LUNARAY
BLOCKCHAINSECURITY

SMART CONTRACT SECURITY AUDIT REPORT

For El Dorado Exchange

8 December 2022

lunaray.co

Table of Contents

1. Overview

On Nov 17, 2022, the security team of Lunaray Technology received the security audit request of the **El Dorado Exchange project**. The team completed the audit of the **El Dorado Exchange smart contract** on Dec 8, 2022. During the audit process, the security audit experts of Lunaray Technology and the EDE project interface Personnel communicate and maintain symmetry of information, conduct security audits under controllable operational risks, and avoid risks to project generation and operations during the testing process.

Through communicat and feedback with EDE project party, it is confirmed that the loopholes and risks found in the audit process have been repaired or within the acceptable range. The result of this EDE smart contract security audit: **Passed**

Audit Report Hash:

DF3329FB63D4B2E7D7081806581B8B380C89BACF1A0C745E00B69D5319A0EB16

2. Background

2.1 Project Description

Project name	El Dorado Exchange
Contract type	Spot and perpetual social trading
Code language	Solidity
Public chain	BNB Chain
Project website	https://www.ede.finance/
Contract file	BasePositionManager.sol,ElpManager.sol,OrderBook.sol, Vault.sol,VaultUtils.sol,Router.sol,VaultPriceFeedV2.sol, PositionRouter.sol,PositionManager.sol,ELP.sol, RewardTracker.sol,RewardRouter.sol
Brief introduction	El Dorado Exchange(EDE) is a decentralized spot and perpetual social trading exchange which prioritizes user security and stable investor returns. In EDE, all the interactions will happen on-chain. Trading is supported by 3 unique multi-asset pools that earn liquidity providers fees from market making, swap fees and leverage trading. Dynamic pricing is supported by Chainlink Oracles and an aggregate of prices from leading volume exchanges.

2.2 Audit Range

El Dorado Exchange provides the smart contract file and the corresponding contract file hash：

Name	Hash
BasePositionManager.sol	2D0C108B9B66CFF683D507B69BE147427D4649E168E7362CD9CC2E0AABEA31E4
ElpManager.sol	38C7DF9BBA631E32C086B2BE0CBEE20A9D3BAA76F659B1E860D8D44FD15160E7
OrderBook.sol	8E42B78AF51896D66CC7361A3F88FEFA42C2BF865337FD8C1A08541949B1427C
Vault.sol	DF8E2069490CB03AA7D422F526F39F57770B1F70B34EE254B47BB57D188D8F42
VaultUtils.sol	327DC5E5BF4580E1ED6C3E7579CE18EFC76EF44C0E47F8E539CF525C29B37C48
VaultPriceFeedV2.sol	72FEB52ED7F8B2392560E4EAAA65CEB9A3A07FC932F59EF04CC60AE37B7FA89C
Router.sol	3843545DFF2930E39EBD1762D761E90E6C0F1677C62FEA8B1CDD34B3D46ACEF7
PositionRouter.sol	48A7C16D90F90357C7A003750E2239A22B15A83DC4D0D00C2569C3EB8B658F99
PositionManager.sol	A2426718768A680E3B424847D7727AAE195CB8C382B3E639B551F81F36D8492A
ELP.sol	36590E87A291B6A5EB198CE47670AC188C3FD02E333046A5A48B907E889F63DA

RewardTracker.sol	8011C4A286C0201BF05B43160A111BFB53295C4817B9FB6C538E538E68343F97
RewardRouter.sol	5365975B20C40BE1D582F91A1BE58E2D64823CCEB931712DB230217E681216BF

3. Project contract details

3.1 Contract Overview

BasePositionManager Contract

BasePositionManager contract is to call and set part of the content, including some implementation methods by others Contract call. The caller role is mainly administrator, and the admin administrator, vault contract, router contract, weth and depositFee address, and the administrator permission can also modify some variable values, and the administrator has the permission to transfer funds. In addition, the contract implements some internal methods for other functions to call.

ElpManager Contract

The main role of the ElpManager contract is to add and remove liquidity, and contains some variables and sums that can be modified by the administrator Some of the data public methods are available. Among them, adding liquidity is divided into two types, which are divided into native chain tokens and other tokens, and removed. There are also two types of liquidity. The token that adds liquidity is mainly set by the administrator to set the whitelist, and then the user can dialogue List tokens to add liquidity.

OrderBook Contract

ElpManager contracts mainly include: creating, updating, canceling, ordering, and executing orders; Add, update, cancel, execute (add/reduce orders); Contains some administrator setting parameters. createSwapOrder has two calling roles, which are normal users. Can be transferred to the on-chain native token and the plugins role set by the router administrator (the role can be transferred to other tokens, and After the role is managed, it needs to manually call the approvePlugin method for the logic to work). orders can be updated and cancelled separately by the creator.

Valut Contract

The Valut contract is the base contract of the entire system, which is mainly called and used by other contracts. The role that calls the contract is Owner, Manager, liquidator and normal user, the Manager role is set by the Owner. The contract that the Owner role can call is initialization function, set the interface contract address, set the management mode, set the administrator liquidator, update the token address and quantity, and set the profit rate, set Token configuration, etc.; The contract that the Manager role can call is to buy and sell USDX, calculate reserves, increase positions, and reduce positions. calculating rewards, etc.; The functions that users can call include setting the router, token exchange, querying token information, etc.; Liquidation function tuning with permissions set by Owner, when Owner sets the liquidation status to private mode, only liquidators can perform liquidation operations When the liquidation status is set to non-private mode, all users can act as liquidators to perform liquidation.

ValutUtils Contract

The ValitUtils contract is mainly used to supplement the contract with the function of the Valut contract, and the main function is that the administrator sets various interest rates. Query position information, obtain funding interest rate, calculate the commission for buying and selling USDX or exchange, query clearing and other functions.

ValutPriceFeedV2 Contract

The ValutPriceFeed V2 contract mainly provides a feeding mechanism, which is called by other contracts to query the token price. The main function of the contract There is Owner to set the price update time, chain flag, AMM status, token address, token configuration, etc.; Users can query token price, get the initial price, get the on-chain price, get the recent price, get the secondary price, etc.

Router Contract

Router contract is for users to increase positions and reduce positions through this contract, and mainly call Valut contract execution make. The main functions of the contract are user authorization, transfer through the router, transfer tokens to the pool, token exchange, Directly increase and reduce positions by specifying a token, increase and reduce positions through ETH, reduce positions and convert them into user-specified tokens or ETH and so on.

PositionRouter Contract

PositionRouter contract is to perform position increase and reduce positions, and the contract inherits the BasePositionManager contract. The contracts called are BasePositionManager, Valut contracts, etc. The main functions of the contract are Admin to set the position manager, Minimum execution commission, leverage status, delay value, etc.; The main functions of the position manager are to perform batch position increase and reduce positions; ordinary User functions include creating additional positions and unincreasing and reducing positions.

PositionManager Contract

PositionManager contract is to set the order manager, partner, liquidator and status for the deployer. The main function of the contract is in the partner or in the traditional mode, perform position addition, position reduction or position reduction and exchange for user-specified tokens; The liquidator authority performs the liquidation function; orders. The administrator performs an increase, decrease or exchange order.

ELP Contract

ELP contracts is to carry out token transfers, etc., and contracts such as Valut and RewardTracker are called. The main function of the contract. There are functions such as administrators setting ELP token information, setting token miners, and making transfers; Users can make authorizations, transfers, get rewards, and more.

RewardTracker Contract

RewardTracker contract functions include administrator initialization contract, setting handler, setting some variable values; The handler role updates the reward rate, account deposit, cancellation of account deposit, and account claim of the pool; User stake, cancel stake, update account rewards, claim, can claim;

RewardRouter Contract

RewardRouter contract functions include administrator initialization contract, setting some variable values, setting whitelist tokens, setting and clearing ELPn, and transferring funds; use Users obtain staking ELPn information, staking ELPn, releasing ELPn, applying for EDE account rewards, requesting EUSD quantity, and obtaining number of rewards, withdrawal of funds to EDE pool, purchase of EUSD, sale of EUSD.

3.2 Contract details

BasePositionManager Contract

Name	Parameter	Attributes
setAdmin	address _admin	onlyOwner
setESBT	address _esbt	onlyOwner
setDepositFee	uint256 _depositFee	onlyAdmin
setIncreasePositionBufferBps	uint256 _increasePositionBufferBps	onlyAdmin
withdrawFees	address _token address _receiver	onlyAdmin
approve	address _token address _spender uint256 _amount	onlyOwner
sendValue	Address payable _receiver uint256 _amount	onlyOwner
_increasePosition	address _account address _collateralToken address _indexToken uint256 _sizeDelta bool _isLong uint256 _price	internal
_decreasePosition	address _account address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong address _receiver uint256 _price	internal
_vaultSwap	address _tokenIn address _tokenOut uint256 _minOut address _receiver	internal
_transferInETH	none	internal
_transferOutETH	uint256 _amountOut addres spayable _receiver	internal
_transferOutETHWithGasLimit	uint256 _amountOut addresspayable _receiver	internal

ElpManager Contract

Name	Parameter	Attributes
setInPrivateMode	bool _inPrivateMode	onlyOwner
setHandler	address _handler bool _isActive	onlyOwner
setESBT	address _esbt	onlyOwner
setCooldownDuration	uint256 _cooldownDuration	onlyOwner
setAumAdjustment	uint256 _aumAddition uint256 _aumDeduction	onlyOwner
addLiquidity	address _token uint256 _amount uint256 _minUsdx uint256 _minElp	external
addLiquidityETH	none	external
_addLiquidity	address _fundingAccount address _account address _token uint256 _amount uint256 _minUsdx uint256 _minElp	private
removeLiquidity	Address _tokenOut uint256 _elpAmount uint256 _minOut address _receiver	external
removeLiquidity ForAccount	address _account address _tokenOut uint256 _elpAmount uint256 _minOut address _receiver	external
_removeLiquidity	address _account address _tokenOut uint256 _elpAmount uint256 _minOut address _receiver	private
removeLiquidityETH	uint256 _elpAmount	external
getPoolInfo	none	public
getPoolTokenList	none	public
getPoolTokenInfo	address _token	public
getAums	none	public
getAumInUSD	bool maximise	public
getAum	bool maximise	public
getAumSimple	bool maximise	public
getWeightDetailed	none	public
_validateHandler	none	private

OrderBook Contract

Name	Parameter	Attributes
initialize	address _router address _vault address _weth uint256 _minExecutionFee uint256 _minPurchaseTokenAmountUsd	onlyOwner
setMinExecutionFee	uint256 _minExecutionFee	onlyOwner
setMinPurchaseToken AmountUsd	uint256 _minPurchaseTokenAmountUsd	onlyOwner
getSwapOrder	address _account uint256 _orderIndex	public
cancelSwapOrder	uint256 _orderIndex	public
updateSwapOrder	uint256 _orderIndex uint256 _minOut uint256 _triggerRatio bool _triggerAboveThreshold	external
executeSwapOrder	address _account uint256 _orderIndex addresspayable _feeReceiver	external
validatePosition OrderPrice	bool _triggerAboveThreshold uint256 _triggerPrice address _indexToken bool _maximizePrice bool _raise	public
getDecreaseOrder	address _account uint256 _orderIndex	public
getIncreaseOrder	address _account uint256 _orderIndex	public
_createIncreaseOrder	address _account address _purchaseToken uint256 _purchaseTokenAmount address _collateralToken address _indexToken uint256 _sizeDelta bool _isLong uint256 _triggerPrice bool _triggerAboveThreshold uint256 _executionFee	private
updateIncreaseOrder	uint256 _orderIndex uint256 _sizeDelta uint256 _triggerPrice bool _triggerAboveThreshold	external
cancelIncreaseOrder	uint256 _orderIndex	public

executeIncreaseOrder	address _address uint256 _orderIndex addresspayable _feeReceiver	external
createDecreaseOrder	address _indexToken uint256 _sizeDelta address _collateralToken uint256 _collateralDelta bool _isLong uint256 _triggerPrice bool _triggerAboveThreshold	external
_createDecreaseOrder	address _account address _collateralToken uint256 _collateralDelta address _indexToken uint256 _sizeDelta bool _isLong uint256 _triggerPrice bool _triggerAboveThreshold	private
executeDecreaseOrder	address _address uint256 _orderIndex address payable _feeReceiver	external
cancelDecreaseOrder	uint256 _orderIndex	public
updateDecreaseOrder	uint256 _orderIndex uint256 _collateralDelta uint256 _sizeDelta uint256 _triggerPrice bool _triggerAboveThreshold	external
_transferInETH	none	private
_transferOutETH	uint256 _amountOut address payable _receiver	private
_vaultSwap	address _tokenIn address _tokenOut uint256 _minOut address _receiver	private

Valut Contract

Name	Parameter	Attributes
setVaultUtils	address _vaultUtils	onlyOwner
setESBT	address _eSBT	onlyOwner
allWhitelistedTokensLength	none	external
setInManagerMode	bool _inManagerMode	onlyOwner
setManager	address _manager bool _isManager	onlyOwner
setInPrivateLiquidationMode	bool _inPrivateLiquidationMode	onlyOwner
setLiquidator	address _liquidator bool _isActive	onlyOwner
setIsSwapEnabled	bool _isSwapEnabled	onlyOwner
setIsLeverageEnabled	bool _isLeverageEnabled	onlyOwner
setPriceFeed	address _priceFeed	onlyOwner
setBufferAmount	address _token uint256 _amount	onlyOwner
setMaxGlobalShortSize	address _token uint256 _amount	onlyOwner
setFundingRate	uint256 _fundingInterval uint256 _fundingRateFactor uint256 _stableFundingRateFactor	onlyOwner
setTokenConfig	address _token uint256 _tokenDecimals uint256 _tokenWeight uint256 _minProfitBps uint256 _maxUSDAmount bool _isStable bool _isShortable	onlyOwner
clearTokenConfig	address _token	onlyOwner
getTokenFundingRate	address _token	external
addRouter	address _router	external
removeRouter	address _router	external
setUsdxAmount	address _token uint256 _amount	onlyOwner
upgradeVault	address _newVault address _token uint256 _amount	onlyOwner
directPoolDeposit	address _token	external
buyUSDX	address _token address _receiver	external

sellUSDX	address _token address _receiver uint256 _usdxAmount	external
claimFeeToken	address _token	external
swap	address _tokenIn address _tokenOut address _receiver	external
increasePosition	address _account address _collateralToken address _indexToken uint256 _sizeDelta bool _isLong	external
decreasePosition	address _account address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong address _receiver	external
_decreasePosition	address _account address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong address _receiver	private
claimFeeReserves	none	external
claimableFeeReserves	none	external
liquidatePosition	address _account address _collateralToken address _indexToken bool _isLong address _feeReceiver	external
validateLiquidation	address _account address _collateralToken address _indexToken bool _isLong bool _raise	public
getMaxPrice	address _token	public
getMinPrice	address _token	public
getRedemptionAmount	address _token uint256 _usdxAmount	public
getRedemptionCollateral	address _token	public

getRedemptionCollateralUsd	address _token	public
adjustForDecimals	uint256 _amount address _tokenDiv address _tokenMul	public
tokenToUsdMin	address _token uint256 _tokenAmount	public
usdToTokenMax	address _token uint256 _usdAmount	public
usdToTokenMin	address _token uint256 _usdAmount	public
usdToToken	address _token uint256 _usdAmount uint256 _price	public
getPosition	address _account address _collateralToken address _indexToken bool _isLong	public
getPositionKey	address _account address _collateralToken address _indexToken bool _isLong uint256 _keyID	public
updateCumulative FundingRate	address _collateralToken address _indexToken	public
getNextFundingRate	address _token	public
getPositionLeverage	address _account address _collateralToken address _indexToken bool _isLong	public
getNextAveragePrice	address _indexToken uint256 _size uint256 _averagePrice bool _isLong uint256_nextPrice uint256_sizeDelta uint256 _lastIncreasedTime	public
getNextGlobalShort AveragePrice	address _indexToken uint256_nextPrice uint256_sizeDelta	public
getGlobalShortDelta	address _token	public
getPositionDelta	address _account address _collateralToken address _indexToken bool _isLong	public
getDelta	address _indexToken uint256 _size uint256 _averagePrice bool _isLong uint256 _lastIncreasedTime	public
getTargetUsdxAmount	address _token	public

_reduceCollateral	address _account address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong	private
_validatePosition	uint256 _size uint256 _collateral	private
_validateRouter	address _account	private
_validateTokens	address _collateralToken address _indexToken bool _isLong	private
_collectSwapFees	address _token uint256 _amount uint256 _feeBasisPoints	private
_collectMarginFees	address _account address _collateralToken address _indexToken bool _isLong uint256 _sizeDelta uint256 _size uint256 _entryFundingRate	private
_transferIn	address _token	private
_transferOut	address _token uint256 _amount address _receiver	private
_increasePoolAmount	address _token uint256 _amount	private
_decreasePoolAmount	address _token uint256 _amount	private
_validateBufferAmount	address _token	private
_increaseUsdxAmount	address _token uint256 _amount	private
_decreaseUsdxAmount	address _token uint256 _amount	private
_increaseReservedAmount	address _token uint256 _amount	private
_decreaseReservedAmount	address _token uint256 _amount	private
_increaseGuaranteedUsd	address _token uint256 _usdAmount	private
_decreaseGuaranteedUsd	address _token uint256 _usdAmount	private
tokenUtilization	address _token	public
_increaseGlobalShortSize	address _token uint256 _amount	private
_decreaseGlobalShortSize	address _token uint256 _amount	private
_validate	bool _condition uint256 _errorCode	private

ValutUtils Contract

Name	Parameter	Attributes
setFees	uint256 _taxBasisPoints uint256 _stableTaxBasisPoints uint256 _mintBurnFeeBasisPoints uint256 _swapFeeBasisPoints uint256 _stableSwapFeeBasisPoints uint256 _marginFeeBasisPoints uint256 _liquidationFeeUsd uint256 _minProfitTime bool _hasDynamicFees	onlyOwner
setMaxLeverage	uint256 _maxLeverage	onlyOwner
setUploadingStatus	address _new_uis bool _isActive	onlyOwner
setDiscountStatus	address _new_uis bool _isActive	onlyOwner
getPosition	address _account address _collateralToken address _indexToken bool _isLong	internal
validateLiquidation	address _account address _collateralToken address _indexToken bool _isLong bool _raise	public
getBuyUsdxFeeBasisPoints	address _token uint256 _usdxAmount	public
getSellUsdxFeeBasisPoints	address _token uint256 _usdxAmount	public
getSwapFeeBasisPoints	address _tokenIn address _tokenOut uint256 _usdxAmount	public
getFeeBasisPoints	address _token uint256 _usdxDelta uint256 _feeBasisPoints uint256 _taxBasisPoints bool _increment	public

ValutPriceFeedV2 Contract

Name	Parameter	Attributes
setSafePriceTimeGap	uint256 _gap	onlyOwner
setChainlinkFlags	address _chainlinkFlags	onlyOwner
setAdjustment	address _token bool _isAdditive uint256 _adjustmentBps	onlyOwner
setUseV2Pricing	bool _useV2Pricing	onlyOwner
setIsAmmEnabled	bool _isEnabled	onlyOwner
setIsSecondaryPriceEnabled	bool _isEnabled	onlyOwner
setSecondaryPriceFeed	address _secondaryPriceFeed	onlyOwner
setTokens	address _btc address _eth address _bnb	onlyOwner
setPairs	address _bnbBusd address _ethBnb address _btcBnb	onlyOwner
setSpreadBasisPoints	address _token uint256 _spreadBasisPoints	onlyOwner
setSpreadThresholdBasisPoints	uint256 _spreadThresholdBasisPoints	onlyOwner
setFavorPrimaryPrice	bool _favorPrimaryPrice	onlyOwner
setPriceSampleSpace	uint256 _priceSampleSpace	onlyOwner
setMaxStrictPriceDeviation	uint256 _maxStrictPriceDeviation	onlyOwner
setTokenChainlink	address _token address _chainlinkContract	onlyOwner
setTokenConfig	address _token address _priceFeed uint256 _priceDecimals bool _isStrictStable	onlyOwner
getOrigPrice	address _token	public
getChainlinkPrice	address _token	public
getLatestPrimaryPrice	address _token	public
getPrimaryPrice	address _token bool _maximise	public
getSecondaryPrice	address _token uint256 _referencePrice bool _maximise	public

Router Contract

Name	Parameter	Attributes
setESBT	address _esbt	onlyOwner
setInfoCenter	address _infCenter	onlyOwner
addPlugin	address _plugin	onlyOwner
removePlugin	address _plugin	onlyOwner
approvePlugin	address _plugin	external
denyPlugin	address _plugin	external
pluginTransfer	address _token address _account address _receiver uint256 _amount	external
pluginIncreasePosition	address _account address _collateralToken address _indexToken uint256 _sizeDelta bool _isLong	external
pluginDecreasePosition	address _account address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong address _receiver	external
directPoolDeposit	address _token uint256 _amount	external
decreasePosition	address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong address _receiver uint256 _price	external
decreasePositionETH	address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong address payable _receiver uint256 _price	external
_increasePosition	address _collateralToken address _indexToken uint256 _sizeDelta bool _isLong uint256 _price	private
_decreasePosition	address _collateralToken address _indexToken	private

	uint256 _collateralDelta uint256 _sizeDelta bool _isLong address _receiver uint256 _price	
_transferETHToVault	none	private
_transferOutETH	uint256 _amountOut address payable _receiver	private
_vaultSwap	address _tokenIn address _tokenOut uint256 _minOut address _receiver	private
_sender	none	private
_validatePlugin	address _account	private
isContract	address addr	private

PositionRouter Contract

Name	Parameter	Attributes
setPositionKeeper	address _account bool _isActive	onlyAdmin
setMinExecutionFee	uint256 _minExecutionFee	onlyAdmin
setIsLeverageEnabled	bool _isLeverageEnabled	onlyAdmin
setDelayValues	uint256 _minBlockDelayKeeper uint256 _minTimeDelayPublic uint256 _maxTimeDelay	onlyAdmin
setRequestKeysStartValues	uint256 _increasePositionRequestKeysStart uint256 _decreasePositionRequestKeysStart	onlyAdmin
pendingIncreasePositions	none	public
pendingDecreasePositions	none	public
executeIncreasePositions	uint256 _endIndex addresspayable _executionFeeReceiver	onlyPositionKeeper
executeDecreasePositions	uint256 _endIndex addresspayable _executionFeeReceiver	onlyPositionKeeper
getRequestQueueLengths	none	external
executeIncreasePosition	bytes32 _key addresspayable _executionFeeReceiver	public
cancelIncreasePosition	bytes32 _key addresspayable _executionFeeReceiver	public
executeDecreasePosition	bytes32 _key addresspayable _executionFeeReceiver	public
cancelDecreasePosition	bytes32 _key addresspayable _executionFeeReceiver	public

getRequestKey	address _account uint256 _index	public
getIncreasePositionRequestPath	bytes32 _key	public
getDecreasePositionRequestPath	bytes32 _key	public
_validateExecution	uint256 _positionBlockNumber uint256 _positionBlockTime address _account	internal
_validateCancellation	uint256 _positionBlockNumber uint256 _positionBlockTime address _account	internal

PositionManager Contract

Name	Parameter	Attributes
setOrderKeeper	address _account bool _isActive	onlyAdmin
setLiquidator	address _account bool _isActive	onlyAdmin
setPartner	address _account bool _isActive	onlyAdmin
setInLegacyMode	bool _inLegacyMode	onlyAdmin
setShouldValidate IncreaseOrder	bool _shouldValidateIncreaseOrder	onlyAdmin
decreasePosition	address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong address _receiver uint256 _price	onlyPartnersOr LegacyMode
decreasePositionETH	address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong address payable _receiver uint256 _price	onlyPartnersOr LegacyMode
liquidatePosition	address _account address _collateralToken address _indexToken bool _isLong address _feeReceiver	onlyLiquidator
executeSwapOrder	address _account uint256 _orderIndex addresspayable _feeReceiver	onlyOrderKeeper
executeIncreaseOrder	address _account uint256 _orderIndex addresspayable _feeReceiver	onlyOrderKeeper
executeDecreaseOrder	address _account uint256 _orderIndex addresspayable _feeReceiver	onlyOrderKeeper
_validateIncreaseOrder	address _account uint256 _orderIndex	internal

ELP Contract

Name	Parameter	Attributes
setMinter	address _minter bool _isActive	onlyOwner
mint	address _account uint256 _amount	onlyMinter
burn	address _account uint256 _amount	onlyMinter
setInfo	string _name string _symbol	onlyOwner
withdrawToken	address _token address _account uint256 _amount	onlyOwner
setHandler	address _handler bool _isActive	onlyOwner
balanceOf	address _account	external
transfer	address _recipient uint256 _amount	external
allowance	address _owner address _spender	external
approve	address _spender uint256 _amount	external
transferFrom	address _sender address _recipient uint256 _amount	external
_mint	address _account uint256 _amount	internal
_burn	address _account uint256 _amount	internal
_transfer	address _sender address _recipient uint256 _amount	internal
_approve	address _owner address _spender uint256 _amount	private
initialize	address _vault address _eusd uint256 _eusdDecimals	onlyOwner
updateStakingAmount	address _account uint256 _amount	external
setStakingPoolAddress	address _pool	onlyOwner
setELPStakingTracker	address _elppool	onlyOwner
setManager	address _manager bool _isManager	onlyOwner
_validateManager	none	private
_validateInWhitelist	address _token	private
setTokenConfig	address _token uint256 _tokenDecimals	onlyOwner
USDbyFee	none	external
TokenFeeReserved	address _token	external
_updateRewardsLight	address _account	private

_updateRewards	address _account	private
claim	address _receiver	public
claimForAccount	address _account	public
claimable	address _account	external
_claim	address _account address _receiver	private
_pendingRewards	none	private
getFeeAmount	uint64 _stasticDays uint64 _shiftDays	external
adjustForEUSDDecimals	uint256 _amount address _tokenDiv	public
withdrawToEDEPool	none	external

RewardTracker Contract

Name	Parameter	Attributes
initialize	address _depositToken address _rewardToken uint256 _poolRewardPerInterval	onlyOwner
setHandler	address _handler bool _isActive	onlyOwner
poolStakedAmount	none	external
updatePoolRewardRate	uint256 _poolRewardPerInterval	external
setInPrivateStakingMode	bool _inPrivateStakingMode	onlyOwner
setInPrivateClaimingMode	bool _inPrivateClaimingMode	onlyOwner
balanceOf	address _account	external
stake	address _depositToken uint256 _amount	external
stakeForAccount	address _fundingAccount address _account address_depositToken uint256_amount	external
unstake	address _depositToken uint256 _amount	external
unstakeForAccount	address _account address _depositToken uint256 _amount address _receiver	external
updateRewardsForUser	address _account	external
claim	address _receiver	external
claimForAccount	address _account address _receiver	external
claimable	address _account	external
_validateHandler	none	private
_stake	address _fundingAccount address _account address _depositToken uint256 _amount	private
_unstake	address _account address _depositToken uint256 _amount address _receiver	private
_pendingRewards	none	private
_updateRewards	address _account	private

RewardRouter Contract

Name	Parameter	Attributes
initialize	address _rewardToken address _eusd address _weth address _pricefeed uint256 _base_fee_point	onlyOwner
setPriceFeed	address _pricefeed	onlyOwner
setBaseFeePoint	uint256 _base_fee_point	onlyOwner
setCooldownDuration	uint256 _setCooldownDuration	onlyOwner
setTokenConfig	address _token uint256 _token_decimal	onlyOwner
delToken	address _token	onlyOwner
setELPn	address _elp_n uint256 _elp_n_weight address _stakedELPnVault uint256 _elp_n_decimal address _stakedElpTracker	onlyOwner
clearELPn	address _token	onlyOwner
withdrawToken	address _token address _account uint256 _amount	onlyOwner
stakedELPnAmount	none	external
stakeELPn	address _token uint256 _elpAmount	external
unstakeELPn	address _tokenIn uint256 _tokenInAmount	external
claimEDEForAccount	address _account	external
claimEDE	none	external
claimEUSDForAccount	address _account	public
claimEUSD	none	public
claimableEUSDForAccount	address _account	external
claimableEUSD	none	external
claimableEUSDListForAccount	address _account	external
claimableEUSDList	none	external
claimAllForAccount	address _account	external

claimAll	none	external
_claimEUSD	address _account	private
_claimEDE	address _account	private
claimableEDEListForAccount	address _account	external
claimableEDEList	none	external
claimableEDEForAccount	address _account	external
claimableEDE	none	external
withdrawToEDEPool	none	external
_USDbyFee	none	internal
_collateralAmount	address token	internal
EUSDCirculation	none	public
feeAUM	none	public
lvt	none	public
_buyEUSDFee	uint256 _aumToEUSD uint256 _EUSDSupply	internal
_sellEUSDFee	uint256 _aumToEUSD uint256 _EUSDSupply	internal
buyEUSD	address _token uint256 _amount	external
buyEUSDNative	none	external
_buyEUSD	address _account address _token uint256 _amount	internal
claimGeneratedFee	address _token	public
sellEUSD	address _token uint256 _EUSDamount	public
sellEUSDNative	uint256 _EUSDamount	public
_sellEUSD	address _account address _token uint256 _EUSDamount	internal
getEUSDPoolInfo	none	external
getEUSDCollateralDetail	none	external

4. Audit details

4.1 Findings Summary

Severity	Found	Resolved	Acknowledged
High	0	0	0
Medium	3	2	1
Low	7	7	0
Info	6	1	5

4.2 Risk distribution

Name	Risk level	Repair status
Callable functions	Medium	Acknowledged
Reentry attack	Medium	Resolved
Specific role permissions	Medium	Resolved
Variable bounds	Low	Resolved
Generate orders multiple times	Low	Resolved
Path judgment	Low	Resolved
The token is controllable	Low	Resolved
Create a large number of failed orders	Low	Resolved
Self Transfer	Low	Resolved
Transfer order	Low	Resolved
The return value is zero	Info	Resolved
Administrator permissions	info	Acknowledged
Token cooldown	info	Acknowledged
Token authorization	info	Acknowledged
Same address judgment	info	Acknowledged
Redundant codes	Info	Acknowledged
Variables are updated	No	normal
Floating Point and Numeric Precision	No	normal
Default visibility	No	normal
tx.origin authentication	No	normal

Faulty constructor	No	normal
Unverified return value	No	normal
Insecure random numbers	No	normal
Timestamp Dependent	No	normal
Transaction order dependency	No	normal
Delegatecall	No	normal
Call	No	normal
Denial of Service	No	normal
Logical Design Flaw	No	normal
Fake recharge vulnerability	No	normal
Short address attack Vulnerability	No	normal
Uninitialized storage pointer	No	normal
Frozen account bypass	No	normal
Uninitialized	No	normal
Integer Overflow	No	normal

4.3 Risk audit details

4.3.1 Callable functions

- **Risk description**

removeLiquidityETH method in ElpManager contracts, After removing liquidity, the withdrawal method is called to retrieve the user's funds, after which the sendValue method is called, which is used the account address is controllable, after which a callback contract can be carried out, since the user has returned the ELP funds, and the method is determined by nonReentrant modification, so there is no reentrancy here, reentrancy by troubleshooting other contracts or other methods, no exploitation has been found Scenario.

```
function removeLiquidityETH(uint256 _elpAmount)
        external
        payable
        nonReentrant
        returns (uint256)
    {
        if (inPrivateMode) {
            revert("ElpManager: action not enabled");
        }
        address _account = msg.sender;
        require(_account != address(0), "BEP20: transfer from the zero
address");
        require(_elpAmount > 0, "ElpManager: invalid _elpAmount");

        require(lastAddedAt[_account].add(cooldownDuration) <=
block.timestamp, "ElpManager: cooldown duration not yet passed");
        require(
            IERC20(elp).balanceOf(_account) >= _elpAmount,
            "insufficient ELP"
        );

        address _tokenOut = weth;
        uint256 aumInUSD = getAumInUSD(false);
        uint256 elpSupply = IERC20(elp).totalSupply();
        uint256 usdxAmount = _elpAmount.mul(aumInUSD).div(elpSupply);

        IMintable(elp).burn(_account, _elpAmount);
        uint256 _amountOut = vault.sellUSDX(
            _tokenOut,
            address(this),
            usdxAmount
```

```
        );

        IWETH(weth).withdraw(_amountOut);
        payable(_account).sendValue(_amountOut);

        emit RemoveLiquidity(
            _account,
            _tokenOut,
            _elpAmount,
            aumInUSD,
            elpSupply,
            usdxAmount,
            _amountOut
        );
        return _amountOut;
    }
```

- **Safety advice**

Officials need to confirm whether the rest of the contract is or not.

- **Repair Status**

EL DORADO EXCHANGE has confirmed.

4.3.2 Reentry attack

- **Risk description**

removeLiquidityETH method in ElpManager contracts, When canceling, the _transferOut ETH method is called to transfer the transfer, but the caller's place is called after the tra nsfer address the sendValue method, _transferOutETH method to execute_receiver.sen dValue(_amountOut); Over her the _receiver can execute other logic or callbacks for the address passed in by the user, _receiver If it is a contract address, there is a risk of reent rancy, and no specific utilization points have been found.

```
function cancelSwapOrder(uint256 _orderIndex) public nonReentrant {
        SwapOrder memory order = swapOrders[msg.sender][_orderIndex];
        require(order.account != address(0), "OrderBook: non-existent o
rder");


         delete swapOrders[msg.sender][_orderIndex];
         if (order.path[0] == weth) {
            _transferOutETH(
                order.executionFee.add(order.amountIn),
                payable(msg.sender)
            ); //BLKMDF
        } else {
            IERC20(order.path[0]).safeTransfer(msg.sender, order.amount
In);
            _transferOutETH(order.executionFee, payable(msg.sender)); /
/BLKMDF
        }


         emit CancelSwapOrder(
            msg.sender,
            _orderIndex,
            order.path,
            order.amountIn,
            order.minOut,
            order.triggerRatio,
            order.triggerAboveThreshold,
            order.shouldUnwrap,
            order.executionFee
        );
    }


function _transferOutETH(uint256 _amountOut, address payable _receiver)
 private
```

```
    {
        IWETH(weth).withdraw(_amountOut);
        _receiver.sendValue(_amountOut);
    }
```

- **Safety advice**

It is recommended to modify the logic of the method to avoid callbacks.

- **Repair Status**

EL DORADO EXCHANGE has fixed.

4.3.3 Specific role permissions

- **Risk description**

In the RewardTracker contract unstakeForAccount method, The parameters passed in the method are controlled by the handler caller if the method is only combined by RewardRouter.There is no risk here, but when there are other handler caller address calls, the user hostage can be arbitrarily released through this method stake and transfer assets.

```
function unstakeForAccount(
        address _account,
        address _depositToken,
        uint256 _amount,
        address _receiver
    ) external override nonReentrant {
        _validateHandler();
        require(_depositToken == depositToken, "Invalid deposit
token");
        _unstake(_account, _depositToken, _amount, _receiver);
    }
function _unstake(
        address _account,
        address _depositToken,
        uint256 _amount,
        address _receiver
    ) private {
        require(_amount > 0, "RewardTracker: invalid _amount");
        require(_depositToken == depositToken, "Invalid deposit
token");
        require(
            stakedAmounts[_account] >= _amount,
            "RewardTracker: _amount exceeds stakedAmount"
        );
        _updateRewards(_account);

        uint256 stakedAmount = stakedAmounts[_account];
        stakedAmounts[_account] = stakedAmount.sub(_amount);
        IELP(depositToken).updateStakingAmount(
            _account,
            stakedAmounts[_account]
        );

        totalDepositSupply = totalDepositSupply.sub(_amount);

        IERC20(_depositToken).safeTransfer(_receiver, _amount);
```

```
}
```

- **Safety advice**

It is recommended that officials pay attention to function call permissions.

- **Repair Status**

EL DORADO EXCHANGE has fixed.

4.3.4 Variable bounds

- **Risk description**

In the BasePositionManager contract _collectFees method, There are variable calculations relative to 10000, and if the depositFee exceeds 10000, it will cause a calculation error.

```
function _collectFees(
        address _account,
        address[] memory _path,
        uint256 _amountIn,
        address _indexToken,
        bool _isLong,
        uint256 _sizeDelta
    ) internal returns (uint256) {
        bool shouldDeductFee = _shouldDeductFee(
            _account,
            _path,
            _amountIn,
            _indexToken,
            _isLong,
            _sizeDelta
        );

        if (shouldDeductFee) {
            uint256 afterFeeAmount = _amountIn
                .mul(BASIS_POINTS_DIVISOR.sub(depositFee))
                .div(BASIS_POINTS_DIVISOR);
            uint256 feeAmount = _amountIn.sub(afterFeeAmount);
            address feeToken = _path[_path.length - 1];
            feeReserves[feeToken] =
feeReserves[feeToken].add(feeAmount);
            return afterFeeAmount;
        }
         return _amountIn;
    }
```

- **Safety advice**

It is recommended to add a setting judgment, and the requirement to set the DepositFee is < 10000

- **Repair Status**

EL DORADO EXCHANGE has fixed.

4.3.5 Generate orders multiple times

- **Risk description**

In the OrderBook contract createSwapOrder method, When the method is called, the funds will be transferred to the incoming value, but there is no judgment that the transfer-in value is 0, and the _transferInETH method will judge whether there is a transfer-in value, but when the value is 0, no error is reported for rollback.

```
function createSwapOrder(
        address[] memory _path,
        uint256 _amountIn,
        uint256 _minOut,
        uint256 _triggerRatio, // tokenB / tokenA
        bool _triggerAboveThreshold,
        uint256 _executionFee,
        bool _shouldWrap,
        bool _shouldUnwrap
    ) external payable nonReentrant {
        require(
            _path.length == 2 || _path.length == 3,
            "OrderBook: invalid _path.length"
        );
        require(
            _path[0] != _path[_path.length - 1],
            "OrderBook: invalid _path"
        );
        require(_amountIn > 0, "OrderBook: invalid _amountIn");
        require(
            _executionFee >= minExecutionFee,
            "OrderBook: insufficient execution fee"
        );

        // always need this call because of mandatory executionFee user
has to transfer in ETH
        _transferInETH();

        if (_shouldWrap) {
            require(_path[0] == weth, "OrderBook: only weth could be
wrapped");
            require(
                msg.value == _executionFee.add(_amountIn),
                "OrderBook: incorrect value transferred"
            );
        } else {
            require(
```

```
                msg.value == _executionFee,
                "OrderBook: incorrect execution fee transferred"
            );
            IRouter(router).pluginTransfer(
                _path[0],
                msg.sender,
                address(this),
                _amountIn
            );
        }

        _createSwapOrder(
            msg.sender,
            _path,
            _amountIn,
            _minOut,
            _triggerRatio,
            _triggerAboveThreshold,
            _shouldUnwrap,
            _executionFee
        );
    }
function _transferInETH() private {
        if (msg.value != 0) {
            IWETH(weth).deposit{value: msg.value}();
        }
    }
```

- **Safety advice**

It is recommended to increase the judgment of the amount of funds transferred.

- **Repair Status**

EL DORADO EXCHANGE has fixed.

4.3.6 Path judgment

- **Risk description**

In the OrderBook contract createIncreaseOrder method, The input path is calculated to get _purchaseToken, but when _path.length equals 0, an exception error condition may occur.

```
function createIncreaseOrder(
        address[] memory _path,
        uint256 _amountIn,
        address _indexToken,
        uint256 _minOut,
        uint256 _sizeDelta,
        address _collateralToken,
        bool _isLong,
        uint256 _triggerPrice,
        bool _triggerAboveThreshold,
        uint256 _executionFee,
        bool _shouldWrap
    ) external payable nonReentrant {
        // always need this call because of mandatory executionFee user
has to transfer in ETH
        _transferInETH();

        require(
            _executionFee >= minExecutionFee,
            "OrderBook: insufficient execution fee"
        );
        if (_shouldWrap) {
            require(_path[0] == weth, "OrderBook: only weth could be
wrapped");
            require(
                msg.value == _executionFee.add(_amountIn),
                "OrderBook: incorrect value transferred"
            );
        } else {
            require(
                msg.value == _executionFee,
                "OrderBook: incorrect execution fee transferred"
            );
            IRouter(router).pluginTransfer(
                _path[0],
                msg.sender,
                address(this),
                _amountIn
            );
```

```
        }
         address _purchaseToken = _path[_path.length - 1];
        uint256 _purchaseTokenAmount;
        if (_path.length > 1) {
            require(_path[0] != _purchaseToken, "OrderBook: invalid
_path");
            IERC20(_path[0]).safeTransfer(vault, _amountIn);
            _purchaseTokenAmount = _swap(_path, _minOut,
address(this));
        } else {
            _purchaseTokenAmount = _amountIn;
        }

        {
            uint256 _purchaseTokenAmountUsd =
IVault(vault).tokenToUsdMin(
                _purchaseToken,
                _purchaseTokenAmount
            );
            require(
                _purchaseTokenAmountUsd >= minPurchaseTokenAmountUsd,
                "OrderBook: insufficient collateral"
            );
        }
         _createIncreaseOrder(
            msg.sender,
            _purchaseToken,
            _purchaseTokenAmount,
            _collateralToken,
            _indexToken,
            _sizeDelta,
            _isLong,
            _triggerPrice,
            _triggerAboveThreshold,
            _executionFee
        );
    }
```

- **Safety advice**

It is recommended to add a _path path length to determine that it is 0.

- **Repair Status**

EL DORADO EXCHANGE has fixed.

4.3.7 The token is controllable

- **Risk description**

In the OrderBook contract createSwapOrder method,When the administrator sets the caller here, the transfer here _path [0] is controllable, and the caller can pass in any token.The transfer, and the contract can be called back through the token transfer method, and there is also this problem when canceling and executing the order, and no specific utilization points have been found.

```
function createSwapOrder(
        address[] memory _path,
        uint256 _amountIn,
        uint256 _minOut,
        uint256 _triggerRatio, // tokenB / tokenA
        bool _triggerAboveThreshold,
        uint256 _executionFee,
        bool _shouldWrap,
        bool _shouldUnwrap
    ) external payable nonReentrant {
        require(
            _path.length == 2 || _path.length == 3,
            "OrderBook: invalid _path.length"
        );
        require(
            _path[0] != _path[_path.length - 1],
            "OrderBook: invalid _path"
        );
        require(_amountIn > 0, "OrderBook: invalid _amountIn");
        require(
            _executionFee >= minExecutionFee,
            "OrderBook: insufficient execution fee"
        );

        // always need this call because of mandatory executionFee user
has to transfer in ETH
        _transferInETH();

        if (_shouldWrap) {
            require(_path[0] == weth, "OrderBook: only weth could be
wrapped");
            require(
                msg.value == _executionFee.add(_amountIn),
                "OrderBook: incorrect value transferred"
            );
        } else {
```

```
            require(
                msg.value == _executionFee,
                "OrderBook: incorrect execution fee transferred"
            );
            IRouter(router).pluginTransfer(
                _path[0],
                msg.sender,
                address(this),
                _amountIn
            );
        }

        _createSwapOrder(
            msg.sender,
            _path,
            _amountIn,
            _minOut,
            _triggerRatio,
            _triggerAboveThreshold,
            _shouldUnwrap,
            _executionFee
        );
    }
```

- **Safety advice**

It is recommended to add token address judgment.

- **Repair Status**

EL DORADO EXCHANGE has fixed.

4.3.8 Create a large number of failed orders

- **Risk description**

In the PositionRouter contract createIncreasePositionETH method, In the function, when the caller's balance is equal to the execution fee, the amount of funds transferred when creating an additional order is 0, which can be successful when creating an order, but it will fail when the position manager performs the increase in position, and malicious users may create a large number of functions with failed execution.

```
function createIncreasePositionETH(
        address[] memory _path,
        address _indexToken,
        uint256 _minOut,
        uint256 _sizeDelta,
        bool _isLong,
        uint256 _acceptablePrice,
        uint256 _executionFee,
        bytes32 /*_referralCode*/
    ) external payable nonReentrant {
        require(_executionFee >= minExecutionFee, "PositionRouter:
invalid executionFee");
        require(msg.value >= _executionFee, "PositionRouter: invalid
msg.value");
        require(_path.length == 1 || _path.length == 2,
"PositionRouter: invalid _path length");
        require(_path[0] == weth, "PositionRouter: invalid _path");

        _transferInETH();

        uint256 amountIn = msg.value.sub(_executionFee);

        _createIncreasePosition(
            msg.sender,
            _path,
            _indexToken,
            amountIn,
            _minOut,
            _sizeDelta,
            _isLong,
            _acceptablePrice,
            _executionFee,
            true
        );
```

```
}
```

- **Safety advice**

It is recommended that the official add the judgment of the transferred funds and conduct the fund review in advance.

- **Repair Status**

EL DORADO EXCHANGE has fixed.

4.3.9 Self Transfer

- **Risk description**

In the ELP contract transfer method, When a user sets the token receiver as his, an invalid transfer will occur, resulting in unnecessary waste of gas.

```
function transfer(address _recipient, uint256 _amount)
        external
        override
        returns (bool)
    {
        _updateRewards(msg.sender);
        _updateRewards(_recipient);
        _transfer(msg.sender, _recipient, _amount);
        return true;
    }
```

- **Safety advice**

It is recommended to increase the self-transfer judgment.

- **Repair Status**

EL DORADO EXCHANGE has fixed.

4.3.10 Transfer order

- **Risk description**

In the RewardRouter contract buyEUSD method, When purchasing EUSD, transfer EUSD to the user first, and then collect the user's purchase token, which may lead to the risk of reentrancy.

```
function buyEUSD(address _token, uint256 _amount)
        external
        nonReentrant
        returns (uint256)
    {
        address _account = msg.sender;
        require(whitelistedToken[_token], "Invalid Token");
        require(_amount > 0, "invalid amount");
        uint256 buyAmount = _buyEUSD(_account, _token, _amount);
        IERC20(_token).transferFrom(_account, address(this), _amount);
        return buyAmount;
    }
```

- **Safety advice**

It is recommended to collect the user's purchase token first, and then transfer EUSD to the user.

- **Repair Status**

EL DORADO EXCHANGE has fixed.

4.3.11 The return value is zero

- **Risk description**

In the Router contract decreasePositionAndSwap method, The exchange may fail in the function, and the position reduction function calls the Valut function, and the collateral transfer is performed directly in the Valut function, and the return value is 0, which makes the subsequent execution of the exchange operation fail.

```
function decreasePositionAndSwapETH(
        address[] memory _path,
        address _indexToken,
        uint256 _collateralDelta,
        uint256 _sizeDelta,
        bool _isLong,
        address payable _receiver,
        uint256 _price,
        uint256 _minOut
    ) external {
        require(_path[_path.length - 1] == weth, "Router: invalid
_path");
        uint256 amount = _decreasePosition(
            _path[0],
            _indexToken,
            _collateralDelta,
            _sizeDelta,
            _isLong,
            address(this),
            _price
        );
        IERC20(_path[0]).safeTransfer(vault, amount);
        uint256 amountOut = _swap(_path, _minOut, address(this));
        _transferOutETH(amountOut, _receiver);
    }
function _decreasePosition(
        address _collateralToken,
        address _indexToken,
        uint256 _collateralDelta,
        uint256 _sizeDelta,
        bool _isLong,
        address _receiver,
        uint256 _price
    ) private returns (uint256) {
        if (_isLong) {
            require(
                IVault(vault).getMinPrice(_indexToken) >= _price,
                "Router: mark price lower than limit"
```

```
            );
        } else {
            require(
                IVault(vault).getMaxPrice(_indexToken) <= _price,
                "Router: mark price higher than limit"
            );
        }

        return
            IVault(vault).decreasePosition(
                _sender(),
                _collateralToken,
                _indexToken,
                _collateralDelta,
                _sizeDelta,
                _isLong,
                _receiver
            );
    }
```

- **Safety advice**

It is recommended to officially confirm the exchange situation when the return value is 0.

- **Repair Status**

EL DORADO EXCHANGE has fixed.

4.3.12 Administrator permissions

- **Risk description**

In the BasePositionManager contract, There are two types of administrator rights in the contract, which can set some parameters and transfer funds.

```
function sendValue(address payable _receiver, uint256 _amount)
external onlyOwner {
        _receiver.sendValue(_amount);
    }
```

In the ELP contract, The owner or handler can transfer any amount of funds from any address to a specified address

```
function withdrawToken(address _token,address _account,uint256 _amount
) external onlyOwner {
        _updateRewards(_account);
        IERC20(_token).safeTransfer(_account, _amount); }
 function transferFrom( address _sender, address _recipient, uint256
_amount ) external override returns (bool) {
        _updateRewards(_sender);
        _updateRewards(_recipient);
        if (isHandler[msg.sender]) {
            _transfer(_sender, _recipient, _amount);
            return true;
        }
        uint256 nextAllowance =
allowances[_sender][msg.sender].sub(_amount,"ELP: transfer amount
exceeds allowance");
        _approve(_sender, msg.sender, nextAllowance);
        _transfer(_sender, _recipient, _amount);
        return true;}
```

- **Safety advice**

It is recommended that administrator privileges be managed using multi-signature.

- **Repair Status**

EL DORADO EXCHANGE indicates that Ownership permissions for all contracts have been transferred to the Timelock.sol contract and that Timelock contract Ownership has been transferred to the multisignature wallet, with WithdrawToken, TransferOwnership, Mint high-risk operations in all contracts being added by Timelock for 24 hours.

4.3.13 Token cooldown

- **Risk description**

In the ElpManager contract, If the user has added liquidity through addLiquidity and addLiquidityETH, due to the cooldown period of the global variable of the liquidity fund, when the user adds a new liquidity, the previous proof-of-liquidity token is also cooled.

```
function _removeLiquidity(
        address _account,
        address _tokenOut,
        uint256 _elpAmount,
        uint256 _minOut,
        address _receiver
    ) private returns (uint256) {
        require(
            _account != address(0),
            "BEP20: transfer from the zero address"
        );
        require(_elpAmount > 0, "ElpManager: invalid _elpAmount");
        require(
            lastAddedAt[_account].add(cooldownDuration) <=
block.timestamp,
            "ElpManager: cooldown duration not yet passed"
        );
        require(
            IERC20(elp).balanceOf(_account) >= _elpAmount,
            "insufficient ELP"
        );
        // calculate aum before sellUSDX
        uint256 aumInUSD = getAumInUSD(false);
        uint256 elpSupply = IERC20(elp).totalSupply();
        uint256 usdxAmount = _elpAmount.mul(aumInUSD).div(elpSupply);
        IMintable(elp).burn(_account, _elpAmount);
        uint256 amountOut = vault.sellUSDX(_tokenOut, _receiver,
usdxAmount);
        require(amountOut >= _minOut, "ElpManager: insufficient
output");

        emit RemoveLiquidity(
            _account,
            _tokenOut,
            _elpAmount,
            aumInUSD,
            elpSupply,
            usdxAmount,
```

```
            amountOut
        );

        return amountOut;
    }
```

- **Safety advice**

It is recommended to officially confirm the cooldown of the token when used at different times.

- **Repair Status**

EL DORADO EXCHANGE has confirmed.

4.3.14 Token authorization

- **Risk description**

In the OrderBook contract, createSwapOrder When the user calls the transfer, the _validatePlugin method is called, and the approvedPlugins judgment condition must be called by the user, otherwise the call fails.

```
if (_shouldWrap) {
            require(_path[0] == weth, "OrderBook: only weth could be
wrapped");
            require(
                msg.value == _executionFee.add(_amountIn),
                "OrderBook: incorrect value transferred"
            );
        } else {
            require(
                msg.value == _executionFee,
                "OrderBook: incorrect execution fee transferred"
            );
            IRouter(router).pluginTransfer(
                _path[0],
                msg.sender,
                address(this),
                _amountIn
            );
        }
function pluginTransfer(
        address _token,
        address _account,
        address _receiver,
        uint256 _amount
    ) external override {
        _validatePlugin(_account);
        IERC20(_token).safeTransferFrom(_account, _receiver, _amount);
    }
```

- **Safety advice**

It is recommended that you officially confirm the status of the function call and whether user authorization is required.

- **Repair Status**

EL DORADO EXCHANGE has confirmed.

4.3.15 Same address judgment

- **Risk description**

In the Valut contract, When executing USDX trading, there is no judgment on the same address, and there may be a situation where USDX is used to execute trading to obtain USDX.

```
function buyUSDX(address _token, address _receiver)
        external
        override
        nonReentrant
        returns (uint256)
    {
        _validate(isManager[msg.sender], 54);
        _validate(whitelistedTokens[_token], 16);
         uint256 tokenAmount = _transferIn(_token);
        _validate(tokenAmount > 0, 17);
         updateCumulativeFundingRate(_token, _token);
        uint256 price = getMinPrice(_token);
        uint256 usdxAmount =
tokenAmount.mul(price).div(vaultUtils.PRICE_PRECISION());
        usdxAmount = adjustForDecimals(usdxAmount, _token, usdx);
        _validate(usdxAmount > 0, 18);
        uint256 feeBasisPoints =
vaultUtils.getBuyUsdxFeeBasisPoints(_token,usdxAmount);
         uint256 amountAfterFees = _collectSwapFees(_token,
tokenAmount, feeBasisPoints);
        uint256 mintAmount =
amountAfterFees.mul(price).div(vaultUtils.PRICE_PRECISION());
        mintAmount = adjustForDecimals(mintAmount, _token, usdx);
        _increaseUsdxAmount(_token, mintAmount);
        _increasePoolAmount(_token, amountAfterFees);
        usdxSupply = usdxSupply.add(mintAmount);
        _increaseUsdxAmount(_receiver, mintAmount);

        return mintAmount;
    }
```

- **Safety advice**

It is recommended to increase the judgment on transferring token addresses.

- **Repair Status**

EL DORADO EXCHANGE has confirmed.

4.3.16 Redundant codes

- **Risk description**

In the Valut contract IncreasePosition method, When the function called when calculating the cumulative financing ratio in the function passes parameters, three parameters are passed, but only one is used.

```
function getEntryFundingRate(
        address _collateralToken,
        address, /* _indexToken */
        bool /* _isLong */
    ) public view override returns (uint256) {
        return vault.cumulativeFundingRates(_collateralToken);
    }
```

In the Valut contract updateCulmutiveFundingRate method, the shouldUpdate value is always true, and the if judgment condition may never be executed.

```
function updateCumulativeFundingRate(
        address _collateralToken,
        address _indexToken
    ) public {
        bool shouldUpdate =
vaultUtils.updateCumulativeFundingRate(_collateralToken, _indexToken);
        if (!shouldUpdate) {
            return;
        }

        if (lastFundingTimes[_collateralToken] == 0) {
            lastFundingTimes[_collateralToken] =
block.timestamp.div(fundingInterval).mul(fundingInterval);
            return;
        }

        if (lastFundingTimes[_collateralToken].add(fundingInterval) >
block.timestamp)
        {
            return;
        }

        uint256 fundingRate = getNextFundingRate(_collateralToken);
        cumulativeFundingRates[_collateralToken] =
cumulativeFundingRates[_collateralToken].add(fundingRate);
        lastFundingTimes[_collateralToken] =
block.timestamp.div(fundingInterval).mul(fundingInterval);
```

```
        emit UpdateFundingRate(
            _collateralToken,
            cumulativeFundingRates[_collateralToken]
        );
    }
```

In the ValutPriceFeedV2 contract, Only the initial price and the on-chain price are compared in the function, but the logic in the document will take the Secondary price for comparison, and Secondary belongs to the redundant function.

```
function getPrice(
        address _token,
        bool _maximise,
        bool,
        bool
    ) public view override returns (uint256) {
        (uint256 pricePr, bool statePr) = getPrimaryPrice(_token,
_maximise);
        (uint256 priceCl, bool stateCl) = getChainlinkPrice(_token);

        uint256 price = 0;

        require(stateCl && statePr, "Price Failure");

        uint256 price_minBound = priceCl.mul(PRICE_VARIANCE_PRECISION -
priceVariance).div(PRICE_VARIANCE_PRECISION);
        uint256 price_maxBound = priceCl.mul(PRICE_VARIANCE_PRECISION +
priceVariance).div(PRICE_VARIANCE_PRECISION);

        if ((pricePr < price_maxBound) && (pricePr > price_minBound)) {
            price = pricePr;
        } else {
            price = priceCl;
        }
        require(price > 0, "invalid price");

        uint256 adjustmentBps = adjustmentBasisPoints[_token];
        if (adjustmentBps > 0) {
            bool isAdditive = isAdjustmentAdditive[_token];
            if (isAdditive) {
                price =
price.mul(BASIS_POINTS_DIVISOR.add(adjustmentBps)).div(BASIS_POINTS_DIV
ISOR);
            } else {
                price =
price.mul(BASIS_POINTS_DIVISOR.sub(adjustmentBps)).div(BASIS_POINTS_DIV
ISOR);
```

```
            }
        }

        return price;
    }
```

In the Router contract, The isContract function is a Private function, but no other functions call this function, and you need to officially confirm whether this function is redundant.

```
function isContract(address addr) private view returns (bool) {
        uint size;
        assembly {
            size := extcodesize(addr)
        }
        return size > 0;
    }
```

In the PositionManager contract, The _validateIncreaseOrder function is an internal call function, which is not used in this contract and is not called through an interface.

```
function _validateIncreaseOrder(address _account, uint256 _orderIndex)
internal view returns (uint256){


        (
            address _purchaseToken,
            uint256 _purchaseTokenAmount,
            address _collateralToken,
            address _indexToken,
            uint256 _sizeDelta,
            bool _isLong,
            ,
            ,
        ) = IOrderBook(orderBook).getIncreaseOrder(_account, _orderInde
x);

        if (!shouldValidateIncreaseOrder) {
            return _sizeDelta;
        }

        if (!_isLong) {
            return _sizeDelta;
        }
        require(_sizeDelta > 0, "PositionManager: long deposit");
        IVault _vault = IVault(vault);
        (uint256 size, uint256 collateral, , , , , , ) = _vault.getPosi
```

```
tion(_account, _collateralToken, _indexToken, _isLong);
        // if there is no existing position, do not charge a fee
        if (size == 0) {
            return _sizeDelta;
        }
        // uint256 nextSize = size.add(_sizeDelta);
        uint256 nextSize = size.add(_sizeDelta);
        //todo: avoid overflow, using safemath
        uint256 collateralDelta = _vault.tokenToUsdMin(_purchaseToken,
_purchaseTokenAmount);

        uint256 nextCollateral = collateral.add(collateralDelta);

        uint256 prevLeverage = size.mul(BASIS_POINTS_DIVISOR).div(colla
teral);

        // allow for a maximum of a increasePositionBufferBps decrease
since there might be some swap fees taken from the collateral
        uint256 nextLeverageWithBuffer = nextSize.mul(BASIS_POINTS_DIVI
SOR + increasePositionBufferBps).div(nextCollateral);

        require(nextLeverageWithBuffer >= prevLeverage, "PositionManage
r: long leverage decrease");

        return _sizeDelta;
    }
```

- **Safety advice**

It is recommended that the redundant code be officially removed.

- **Repair Status**

EL DORADO EXCHANGE Explain that some of the code functions are not yet implemented and some of the redundant code has been removed.

4.3.17 Variables are updated

- **Risk description**

When there is a contract logic to obtain rewards or transfer funds, the coder mistakenly updates the value of the variable that sends the funds, so that the user can use the value of the variable that is not updated to obtain funds, thus affecting the normal operation of the project.

- **Audit Results : Passed**

4.3.18 Floating Point and Numeric Precision

- **Risk Description**

In Solidity, the floating-point type is not supported, and the fixed-length floating-point type is not fully supported. The result of the division operation will be rounded off, and if there is a decimal number, the part after the decimal point will be discarded and only the integer part will be taken, for example, dividing 5 pass 2 directly will result in 2. If the result of the operation is less than 1 in the token operation, for example, 4.9 tokens will be approximately equal to 4, bringing a certain degree of The tokens are not only the tokens of the same size, but also the tokens of the same size. Due to the economic properties of tokens, the loss of precision is equivalent to the loss of assets, so this is a cumulative problem in tokens that are frequently traded.

- **Audit Results : Passed**

4.3.19 Default Visibility

- **Risk description**

In Solidity, the visibility of contract functions is public pass default. therefore, functions that do not specify any visibility can be called externally pass the user. This can lead to serious vulnerabilities when developers incorrectly ignore visibility specifiers for functions that should be private, or visibility specifiers that can only be called from within the contract itself. One of the first hacks on Parity's multi-signature wallet was the failure to set the visibility of a function, which defaults to public, leading to the theft of a large amount of money.

- **Audit Results : Passed**

4.3.20 tx.origin authentication

- **Risk Description**

tx.origin is a global variable in Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract can make the contract vulnerable to phishing-like attacks.

- **Audit Results : Passed**

4.3.21 Faulty constructor

- **Risk description**

Prior to version 0.4.22 in solidity smart contracts, all contracts and constructors had the same name. When writing a contract, if the constructor name and the contract name are not the same, the contract will add a default constructor and the constructor you set up will be treated as a normal function, resulting in your original contract settings not being executed as expected, which can lead to terrible consequences, especially if the constructor is performing a privileged operation.

- **Audit Results : Passed**

4.3.22 Unverified return value

- **Risk description**

Three methods exist in Solidity for sending tokens to an address: transfer(), send(), call.value(). The difference between them is that the transfer function throws an exception throw when sending fails, rolls back the transaction state, and costs 2300gas; the send function returns false when sending fails and costs 2300gas; the call.value method returns false when sending fails and costs all gas to call, which will lead to the risk of reentrant attacks. If the send or call.value method is used in the contract code to send tokens without checking the return value of the method, if an error occurs, the contract will continue to execute the code later, which will lead to the thought result.

- **Audit Results : Passed**

4.3.23 Insecure random numbers

- **Risk Description**

All transactions on the blockchain are deterministic state transition operations with no uncertainty, which ultimately means that there is no source of entropy or randomness within the blockchain ecosystem. Therefore, there is no random number function like rand() in Solidity. Many developers use future block variables such as block hashes, timestamps, block highs and lows or Gas caps to generate random numbers. These quantities are controlled pass the miners who mine them and are therefore not truly random, so using past or present block variables to generate random numbers could lead to a destructive vulnerability.

- **Audit Results : Passed**

4.3.24 Timestamp Dependency

- **Risk description**

In blockchains, data block timestamps (block.timestamp) are used in a variety of applications, such as functions for random numbers, locking funds for a period of time, and conditional statements for various time-related state changes. Miners have the ability to adjust the timestamp as needed, for example block.timestamp or the alias now can be manipulated pass the miner. This can lead to serious vulnerabilities if the wrong block timestamp is used in a smart contract. This may not be necessary if the contract is not particularly concerned with miner manipulation of block timestamps, but care should be taken when developing the contract.

- **Audit Results : Passed**

4.3.25 Transaction order dependency

- **Risk description**

In a blockchain, the miner chooses which transactions from that pool will be included in the block, which is usually determined pass the gasPrice transaction, and the miner will choose the transaction with the highest transaction fee to pack into the block. Since the information about the transactions in the block is publicly available, an attacker can watch the transaction pool for transactions that may contain problematic solutions, modify or revoke the attacker's privileges or change the state of the contract to the attacker's detriment. The attacker can then take data from this transaction and create a higher-level transaction gasPrice and include its transactions in a block before the original, which will preempt the original transaction solution.

- **Audit Results : Passed**

4.3.26 Delegatecall

- **Risk Description**

In Solidity, the delegatecall function is the standard message call method, but the code in the target address runs in the context of the calling contract, i.e., keeping msg.sender and msg.value unchanged. This feature supports implementation libraries, where developers can create reusable code for future contracts. The code in the library itself can be secure and bug-free, but when run in another application's environment, new vulnerabilities may arise, so using the delegatecall function may lead to unexpected code execution.

- **Audit Results : Passed**

4.3.27 Call

- **Risk Description**

The call function is similar to the delegatecall function in that it is an underlying function provided pass Solidity, a smart contract writing language, to interact with external contracts or libraries, but when the call function method is used to handle an external Standard Message Call to a contract, the code runs in the environment of the external contract/function The call function is used to interact with an external contract or library. The use of such functions requires a determination of the security of the call parameters, and caution is recommended. An attacker could easily borrow the identity of the current contract to perform other malicious operations, leading to serious vulnerabilities.

- **Audit Results : Passed**

4.3.28 Denial of Service

- **Risk Description**

Denial of service attacks have a broad category of causes and are designed to keep the user from making the contract work properly for a period of time or permanently in certain situations, including malicious behavior while acting as the recipient of a transaction, artificially increasing the gas required to compute a function causing gas exhaustion (such as controlling the size of variables in a for loop), misuse of access control to access the private component of the contract, in which the Owners with privileges are modified, progress state based on external calls, use of obfuscation and oversight, etc. can lead to denial of service attacks.

- **Audit Results : Passed**

4.3.29 Logic Design Flaw

- **Risk Description**

In smart contracts, developers design special features for their contracts intended to stabilize the market value of tokens or the life of the project and increase the highlight of the project, however, the more complex the system, the more likely it is to have the possibility of errors. It is in these logic and functions that a minor mistake can lead to serious depasstions from the whole logic and expectations, leaving fatal hidden dangers, such as errors in logic judgment, functional implementation and design and so on.

- **Audit Results : Passed**

4.3.30 Fake recharge vulnerability

- **Risk Description**

The success or failure (true or false) status of a token transaction depends on whether an exception is thrown during the execution of the transaction (e.g., using mechanisms such as require/assert/revert/throw). When a user calls the transfer function of a token contract to transfer funds, if the transfer function runs normally without throwing an exception, the transaction will be successful or not, and the status of the transaction will be true. When balances[msg.sender] < _value goes to the else logic and returns false, no exception is thrown, but the transaction acknowledgement is successful, then we believe that a mild if/else judgment is an undisciplined way of coding in sensitive function scenarios like transfer, which will lead to Fake top-up vulnerability in centralized exchanges, centralized wallets, and token contracts.

- **Audit Results : Passed**

4.3.31 Short Address Attack Vulnerability

- **Risk Description**

In Solidity smart contracts, when passing parameters to a smart contract, the parameters are encoded according to the ABI specification. the EVM runs the attacker to send encoded parameters that are shorter than the expected parameter length. For example, when transferring money on an exchange or wallet, you need to send the transfer address address and the transfer amount value. The attacker could send a 19-passte address instead of the standard 20-passte address, in which case the EVM would fill in the 0 at the end of the encoded parameter to make up the expected length, which would result in an overflow of the final transfer amount parameter value, thus changing the original transfer amount.

- **Audit Results : Passed**

4.3.32 Uninitialized storage pointer

- **Risk description**

EVM uses both storage and memory to store variables. Local variables within functions are stored in storage or memory pass default, depending on their type. uninitialized local storage variables could point to other unexpected storage variables in the contract, leading to intentional or unintentional vulnerabilities.

- **Audit Results : Passed**

4.3.33 Frozen Account bypass

- **Risk Description**

In the transfer operation code in the contract, detect the risk that the logical functionality to check the freeze status of the transfer account exists in the contract code and can be passpassed if the transfer account has been frozen.

- **Audit Results : Passed**

4.3.34 Uninitialized

- **Risk description**

The initialize function in the contract can be called pass another attacker before the owner, thus initializing the administrator address.

- **Audit Results : Passed**

4.3.35 Integer Overflow

- **Risk Description**

Integer overflows are generally classified as overflows and underflows. The types of integer overflows that occur in smart contracts include three types: multiplicative overflows, additive overflows, and subtractive overflows. In Solidity language, variables support integer types in steps of 8, from uint8 to uint256, and int8 to int256, integers specify fixed size data types and are unsigned, for example, a uint8 type , can only be stored in the range 0 to 2^8-1, that is, [0,255] numbers, a uint256 type can only store numbers in the range 0 to 2^256-1. This means that an integer variable can only have a certain range of numbers represented, and cannot exceed this formulated range. Exceeding the range of values expressed pass the variable type will result in an integer overflow vulnerability.

- **Audit Results : Passed**

5. Security Audit Tool

Tool name	Tool Features
Oyente	Can be used to detect common bugs in smart contracts
securify	Common types of smart contracts that can be verified
MAIAN	Multiple smart contract vulnerabilities can be found and classified
Lunaray Toolkit	self-developed toolkit

Disclaimer：

Lunaray Technology only issues a report and assumes corresponding responsibilities for the facts that occurred or existed before the issuance of this report, Since the facts that occurred after the issuance of the report cannot determine the security status of the smart contract, it is not responsible for this.

Lunaray Technology conducts security audits on the security audit items in the project agreement, and is not responsible for the project background and other circumstances, The subsequent on-chain deployment and operation methods of the project party are beyond the scope of this audit.

This report only conducts a security audit based on the information provided by the information provider to Lunaray at the time the report is issued, If the information of this project is concealed or the situation reflected is inconsistent with the actual situation, Lunaray Technology shall not be liable for any losses and adverse effects caused thereby.

There are risks in the market, and investment needs to be cautious. This report only conducts security audits and results announcements on smart contract codes, and does not make investment recommendations and basis.

https://lunaray.co

https://github.com/lunaraySec

https://twitter.com/lunaray_Sec

http://t.me/lunaraySec