currentEpoch” cannot equal to ”stakeData.lastUpdateEpoch” due to a check above, and also ”currentEpoch” cannot be less than ”stakeData.lastUpdateEpoch”.

Recommendation Consider removing the second part of the condition.

```
122 if (stakeData.addSharesNextEpoch > 0 && currentEpoch > stakeData.
       ↪ lastUpdatedEpoch) {
```

CVF-6. FIXED

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

Description Hardcoding the number of reward tokens is error prone.

Recommendation Consider using ”type(RewardToken).min” and ”type(RewardToken).max” instead.

```
142 for (uint8 i = 0; i < 3; i++) {
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** UniswapV2TwapOracle.sol

Recommendation As the number of decimals for a chainlink feed isn't supposed to change over time, it would be more efficient to query it once in the constructor and storing in an immutable variable.

```
83 return quoteTokenChainlinkFeed.decimals();
```

CVF-10. FIXED

- **Category** Documentation
- **Source** IOCHI.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the arguments and/or documenting.

```
36 function mint(uint256, uint256, uint256, uint64) external;
```

```
38 function burn(uint256) external;
```

```
42 function claimRewards(uint256) external;
```

```
44 function getUnclaimedRewardsValue(uint256) external view returns (
   ↪ int256);
```

CVF-11. FIXED

- **Category** Documentation
- **Source** IChiLocking.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the arguments and/or documenting.

```
43 function lockChi(address, uint256, uint256) external;

45 function updateEpoch(uint256, uint256) external;

47 function claimStETH(address) external returns (uint256);

49 function unclaimedStETHAmount(address) external view returns (
   ↪ uint256);

55 function getVotingPower(address) external view returns (uint256);
```

CVF-12. FIXED

- **Category** Documentation
- **Source** IRewardController.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the arguments and/or documenting.

```
31 function rewardUSC(uint256) external;

37 function unclaimedStETHAmount(address) external view returns (
   ↪ uint256);
```

CVF-13. FIXED

- **Category** Documentation
- **Source** IUSCStaking.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the arguments and/or documenting.

```
16 function updateEpoch(uint256, uint256, uint256) external;
```

```
18 function lockChi(uint256) external;
```

CVF-15. FIXED

- **Category** Suboptimal
- **Source** OCHI.sol

Description The value ”lpRewards[uscEthPair]” is read from the storage twice.

Recommendation Consider reading once and reusing.

```
105 IERC20(address(uscEthPair)).safeTransferFrom(msg.sender, address(
    ↪ lpRewards[uscEthPair]), uscEthPairAmount);
```

```
108 lpRewards[uscEthPair].lockLP(tokenId, uscEthPairAmount, uint64(
    ↪ lockPeriodInEpochs));
```

CVF-16. FIXED

- **Category** Suboptimal
- **Source** OCHI.sol

Description The value ”lpRewards[chiEthPair]” is read from the storage twice.

Recommendation Consider reading once and reusing.

```
106 IERC20(address(chiEthPair)).safeTransferFrom(msg.sender, address(
    ↪ lpRewards[chiEthPair]), chiEthPairAmount);
```

```
109 lpRewards[chiEthPair].lockLP(tokenId, chiEthPairAmount, uint64(
    ↪ lockPeriodInEpochs));
```

CVF-17. FIXED

- **Category** Suboptimal
- **Source** OCHI.sol

Description The value ”options[tokenId].amount” is read from the storage twice.

Recommendation Consider reading once and reusing.

```
137 chi.safeTransfer(msg.sender, options[tokenId].amount);
```

```
140 emit Burn(msg.sender, tokenId, options[tokenId].amount);
```

CVF-19. FIXED

- **Category** Procedural
- **Source** Arbitrage.sol

Recommendation This constant value should be precomputed.

```
426 uint256 f = Math.mulDiv(1e18, (MAX_FEE - POOL_FEE), MAX_FEE);
```

CVF-20. FIXED

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

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

Recommendation Consider using the ”mulDiv” function.

```
111 return (totalStEthDeposited * stEthPrice + ethBalance * ethPrice) /
    ↪ 1e18;
```

7 Moderate Issues

CVF-1. INFO

- **Category** Suboptimal
- **Source** ChiVesting.sol

Recommendation While the ”mulDiv” function is very efficient in generic case, for specific cases better approaches do exit. For example, when the denominator is a compile-time constant, its modular reciprocal could be precomputed. If the denominator fits into 128 bits, the following approach could be used: https://medium.com/coinmonks/math-in-solidity-art-3-percents-and-proportions-4db014e080b1#4821

Client Comment *We agree with this, but it's not very important optimization, and will decrease the readability and understanding of the code. We'll have it in mind for version 2.*

```
93  totalVotingPower += Math.mulDiv(chiAmount, _epochsUntilEnd(),
        ↪ MAX_LOCK_DURATION);

116   totalVotingPower += Math.mulDiv(chiEmissions, _epochsUntilEnd(),
          ↪ MAX_LOCK_DURATION);

135   uint256 decreaseVotingPower = Math.mulDiv(amountToUnlock,
          ↪ epochsUntilEnd, MAX_LOCK_DURATION);

192 totalAmount += Math.mulDiv(rewardPerShare, vesting.shares, 1e18);

223 return Math.mulDiv(availableWithdrawPerShare, shares, 1e18);
```

CVF-7. INFO

- **Category** Suboptimal
- **Source** StakingWithEpochs.sol

Recommendation While the ”mulDiv” function is very efficient in generic case, for specific cases better approaches do exist. For example, when the denominator is a compile-time constant, its modular reciprocal could be precomputed. Also, when the denominator fits into 128 bits, the following approach could be used: https://medium.com/coinmonks/math-n-solidity-part-3-percents-and-proportions-4db014e080b1#4821

Client Comment *We agree with this, but it's not very important optimization, and will decrease the readability and understanding of the code. We'll have it in mind for version 2.*

145
```
stakeData.unclaimedRewards[token] += Math.mulDiv(
  toEpoch.cumulativeRewardsPerShare[token] - fromEpoch.
    ↪ cumulativeRewardsPerShare[token],
  shares,
  1e18
);
```

CVF-9. INFO

- **Category** Overflow/Underflow
- **Source** UniswapV2TwapOracle.sol

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

Client Comment *We agree with this, but currently it's not important given the context and practical maximum of those variables. We'll have it in mind for version 2.*

153
```
uint256 priceInUSD = (uint256(quoteTokenPriceInUSD) * quotedAmount)
  ↪ / 1 ether; // quote token price will be always greater than 0,
  ↪  so it's ok to convert int to uint
```

CVF-14. INFO

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

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

Recommendation Consider using the ”mulDiv” function.

Client Comment *We agree with this, but currently it’s not important given the context and practical maximum of those variables. We’ll have it in mind for version 2.*

```
108 int256 totalProfit = (int256(currentLPValue) - int256(prevLPValue))
    ↪ * int256(epochMinLPBalance);
```

```
143 int256 totalUSDreward = (int256(position.amountLocked) * profitDelta
    ↪ ) / int256(10 ** DECIMALS);
```

CVF-18. INFO

- **Category** Suboptimal
- **Source** Arbitrage.sol

Recommendation While the ”mulDiv” function is very efficient in generic case, for specific cases better approaches do exist. For example, when the denominator is a compile-time constant, its modular reciprocal could be precomputed. Also, if the denominator fits into 128 bits, the following approach could be used: https://medium.com/coinmonks/math-in-olidity-part-3-percents-and-proportions-4db014e080b1#4821

Client Comment *We agree with this, but it's not very important optimization, and will decrease the readability and understanding of the code. We'll have it in mind for version 2.*

```
340 uint256 chiArbitrageReward = Math.mulDiv(chiReceived, discount,
      ↪ BASE_PRICE);
```

```
356 uint256 chiArbitrageReward = Math.mulDiv(chiToCoverEth, discount,
      ↪ BASE_PRICE);
```

```
396 uint256 maxPriceDiff = Math.mulDiv(uscPrice, priceTolerance,
      ↪ MAX_PRICE_TOLERANCE);
```

```
426 uint256 f = Math.mulDiv(1e18, (MAX_FEE - POOL_FEE), MAX_FEE);
```

```
429 uint256 b = Math.mulDiv(reserveIn, 1e18 + f, 1e18);
430 uint256 b_sqr = Math.mulDiv(b, b, 1e18);
```

```
433 uint256 c_1 = Math.mulDiv(reserveIn, reserveIn, 1e18);
```

```
441   uint256 d = Math.mulDiv(4 * f, c, 1e18);
```

```
449   uint256 d = Math.mulDiv(4 * f, c, 1e18);
```

```
522 return Math.mulDiv(amount, price, 1e18);
```

CVF-21. INFO

- **Category** Suboptimal
- **Source** ChiLocking.sol

Description This loop doesn't scale and could exceed the block gas limit.

Recommendation Consider maintaining an aggregated value in the storage.

Client Comment *It's not possible to change given the logic of calculations. We already have removing unneeded locking positions.*

```
102 for (uint256 i = 0; i < lockData.positions.length; i++) {
```

```
263 for (uint256 i = 0; i < lockData.positions.length; i++) {
```

```
273 for (uint256 i = 0; i < lockData.positions.length; i++) {
```

CVF-22. FIXED

- **Category** Suboptimal
- **Source** PriceFeedAggregator.sol

Description Performing a separate external call to the "decimals" function is gas consuming.

Recommendation Consider either querying the number of decimals for a feed once in the "setPriceFeed" function and storing along with the feed address (in the same storage slot) or modifying the "IOracle" API to return the number of decimals along with the current price from the "peek" function.

```
35 return (priceFeeds[base].peek(), priceFeeds[base].decimals());
```

CVF-23. FIXED

- **Category** Unclear behavior
- **Source** UniswapV2TwapOracle.sol

Recommendation The hardcoded denominator "1 ether" should be replaced with a variable, probably with "baseAmount".

```
153 uint256 priceInUSD = (uint256(quoteTokenPriceInUSD) * quotedAmount)
    ↪ / 1 ether; // quote token price will be always greater than 0,
    ↪  so it's ok to convert int to uint
```

CVF-24. FIXED

- **Category** Flaw
- **Source** LPRewards.sol

Description As zero locked amount has a special meaning ”position doesn’t exist”, there should be a check to ensure that ”amount” is not zero.

Recommendation Consider adding such a check.

56
```
if (position.amountLocked != 0) {
```

CVF-25. FIXED

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

Description Overflow is possible here.

Recommendation Consider using safe conversion.

61
```
epochData[nextEpoch].totalDeltaAmountLocked += int256(amount);
epochData[nextEpoch + epochDuration].totalDeltaAmountLocked -=
   ↪ int256(amount);
```

CVF-26. FIXED

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

Description The returned value is ignored.

Recommendation Consider properly checking the returned value or using the ”safeTransfer’ function.

87
```
lpToken.transfer(address(lpToken), lpTokensToBurn);
```

128
```
lpToken.transfer(receiver, amount);
```

CVF-27. FIXED

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

Description Overflow is possible when converting types.

Recommendation Consider using safe conversion.

```
108 int256 totalProfit = (int256(currentLPValue) - int256(prevLPValue))
    ↪ * int256(epochMinLPBalance);
```

```
143 int256 totalUSDreward = (int256(position.amountLocked) * profitDelta
    ↪ ) / int256(10 ** DECIMALS);
```

CVF-28. FIXED

- **Category** Suboptimal
- **Source** OCHI.sol

Description This sounds too restrictive. It could make perfect sense to execute an out of the money physically settled option. For example, when the spot market is volatile and the oracle price is outdated. Or when the market is illiquid and it is not possible to buy the desired amount without significant price slippage.

```
117 /// @notice Executing option should revert if option strike price is
    ↪  above market price
```

CVF-29. FIXED

- **Category** Suboptimal
- **Source** OCHI.sol

Description It is possible that the next epoch already finished as well.

Recommendation Consider either calculating the new ”currentEpoch” value as ”(block-Timstamp - firstEpochTime) / EPOCH_DURATION + 1” or clearly explaining why it should be always incremented by one.

```
152 currentEpoch++;
```

CVF-30. FIXED

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

Description The returned value is ignored.

Recommendation Consider properly checking the returned value or using the ”safeTransfer” function.

```
89  IERC20(WETH).transfer(address(reserveHolder), ethAmount);

207   IERC20(WETH).transfer(address(reserveHolder), ethAmountForReserves
        ↪ );

217 IERC20(WETH).transfer(msg.sender, rewardAmount);

236 IERC20(WETH).transfer(address(reserveHolder), deltaInETH);

240 IERC20(WETH).transfer(msg.sender, rewardAmount);

344 IERC20(CHI).transfer(msg.sender, chiArbitrageReward);

361 IERC20(WETH).transfer(address(reserveHolder), ethReceived);

363 IERC20(CHI).transfer(msg.sender, chiArbitrageReward);
```

CVF-31. INFO

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

Description The returned value is ignored.

Recommendation Consider properly checking the returned value or using the ”safeApprove” function.

Client Comment *It’s not important given that token contracts we are using always returns ‘true‘ on approve. Even if there is some mistake in approval, the following transfer would revert which is expected behaviour. ‘safeApprove‘ is depricated, and is behaving differently of what we want - we don’t want to revert if there is already allowance bigger than 0.*

```
504 IERC20(tokenIn).approve(address(swapRouter), amount);
```

CVF-32. INFO

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

Description The returned value is ignored.

Recommendation Consider properly checking the returned value or using the ”safeApprove” function.

Client Comment *It’s not important given that token contracts we are using always returns ‘true‘ on approve. Even if there is some mistake in approval, the following transfer would revert which is expected behaviour. ‘safeApprove‘ is depricated, and is behaving differently of what we want - we don’t want to revert if there is already allowance bigger than 0.*

```
209 IERC20(assetIn).approve(address(uniswapRouter), amountIn);
```

```
226 stETH.approve(address(uniswapRouter), stEthAmountIn);
```

8 Minor Issues

CVF-33. FIXED

- **Category** Procedural
- **Source**

Recommendation Consider specifying as ”^0.8.0” unless there is something special regarding this particular version. Also relevant for: CHI.sol, USC.sol, veCHI.sol, ChiStaking.sol, StakingWithEpochs.sol, ChiLocking.sol, RewardController.sol, USCStaking.sol, TimeWeightedBonding.sol, ChainlinkOracle.sol, PriceFeedAggregator.sol, UniswapV2TwapOracle.sol, ExternalContractAddresses.sol, IMintableBurnable.sol, IArbitrage.sol, IArbitrageERC20.sol, IBurnableERC20.sol, IToken.sol, ICHI.sol, IOCHI.sol, IChiLocking.sol, IStakingWithEpochs.sol, IStaking.sol, IChiStaking.sol, IMintableERC20.sol, IOracle.sol, IReserveHolder.sol, IRewardController.sol, ISTETH.sol, ITimeWeightedBonding.sol, IUSC.sol, IUSCStaking.sol, IWETH.sol, IUniswapV2TwapOracle.sol, PoolHelper.sol, OCHI.sol, Arbitrage.sol, ReserveHolder.sol.

```
2 pragma solidity ^0.8.20;
```

CVF-34. FIXED

- **Category** Suboptimal
- **Source** CHI.sol

Recommendation It would be much more efficient to just return ”CHI” here.

```
17 return super.name();
```

```
22 return super.symbol();
```

CVF-35. FIXED

- **Category** Suboptimal
- **Source** CHI.sol

Recommendation It would be much more efficient to just return 18 here.

```
27 return super.decimals();
```

CVF-36. FIXED

- **Category** Suboptimal
- **Source** USC.sol

Recommendation It would be much more efficient to just return "USC" here.

```
15 return super.name();
```

```
20 return super.symbol();
```

CVF-37. FIXED

- **Category** Suboptimal
- **Source** USC.sol

Recommendation It would be much more efficient to just return 18 here.

```
25 return super.decimals();
```

CVF-38. FIXED

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

Recommendation These variables should be declared as immutable.

```
14 IChiLocking public chiLocking;
   IChiVesting public chiVesting;
```

CVF-39. INFO

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

Recommendation The type for this variable should be "IRewardController".

Client Comment *We won't change this as we are not calling external functions, we are using it only as address for modifier.*

```
27 address public rewardController;
```

CVF-40. INFO

- **Category** Documentation
- **Source** ChiVesting.sol

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

Recommendation Consider documenting.

```
29 mapping(address => bool) public chiVesters;
30 mapping(address => VestingData) public vestingData;
```

CVF-41. FIXED

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

Description These functions should emit some events.

```
58 function setRewardController(address _rewardController) external
   ↪ onlyOwner {
```

```
65 function setChiVester(address contractAddress, bool toSet) external
   ↪ onlyOwner {
```

CVF-42. INFO

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

Recommendation The argument type should be ”IRewardController”.

Client Comment *We won’t change this as we are not calling external functions, we are using it only as address for modifier.*

```
58 function setRewardController(address _rewardController) external
   ↪ onlyOwner {
```

CVF-43. INFO

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

Recommendation The value ”1e18” should be a named constant.

Client Comment *Currently it’s more understandable this way, we will consider refactoring for version 2.*

```
107   stETHRewardPerShare = Math.mulDiv(stETHrewards, 1e18, totalShares)
         ↪ ;
```

```
126   amountUnlockedPerShare = Math.mulDiv(amountToUnlock, 1e18,
         ↪ totalShares);
```

```
192 totalAmount += Math.mulDiv(rewardPerShare, vesting.shares, 1e18);
```

```
223 return Math.mulDiv(availableWithdrawPerShare, shares, 1e18);
```

CVF-44. FIXED

- **Category** Procedural
- **Source** ChiVesting.sol

Recommendation This commented out code should be removed.

```
204 // uint256 lockedAmount = vesting.startAmount -
       ↪ _availableUnlockFromTo(0, currentEpoch - 1, vesting.shares);
    // return Math.mulDiv(lockedAmount, _epochsUntilEnd(),
       ↪ MAX_LOCK_DURATION);
```

CVF-45. FIXED

- **Category** Documentation
- **Source** ChiLocking.sol

Description The units are unclear.

Recommendation Consider documenting.

```
17 uint256 public constant MAX_LOCK_DURATION = 208;
```

CVF-46. INFO

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

Recommendation The type for this variable should be ”IRewardController”.

Client Comment *We won’t change this as we are not calling external functions, we are using it only as address for modifier.*

```
21 address public rewardController;
```

CVF-47. FIXED

- **Category** Documentation
- **Source** ChiLocking.sol

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

Recommendation Consider documenting.

```
33 mapping(address => bool) public chiLockers;
   mapping(address => LockingData) public locks;
```

CVF-48. INFO

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

Recommendation The type of the ”_chiStaking” argument should be ”IChiStaking”.

Client Comment *We won’t change this as we are not calling external functions, we are using it only as address for modifier.*

```
51 function initialize(IERC20 _chi, address _chiStaking) external
   ↪ initializer {
```

CVF-49. INFO

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

Recommendation The argument type should be ”IUSCStaking”.

Client Comment *We won’t change this as we are not calling external functions, we are using it only as address for modifier.*

```
60 function setUscStaking(address _uscStaking) external onlyOwner {
```

CVF-50. FIXED

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

Description These functions should emit some events.

```
60 function setUscStaking(address _uscStaking) external onlyOwner {
```

```
66 function setRewardController(address _rewardController) external
   ↪ onlyOwner {
```

```
73 function setChiLocker(address contractAddress, bool toSet) external
   ↪ onlyOwner {
```

CVF-51. INFO

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

Recommendation The argument type should be ”IRewardController”.

Client Comment *We won’t change this as we are not calling external functions, we are using it only as address for modifier.*

```
66 function setRewardController(address _rewardController) external
   ↪ onlyOwner {
```

CVF-52. FIXED

- **Category** Suboptimal
- **Source** ChiLocking.sol

Description The expression ”epochs[fromEpoch]” is calculated twice.

Recommendation Consider calculating once and reusing.

```
123 epochs[fromEpoch].lockedSharesInEpoch += shares;
    epochs[fromEpoch].totalLockedChiInEpoch += amount;
```

CVF-53. FIXED

- **Category** Suboptimal
- **Source** ChiLocking.sol

Description The expression ”epochs[fromEpoch + duration]” is calculated twice.

Recommendation Consider calculating once and reusing.

```
125 epochs[fromEpoch + duration].sharesToUnlock += shares;
```

```
132 epochs[fromEpoch + duration].numberOfEndingPositions += 1;
```

CVF-54. INFO

- **Category** Suboptimal
- **Source** ChiLocking.sol

Description Skipping locked positions could consume lots of gas.

Recommendation Consider implementing some way for the caller to specify what position to start iterating at.

Client Comment *Not possible to change due to calculation logic. We don’t expect user to have many locking positions, and will consider refactoring and optimization for version 2.*

```
233 if (currentEpoch < position.startEpoch + position.duration) {
      pos++;
```

CVF-55. INFO

- **Category** Suboptimal
- **Source** ChiLocking.sol

Recommendation While the ”mulDiv” function is very efficient in generic case, for specific cases better approaches do exist. For example, when the denominator is a compile-time constant, its modular reciprocal could be precomputed. Also, if the denominator fits into 128 bits, the following approach could be used: https://medium.com/coinmonks/math-in-olidity-part-3-percents-and-proportions-4db014e080b1#4821

Client Comment *We agree with this, but it's not very important optimization, and will decrease the readability and understanding of the code. We'll have it in mind for version 2.*

```
321 return Math.mulDiv(rewardPerShare, position.shares, 1e18);
```

```
332 return Math.mulDiv(rewardPerChi, unlockedChiAmount, 1e18);
```

CVF-56. FIXED

- **Category** Documentation
- **Source** ChiStaking.sol

Description Units are unclear.

Recommendation Consider documenting.

```
17 uint256 public constant MIN_LOCK_DURATION = 4;
   uint256 public constant MAX_LOCK_DURATION = 208;
```

CVF-57. FIXED

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

Description These functions should emit some events.

```
29 function setChiLocking(IChiLocking _chiLocking) external onlyOwner {
```

```
35 function setRewardController(address _rewardController) external
   ↪ onlyOwner {
```

CVF-58. INFO

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

Recommendation The argument type should be ”IRewardController”.

Client Comment *We won’t change this as we are not calling external functions, we are using it only as address for modifier.*

```
35 function setRewardController(address _rewardController) external
   ↪ onlyOwner {
```

CVF-59. INFO

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

Recommendation The type for this variable should be ”IRewardController”.

Client Comment *We won’t change this as we are not calling external functions, we are using it only as address for modifier.*

```
22 address public rewardController;
```

CVF-60. FIXED

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

Recommendation The semantics of the keys is unclear. Consider documenting.

```
25 mapping(address => StakeData) public stakes;
```

CVF-61. FIXED

- **Category** Suboptimal
- **Source** StakingWithEpochs.sol

Description The value ”stakeData.addSharesNextEpoch” is read from the storage twice.

Recommendation Consider reading once and reusing.

```
91 if (stakeData.shares + stakeData.addSharesNextEpoch < amount) {
     revert AmountBelowStakedBalance(stakeData.shares + stakeData.
        ↪ addSharesNextEpoch, amount);
```

CVF-62. FIXED

- **Category** Suboptimal
- **Source** StakingWithEpochs.sol

Description The value ”stakeData.addSharesNextEpoch” is read from the storage several times.

Recommendation Consider reading once and reusing.

```
97 if (stakeData.addSharesNextEpoch > amount) {
     stakeData.addSharesNextEpoch -= amount;
```

```
101   uint256 fromCurrentShares = amount - stakeData.addSharesNextEpoch;
```

```
104   epochs[currentEpoch + 1].shares -= stakeData.addSharesNextEpoch;
```

CVF-63. INFO

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

Recommendation This value should be a named constant.

Client Comment *Currently it’s more understandable this way, we will consider refactoring for version 2.*

```
148 1e18
```

CVF-64. INFO

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

Recommendation The value ”1e18” should be a named constant.

Client Comment *Currently it’s more understandable this way, we will consider refactoring for version 2.*

```
164 Math.mulDiv(amount, 1e18, epoch.shares);
```

CVF-65. FIXED

- **Category** Procedural
- **Source** StakingWithEpochs.sol

Recommendation This TODO should be resolved or removed.

```
179 // TODO: make common function with _updateUnclaimedRewards
```

CVF-66. INFO

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

Recommendation The argument type should be ”IRewardController”.

Client Comment *We won’t change this as we are not calling external functions, we are using it only as address for modifier.*

```
200 function _setRewardController(address _rewardController) internal {
```

CVF-67. FIXED

- **Category** Documentation
- **Source** RewardController.sol

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

Recommendation Consider documenting.

```
41 mapping(uint256 => EpochData) public epochs;
```

CVF-68. FIXED

- **Category** Suboptimal
- **Source** RewardController.sol

Recommendation This value should be a named constant or even an immutable variable passed as a constructor argument.

Client Comment *We will consider refactoring for version 2.*

```
71 chiIncentivesPerEpoch = 38461 * 1 ether;
```

CVF-69. FIXED

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

Description These functions should emit some events.

```
76 function setChiIncentivesPerEpoch(uint256 _chiIncentivesPerEpoch)
      ↪ external onlyOwner {
```

```
82 function setArbitrager(IArbitrage _arbitrager) external onlyOwner {
```

CVF-70. FIXED

- **Category** Documentation
- **Source** USCStaking.sol

Description Units are unclear.

Recommendation Consider documenting.

```
18 uint256 public constant MIN_LOCK_DURATION = 4;
   uint256 public constant MAX_LOCK_DURATION = 208;
```

CVF-71. INFO

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

Recommendation The argument type should be ”IRewardController”.

Client Comment *We won’t change this as we are not calling external functions, we are using it only as address for modifier.*

```
35 function setRewardController(address _rewardController) external
      ↪ onlyOwner {
```

CVF-72. FIXED

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

Recommendation The units are unclear. Consider documenting.

```
20 uint256 public constant MAX_LOCK_PERIOD = 208;
```

CVF-73. INFO

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

Recommendation The type for this variable should be ”IWETH9”.

Client Comment *Not important here, we are casting when calling external functions.*

```
22 address public constant WETH = ExternalContractAddresses.WETH;
```

CVF-74. FIXED

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

Recommendation This function should emit come event.

```
48 function setCliffTimestampEnd(uint256 _cliffTimestampEnd) external
      ↪ onlyOwner {
```

CVF-75. INFO

- **Category** Suboptimal
- **Source** TimeWeightedBonding.sol

Recommendation While the ”mulDiv” function is very efficient in generic case, for specific cases better approaches do exist. For example, when the denominator is a compile-time constant, its modular reciprocal could be precomputed. Also, when the denominator fits into 128 bits, the following approach could be used: https://medium.com/coinmonks/math-n-solidity-part-3-percents-and-proportions-4db014e080b1#4821

Client Comment *We agree with this, but it's not very important optimization, and will decrease the readability and understanding of the code. We'll have it in mind for version 2.*

```
88 return Math.mulDiv(_peek(address(chi)), BASE_PRICE - discount,
      ↪ BASE_PRICE);
```

CVF-76. INFO

- **Category** Suboptimal
- **Source** TimeWeightedBonding.sol

Recommendation Consider including the original revert reason into the error.

Client Comment *It is descriptive enough this way*

```
100 revert EtherSendFailed(to, value);
```

CVF-77. INFO

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

Recommendation The type for this variable should be ”IERC20”.

Client Comment *Not important here, we are casting when calling external functions.*

```
13 address public immutable baseToken;
```

CVF-78. INFO

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

Recommendation The arguments types should be ”IERC20” and ”AggregatorV3Interface” respectively.

Client Comment *Not important here, we are casting when calling external functions.*

```
16 constructor(address _baseToken, address _chainlinkFeed) {
```

CVF-79. FIXED

- **Category** Suboptimal
- **Source** ChainlinkOracle.sol

Recommendation As the number of decimals for a chainlink oracle isn’t supposed to change over time, it would be more efficient to query it once in the constructor and store in an immutable variable.

```
28 return chainlinkFeed.decimals();
```

CVF-80. FIXED

- **Category** Suboptimal
- **Source** ChainlinkOracle.sol

Recommendation Consider adding an explicit assert for this assumption.

```
34 return uint256(priceInUSD); // ETH and stETH prices will be always
     ↪ greater than 0
```

CVF-81. FIXED

- **Category** Documentation
- **Source** PriceFeedAggregator.sol

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

Recommendation Consider documenting,

```
12 mapping(address => IOracle) public priceFeeds;
```

CVF-82. INFO

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

Recommendation The key type should be ”IERC20”.

Client Comment *Not important here, we are casting when calling external functions.*

```
12 mapping(address => IOracle) public priceFeeds;
```

CVF-83. INFO

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

Recommendation The arguments types should be ”IERC20” and ”IOracle” respectively.

Client Comment *Not important here, we are casting when calling external functions.*

```
17 function setPriceFeed(address base, address feed) external onlyOwner
   ↪ {
```

CVF-84. INFO

- **Category** Procedural
- **Source** UniswapV2TwapOracle.sol

Description We didn’t review these files.

Client Comment *Those are libraries written by Uniswap.*

```
9  import "contracts/uniswap/libraries/FixedPoint.sol";
10 import "contracts/uniswap/libraries/UniswapV2OracleLibrary.sol";
   import "contracts/uniswap/libraries/UniswapV2Library.sol";
```

CVF-85. INFO

- **Category** Bad datatype
- **Source** UniswapV2TwapOracle.sol

Recommendation The type for these variables should be ’IERC20”.

Client Comment *Not important here, we are casting when calling external functions.*

```
26 address public immutable baseToken;
```

```
29 address public immutable token0;
30 address public immutable token1;
```

CVF-86. INFO

- **Category** Bad datatype
- **Source** UniswapV2TwapOracle.sol

Recommendation The type for this argument should be ”IUniswapV2Factory”.

Client Comment *Not important here, we are casting when calling external functions.*

```
36 address _factory,
```

CVF-87. INFO

- **Category** Bad datatype
- **Source** UniswapV2TwapOracle.sol

Recommendation The type for these arguments should be ”IERC20”.

Client Comment *Not important here, we are casting when calling external functions.*

```
37 address _baseToken,
   address _quoteToken,
```

CVF-88. INFO

- **Category** Suboptimal
- **Source** UniswapV2TwapOracle.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 *We need to know base value of token to return price for 1 whole amount.*

```
44 baseAmount = uint128(10 ** (IERC20Metadata(baseToken).decimals()));
```

CVF-89. INFO

- **Category** Bad datatype
- **Source** UniswapV2TwapOracle.sol

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

Client Comment *Not important here, we are casting when calling external functions.*

```
112 function getTwapQuote(address token, uint256 amountIn) public view
    ↪ returns (uint256 amountOut) {
```

CVF-90. FIXED

- **Category** Documentation
- **Source** UniswapV2TwapOracle.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

```
149 function peek() external view returns (uint256) {
```

CVF-91. INFO

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

Recommendation The type of this constant should be ”IWETH9”.

Client Comment *Not important here, we are casting when calling external functions.*

```
6 address public constant WETH = 0
    ↪ xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
```

CVF-92. INFO

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

Recommendation The type of this constant should be ”ISTETH”.

Client Comment *Not important here, we are casting when calling external functions.*

```
7 address public constant stETH = 0
    ↪ xae7ab96520DE3A18E5e111B5EaAb095312D7fE84;
```

CVF-93. INFO

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

Recommendation The type of this constant should be ”IUniswapV2Router02”.

Client Comment *Not important here, we are casting when calling external functions.*

```
8 address public constant UNI_V2_SWAP_ROUTER = 0
    ↪ x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;
```

CVF-94. INFO

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

Recommendation The type of this constant should be ”IUniswapV2Factory”.

Client Comment *Not important here, we are casting when calling external functions.*

```
9 address public constant UNI_V2_POOL_FACTORY = 0
    ↪ x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;
```

CVF-95. INFO

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

Recommendation The type of this constant should be ”IUniswapV2Pair”.

Client Comment *Not important here, we are casting when calling external functions.*

```
10 address public constant UNI_ETH_STETH_PAIR = 0
    ↪ x4028DAAC072e492d34a3Afdbef0ba7e35D8b55C4;
```

CVF-96. INFO

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

Recommendation The type of these constants should be ”AggregatorV3Interface”.

Client Comment *Not important here, we are casting when calling external functions.*

```
11 address public constant ETH_USD_CHAINLINK_FEED = 0
    ↪ x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419;
address public constant STETH_USD_CHAINLINK_FEED = 0
    ↪ xCfE54B5cD566aB89272946F602D76Ea879CAb4a8;
```

CVF-97. FIXED

- **Category** Procedural
- **Source** IMintable.sol

Description Consider specifying as "^0.8.0" unless there is something special about this particular version.

Recommendation Also relevant for: ILPRewards.sol, IPriceFeedAggregator.sol, LPRewards.sol, Mintable.sol.

```
2 pragma solidity ^0.8.20;
```

CVF-98. FIXED

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

Recommendation Events are usually named via nouns, such as "MinterStatus".

Client Comment *It's not per our coding style.*

```
8 event UpdateMinter(address indexed _account, bool indexed _status);
```

CVF-99. FIXED

- **Category** Readability
- **Source** IMintable.sol

Recommendation Consider giving descriptive names to the arguments.

```
10 function isMinter(address) external view returns (bool);

12 function updateMinter(address, bool) external;
```

CVF-100. FIXED

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

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

Client Comment *It's not per our coding style.*

```
16 event LockLP(uint256 indexed lockingTokenId, uint256 amount, uint64
      ↪ currentEpoch, uint64 endingEpoch);
   event ClaimRewards(uint256 indexed lockingTokenId, address indexed
      ↪ account, uint256 amount, int256 rewardUSD);
   event UpdateEpoch(uint64 indexed epoch, uint256 lpValue, int256
      ↪ totalAmountLocked, int256 profitPerToken);
   event RecoverLPTokens(address indexed account, uint256 amount);
```

CVF-101. FIXED

- **Category** Readability
- **Source** ILPRewards.sol

Recommendation Consider giving a descriptive name to the argument.

```
33 function recoverLPTokens(address) external;
```

CVF-102. FIXED

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

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

Client Comment *It's not per our coding style.*

```
5 event SetPriceFeed(address indexed base, address indexed feed);
```

CVF-103. FIXED

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

Recommendation The type of the ”base” parameter should be ”IERC20”.

```
5 event SetPriceFeed(address indexed base, address indexed feed);
```

CVF-104. FIXED

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

Recommendation The type of the ”feed” parameter should be ”IOracle”.

```
5 event SetPriceFeed(address indexed base, address indexed feed);
```

CVF-105. FIXED

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

Recommendation The arguments types should be ”IERC20” and ”IOracle” respectively.

Client Comment *Not important here, we are casting when calling external functions.*

```
9 function setPriceFeed(address, address) external;
```

CVF-106. FIXED

- **Category** Documentation
- **Source** IPriceFeedAggregator.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the arguments and/or documenting.

```
9 function setPriceFeed(address, address) external;
```

```
11 function peek(address) external view returns (uint256, uint8);
```

CVF-107. FIXED

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

Recommendation The argument type should be ”IERC20”.

Client Comment *Not important here, we are casting when calling external functions.*

```
11 function peek(address) external view returns (uint256, uint8);
```

CVF-108. FIXED

- **Category** Documentation
- **Source** IPriceFeedAggregator.sol

Description Semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or documenting.

```
11 function peek(address) external view returns (uint256, uint8);
```

CVF-109. FIXED

- **Category** Readability
- **Source** IMintableBurnable.sol

Recommendation Consider giving descriptive names to the arguments.

```
7 function mint(address, uint256) external;
```

```
9 function burnFrom(address, uint256) external;
```

CVF-110. FIXED

- **Category** Procedural
- **Source** IArbitrage.sol

Recommendation This interface should be moved into a separate file named ”IArbitrageEvents.sol”.

```
4 interface IArbitrageEvents {
```

CVF-111. FIXED

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

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

```
6 event ExecuteArbitrage(
```

CVF-112. FIXED

- **Category** Documentation
- **Source** IArbitrage.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or documenting.

```
22 function mint() external payable returns (uint256);
```

```
24 function executeArbitrage() external returns (uint256);
```

CVF-113. FIXED

- **Category** Documentation
- **Source** IArbitrage.sol

Description The number format of the argument is unclear.

Recommendation Consider documenting.

```
26 function setPriceTolerance(uint16 _priceTolerance) external;
```

CVF-114. FIXED

- **Category** Documentation
- **Source** IBurnableERC20.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the arguments and/or documenting.

```
5 function burnFrom(address, uint256) external;
```

CVF-115. FIXED

- **Category** Readability
- **Source** IToken.sol

Recommendation Consider giving descriptive names to the arguments.

```
14 function mint(address, uint256) external;
```

```
16 function burn(uint256) external;
```

CVF-116. FIXED

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

Description In the name of this interface ”CHI” is in upper case, while in other names it is ”Chi”.

Recommendation Consider using consistent naming.

```
6 interface ICHI is IToken {
```

CVF-117. FIXED

- **Category** Documentation
- **Source** ICHI.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the arguments and/or documenting.

```
7 function burnFrom(address, uint256) external;
```

CVF-118. FIXED

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

Description In the name of this interface ”CHI” is in upper case, while in other names it is ”Chi”.

Recommendation Consider using consistent naming.

```
4 interface IOCHI {
```

CVF-119. FIXED

- **Category** Documentation
- **Source** IOCHI.sol

Description The number format of these fields is unclear.

Recommendation Consider documenting.

```
7  uint256 strikePrice;
20 uint256 strikePrice,
```

CVF-120. FIXED

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

Recommendation Events are usually named via nouns, such as ”Epoch” or ”RewardsOCHI-Claim”.

Client Comment *It’s not per our coding style.*

```
24 event UpdateEpoch(uint64 indexed epoch, uint256 timestamp);
   event ClaimRewardsOCHI(uint256 indexed tokenId);
   event RecoverLPTokens();
```

CVF-121. FIXED

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

Recommendation Events are usually named via nouns, such as ”Lock”, ”Epoch” etc.

Client Comment *It’s not per our coding style.*

```
28 event LockChi(address indexed account, uint256 amount, uint256
      ↪ shares, uint256 startEpoch, uint256 endEpoch);
   event UpdateEpoch(
```

```
36 event ClaimStETH(address indexed account, uint256 amount);
   event WithdrawChiFromAccount(address indexed account, uint256 amount
      ↪ );
```

CVF-122. FIXED

- **Category** Documentation
- **Source** IChiLocking.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving descriptive name to the returned value.

```
47 function claimStETH(address) external returns (uint256);
```

CVF-123. FIXED

- **Category** Readability
- **Source** IStakingWithEpochs.sol

Recommendation Consider giving descriptive names to the arguments.

```
30 function stake(uint256) external;

32 function unstake(uint256) external;

34 function getUnclaimedRewards(address, RewardToken) external view
   ↪ returns (uint256);

36 function getCumulativeRewardsPerShare(uint256, RewardToken) external
   ↪  view returns (uint256);
```

CVF-124. FIXED

- **Category** Readability
- **Source** IStaking.sol

Recommendation Consider giving descriptive names to the arguments.

```
7  function setRewardController(address) external;

11 function claimStETH(address) external returns (uint256);

13 function unclaimedStETHAmount(address) external view returns (
   ↪ uint256);
```

CVF-125. FIXED

- **Category** Documentation
- **Source** IStaking.sol

Description The semantics of the returned values is unclear.

Recommendation Consider documenting.

```
11 function claimStETH(address) external returns (uint256);
```

CVF-126. FIXED

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

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

Client Comment *It’s not per our coding style.*

```
8 event ClaimStETH(address indexed account, uint256 amount);
```

CVF-127. FIXED

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

Recommendation Events are usually named via nouns, such as ”Vesting”, ”Epoch”, etc.

Client Comment *It’s not per our coding style.*

```
19 event AddVesting(address indexed account, uint256 amount, uint256
   ↪ shares);
20 event UpdateEpoch(uint256 indexed epoch, uint256 stETHrewards,
   ↪ uint256 totalLockedChi);
   event WithdrawChi(address indexed account, uint256 amount);
   event ClaimStETH(address indexed account, uint256 amount);
```

CVF-128. FIXED

- **Category** Documentation
- **Source** IChiVesting.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the arguments and/or documenting.

```
31 function addVesting(address, uint256) external;

33 function updateEpoch(uint256, uint256) external;

35 function claimStETH(address) external returns (uint256);

37 function unclaimedStETHAmount(address) external view returns (
   ↪ uint256);

41 function getVotingPower(address) external view returns (uint256);

45 function withdrawChi(uint256) external;

47 function availableChiWithdraw(address account) external view returns
   ↪  (uint256);
```

CVF-129. FIXED

- **Category** Documentation
- **Source** IChiVesting.sol

Description The semantics of the returned value is unclear.

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

```
35 function claimStETH(address) external returns (uint256);
```

CVF-130. FIXED

- **Category** Documentation
- **Source** IMintableERC20.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the arguments and/or documenting.

```
7 function mint(address, uint256) external;
```

CVF-131. FIXED

- **Category** Documentation
- **Source** IOracle.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

```
9 function peek() external view returns (uint256 answer);
```

CVF-132. FIXED

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

Recommendation Events are usually named via nouns, such as ”SwapRedemption”, ”RewardsClaim” etc.

Client Comment *It’s not per our coding style.*

```
8  event RedeemSwap(uint256 ethAmount, uint256 stEthAmount);
   event ClaimRewards(address indexed account, uint256 amount);
10 event Receive(address indexed account, uint256 amount);
```

CVF-133. FIXED

- **Category** Documentation
- **Source** IReserveHolder.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the returned values and/or documenting.

```
17 function deposit(uint256) external;

21 function redeem(uint256) external returns (uint256);

23 function claimRewards(address, uint256) external;

25 function setArbitrager(address, bool) external;

27 function setClaimer(address) external;

29 function setEthThreshold(uint256) external;

31 function setSwapEthTolerance(uint256) external;
```

CVF-134. FIXED

- **Category** Documentation
- **Source** IReserveHolder.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving descriptive name to the returned value and/or documenting.

```
21 function redeem(uint256) external returns (uint256);
```

CVF-135. FIXED

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

Recommendation Events are usually named via nouns, such as ”Epoch” or ”StETHClaim”.

Client Comment *It’s not per our coding style.*

```
24 event UpdateEpoch(uint256 indexed epoch, uint256 totalStEthReward,
      ↪ uint256 totalChiIncentives);
   event ClaimStEth(address indexed account, uint256 amount);
```

CVF-136. FIXED

- **Category** Documentation
- **Source** ISTETH.sol

Description The semantics of the argument is unclear.

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

```
7 function submit(address) external payable returns (uint256);
```

CVF-137. FIXED

- **Category** Documentation
- **Source** ISTETH.sol

Description The semantics of the returned value is unclear.

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

```
7 function submit(address) external payable returns (uint256);
```

CVF-138. FIXED

- **Category** Documentation
- **Source** ITimeWeightedBonding.sol

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

Client Comment *It’s not per our coding style.*

```
5 event RecoverChi(address indexed account, uint256 amount);
```

CVF-139. FIXED

- **Category** Documentation
- **Source** ITimeWeightedBonding.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the returned values and/or documenting.

```
11 function vest(address, uint256) external;
```

```
13 function buy(uint256) external payable;
```

```
15 function recoverChi(uint256) external;
```

CVF-140. FIXED

- **Category** Procedural
- **Source** IUSC.sol

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

```
6 interface IUSC is IToken {}
```

CVF-141. INFO

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

Recommendation Events are usually named via nouns, such as ”Epoch”, ”Lock”, etc.

Client Comment *It's not per our coding style.*

```
8  event UpdateEpoch(uint256 indexed epoch, uint256 chiEmissions,
       ↪ uint256 uscRewards, uint256 stETHrewards);
   event LockChi(address indexed account, uint256 amount, uint256
       ↪ duration);
10 event ClaimUSCRewards(address indexed account, uint256 amount);
   event ClaimStETH(address indexed account, uint256 amount);
```

CVF-142. FIXED

- **Category** Documentation
- **Source** IUniswapV2TwapOracle.sol

Description The number format of these fields is unclear.

Recommendation Consider documenting.

```
8  uint256 price0;
   uint256 price1;
```

CVF-143. INFO

- **Category** Bad naming
- **Source** IUniswapV2TwapOracle.sol

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

Client Comment *It's not per our coding style.*

```
13 event UpdateCumulativePricesSnapshot();
```

CVF-144. FIXED

- **Category** Documentation
- **Source** IUniswapV2TwapOracle.sol

Description The semantics of the arguments is unclear.

Recommendation Consider giving descriptive names to the arguments and/or documenting.

```
21 function getTwapQuote(address, uint256) external view returns (
   ↪ uint256);
```

CVF-145. FIXED

- **Category** Documentation
- **Source** IUniswapV2TwapOracle.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

```
21 function getTwapQuote(address, uint256) external view returns (
   ↪ uint256);
```

CVF-146. FIXED

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

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

Recommendation Consider using the ”mulDiv” function.

```
17 uint256 totalValue = (token0amount * price0 + token1amount * price1)
   ↪  / 10 ** 18;
```

```
26 return (getTotalPoolUSDValue(pair, priceFeedAggregator) * lpAmount)
   ↪ / pair.totalSupply();
```

CVF-147. FIXED

- **Category** Readability
- **Source** PoolHelper.sol

Recommendation The value ”10 ** 18” could be rendered as ”1e18”.

```
17 uint256 totalValue = (token0amount * price0 + token1amount * price1)
   ↪  / 10 ** 18;
```

CVF-148. FIXED

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

Recommendation The value ”10 ** 18” should be a named constant.

```
17 uint256 totalValue = (token0amount * price0 + token1amount * price1)
   ↪  / 10 ** 18;
```

CVF-149. FIXED

- **Category** Procedural
- **Source** LPRewards.sol

Description Variables are usually not named in UPPER_CASE.

Recommendation Consider renaming in camelCase.

```
15 uint8 public immutable DECIMALS;
```

CVF-150. INFO

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

Recommendation The type for this variable should be ”IOCHI”.

Client Comment *Not important here, we are casting when calling external functions.*

```
19 address public ochi;
```

CVF-151. FIXED

- **Category** Documentation
- **Source** LPRewards.sol

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

Recommendation Consider documenting.

```
26 mapping(uint256 => LockingTokenData) public lockingTokenData;
   mapping(uint256 => EpochData) public epochData;
```

CVF-152. FIXED

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

Recommendation This function should emit some event.

```
44 function setOCHI(address _ochi) external onlyOwner {
```

CVF-153. FIXED

- **Category** Procedural
- **Source** LPRewards.sol

Recommendation The value "10 ** DECIMALS" should be computed once inside the constructor and stored in an immutable variable.

```
106 currentLPValue = PoolHelper.getUSDValueForLP(10 ** DECIMALS, lpToken
    ↪ , priceFeedAggregator);
```

```
143 int256 totalUSDreward = (int256(position.amountLocked) * profitDelta
    ↪ ) / int256(10 ** DECIMALS);
```

CVF-154. FIXED

- **Category** Readability
- **Source** OCHI.sol

Recommendation This value could be rendered as "1e8".

```
27 uint256 public constant BASE_PRICE = 10 ** 8;
```

CVF-155. FIXED

- **Category** Readability
- **Source** OCHI.sol

Recommendation This value could be rendered as ”1e18”.

```
28 uint256 public constant MULTIPLIER = 10 ** 18;
```

CVF-156. INFO

- **Category** Readability
- **Source** OCHI.sol

Recommendation This value could be rendered as ”0.8e4”.

Client Comment *It is more readable this way*

```
29 uint256 public constant TARGET_RATIO = 80_00;
```

CVF-157. INFO

- **Category** Readability
- **Source** OCHI.sol

Recommendation This value could be rendered as ”1e4”.

Client Comment *It is more readable this way*

```
30 uint256 public constant DENOMINATOR = 100_00;
```

CVF-158. INFO

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

Recommendation The type for this constant should be ”IWETH9”.

Client Comment *Not important here, we are casting when calling external functions.*

```
31 address public constant WETH = ExternalContractAddresses.WETH;
```

CVF-159. FIXED

- **Category** Documentation
- **Source** OCHI.sol

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

Recommendation Consider documenting.

```
44 mapping(uint256 => ChiOption) public options;
```

CVF-160. FIXED

- **Category** Suboptimal
- **Source** OCHI.sol

Description The expression ”options[tokenId]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
124 if (currentEpoch < options[tokenId].lockedUntil) {
127 if (currentEpoch > options[tokenId].validUntil) {
130 if (_peek(address(chi)) < options[tokenId].strikePrice) {
137 chi.safeTransfer(msg.sender, options[tokenId].amount);
140 emit Burn(msg.sender, tokenId, options[tokenId].amount);
```

CVF-161. INFO

- **Category** Suboptimal
- **Source** OCHI.sol

Recommendation This could be replaced with a simple multiplication by MUTIPLIER / (2 * MAX_LOCK_PERIOD_EPOCHS)

Client Comment *It is easier to understand it this way since it replicates business logic*

```
193 uint256 timeMultiplier = Math.mulDiv(lockPeriodInEpochs, MULTIPLIER,
    ↪  2 * MAX_LOCK_PERIOD_EPOCHS);
```

CVF-162. INFO

- **Category** Suboptimal
- **Source** OCHI.sol

Recommendation While the ”mulDiv” function is very efficient in generic case, for specific cases, better approaches do exist. For example, when the denominator is a compile-time constant, its modular reciprocal could be precomputed. Also, the approach described here is more efficient for denominator that do fit into 128 bits: https://medium.com/coinmonks/math-in-solidity-part-3-percents-and-proportions-4db014e080b1#4821

Client Comment *We agree with this, but it's not very important optimization, and will decrease the readability and understanding of the code. We'll have it in mind for version 2.*

```
197 uint256 discount = Math.mulDiv(timeMultiplier, Math.min(
       ↪ poolMultiplier + chiMultiplier, MULTIPLIER), MULTIPLIER);

199 strikePrice = Math.mulDiv(chiPrice, MULTIPLIER - discount,
       ↪ MULTIPLIER);

201 uint256 chiValue = Math.mulDiv(chiAmount, chiPrice, MULTIPLIER);

214   Math.mulDiv(uscEthPairTotalSupply, TARGET_RATIO, DENOMINATOR) ||

216   Math.mulDiv(chiEthPairTotalSupply, TARGET_RATIO, DENOMINATOR)
```

CVF-163. INFO

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

Description This event is emitted even if nothing actually changed.

Client Comment *Thsi is expected behaviour*

```
27 emit UpdateMinter(account, status);
```

CVF-164. INFO

- **Category** Procedural
- **Source** Arbitrage.sol

Recommendation Consider importing this from ”@uniswap/v2-periphery” unless there are some differences..

Client Comment *Our contracts are the same as uniswap we just changed compiler version due to compiler incompatibility*

```
18 import "contracts/uniswap/libraries/UniswapV2Library.sol";
```

CVF-165. FIXED

- **Category** Procedural
- **Source** Arbitrage.sol

Description We didn’t review this file.

```
18 import "contracts/uniswap/libraries/UniswapV2Library.sol";
```

CVF-166. FIXED

- **Category** Readability
- **Source** Arbitrage.sol

Recommendation This value could be rendered as ”1e8”.

```
28 uint256 public constant BASE_PRICE = 10 ** 8;
   uint256 public constant USC_TARGET_PRICE = 10 ** 8;
```

CVF-167. FIXED

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

Recommendation The type for this variable should be ”IUniswapV2Pair”.

```
44 address public immutable uscEthPool;
```

CVF-168. FIXED

- **Category** Suboptimal
- **Source** Arbitrage.sol

Recommendation These functions should emit some events.

```
70 function setPriceTolerance(uint16 _priceTolerance) external
   ↪ onlyOwner {
```

```
80 function setMaxMintPriceDiff(uint256 _maxMintPriceDiff) external
   ↪ onlyOwner {
```

CVF-169. FIXED

- **Category** Readability
- **Source** Arbitrage.sol

Recommendation Should be ”else if”.

```
151 if (isPriceAboveTarget) {
```

CVF-170. FIXED

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

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

Client Comment *Not important here, we are casting when calling external functions.*

```
166 function _mint(uint256 amount, address token) private returns (
   ↪ uint256) {
```

CVF-171. FIXED

- **Category** Documentation
- **Source** Arbitrage.sol

Description These function names are confusing.

Recommendation Consider either making them more descriptive, or adding documentation comments explaining the arbitrage logic.

```
188 function _executeArbitrage1(uint256 reserveDiff, uint256 ethPrice)
    ↪ private returns (uint256) {
```

```
223 function _executeArbitrage2(uint256 reserveDiff, uint256 ethPrice)
    ↪ private returns (uint256) {
```

```
246 function _executeArbitrage3(uint256 reserveDiff, uint256 ethPrice)
    ↪ private returns (uint256) {
```

```
288 function _executeArbitrage4(uint256 reserveDiff, uint256 ethPrice)
    ↪ private returns (uint256) {
```

```
334 function _executeArbitrage5(uint256 reserveDiff, uint256 discount,
    ↪ uint256 ethPrice) private returns (uint256) {
```

```
352 function _executeArbitrage6(uint256 reserveDiff, uint256 discount,
    ↪ uint256 ethPrice) private returns (uint256) {
```

CVF-172. INFO

- **Category** Procedural
- **Source** Arbitrage.sol

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

Client Comment *Not needed here. This is pretty standard receiver function*

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

CVF-173. INFO

- **Category** Procedural
- **Source** ReserveHolder.sol

Recommendation Consider including from ”@uniswap/v2-perphery” unless there are some differences.

Client Comment *It is done this way because of compiler incompatibility. Our libraries are exact as Uniswap just changed compiler version*

```
9 import "./uniswap/libraries/UniswapV2Library.sol";
```

CVF-174. FIXED

- **Category** Procedural
- **Source** ReserveHolder.sol

Description We didn’t review this file.

```
9 import "./uniswap/libraries/UniswapV2Library.sol";
```

CVF-175. FIXED

- **Category** Readability
- **Source** ReserveHolder.sol

Recommendation This value could be rendered as ”1e8”.

```
24 uint256 public constant BASE_PRICE = 10 ** 8;
```

CVF-176. INFO

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

Recommendation This value should be a named constant.

Client Comment *It can be misleadign knowning that we have setter to change this value after*

```
70 swapEthTolerance = 0.1 ether;
```

CVF-177. FIXED

- **Category** Suboptimal
- **Source** ReserveHolder.sol

Recommendation These functions should emit some events.

```
76 function setArbitrager(address arbitrager, bool status) external
      ↪ onlyOwner {
```

```
83 function setClaimer(address _claimer) external onlyOwner {
```

```
91 function setEthThreshold(uint256 _ethThreshold) external onlyOwner {
```

```
102 function setSwapEthTolerance(uint256 _swapEthTolerance) external
      ↪ onlyOwner {
```

CVF-178. INFO

- **Category** Documentation
- **Source** ReserveHolder.sol

Recommendation Consider adding a comment explaining why this logic is necessary.

Client Comment *This logic is neccessary because stETH has rounding errors*

```
129 totalStEthDeposited += stETH.balanceOf(address(this)) -
      ↪ balanceBefore;
```

```
150   totalStEthDeposited -= stEthBalanceBefore - stEthBalanceAfter;
```

```
176   totalStEthDeposited -= stEthBalanceBefore - stEthBalanceAfter;
```

CVF-179. INFO

- **Category** Suboptimal
- **Source** ReserveHolder.sol

Recommendation While the ”mulDiv” function is very efficient in generic case, for specific case better approaches do exist. For example, when the denominator is a compile-time constant, its modular reciprocal could be precomputed. Also, when a denominator fits into 128 bits, the following approach could be used: https://medium.com/coinmonks/math-in-olidity-part-3-percents-and-proportions-4db014e080b1#4821

Client Comment *We agree with this, but it's not very important optimization, and will decrease the readability and understanding of the code. We'll have it in mind for version 2.*

```
141 uint256 ethValue = Math.mulDiv(WETH.balanceOf(address(this)),
       ↪ ethPrice, BASE_PRICE);
    uint256 stEthValue = Math.mulDiv(stETH.balanceOf(address(this)),
       ↪ stEthPrice, BASE_PRICE);
    uint256 ethThresholdValue = Math.mulDiv((ethValue + stEthValue),
       ↪ ethThreshold, MAX_PERCENTAGE);
```

```
146   uint256 stEthAmountToSwap = Math.mulDiv((ethThresholdValue -
         ↪ ethValue), BASE_PRICE, stEthPrice);
```

```
155   uint256 ethAmountToSwap = Math.mulDiv((ethValue -
         ↪ ethThresholdValue), BASE_PRICE, ethPrice);
```

CVF-180. INFO

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

Description The function name looks like a getter, while the function actually does change the blockchain state.

Recommendation Consider renaming.

```
200 function getWETH() external {
```

CVF-181. INFO

- **Category** Suboptimal
- **Source** ReserveHolder.sol

Recommendation Consider including the original revert reason into the error.

Client Comment *It is descriptive enough this way.*

```
252 revert EtherSendFailed(to, value);
```

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