LUNARAY
BLOCKCHAINSECURITY

SMART CONTRACT SECURITY AUDIT REPORT

For EL DORADO EXCHANGE 2.0

06 March 2023

lunaray.co

Table of Contents

1. Overview....4
2. Background....5
   2.1 Project Description....5
   2.2 Audit Range....6
3. Project contract details....7
   3.1 Contract Overview....7
   3.2 Contract details....10
4. Audit details....30
   4.1 Findings Summary....30
   4.2 Risk distribution....31
   4.3 Risk audit details....33
      4.3.1 Administrator Permissions....33
      4.3.2 Dead Code....34
      4.3.3 Redundant codes....36
      4.3.4 Variable Override....38
      4.3.5 Variables are updated....38
      4.3.6 Floating Point and Numeric Precision....39
      4.3.7 Default Visibility....39
      4.3.8 tx.origin authentication....40
      4.3.9 Faulty constructor....40
      4.3.10 Unverified return value....41
      4.3.11 Insecure random numbers....41
      4.3.12 Timestamp Dependency....42
      4.3.13 Transaction order dependency....42
      4.3.14 Delegatecall....43
      4.3.15 Call....43
      4.3.16 Denial of Service....44
      4.3.17 Logic Design Flaw....44
      4.3.18 Fake recharge vulnerability....45

4.3.19 Short Address Attack Vulnerability 45
4.3.20 Uninitialized storage pointer 46
4.3.21 Frozen Account bypass 46
4.3.22 Uninitialized 46
4.3.23 Reentry Attack 47
4.3.24 Integer Overflow 47
9. Security Audit Tool 48

1. Overview

On Mar 2, 2023, 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 Mar 5, 2023. During the audit process, the security audit experts of Lunaray Technology and the EL DORADO EXCHANGE 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 EL DORADO EXCHANGE 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 EL DORADO EXCHANGE smart contract security audit: **Passed**

Audit Report Hash:
D14CE3443AD2C284FB361CA8B78A64A53547307354BA269BC4C052A0255E5342

2. Background

2.1 Project Description

Project name	El Dorado Exchange
Contract type	Spot and perpetual social trading
Code language	Solidity
Public chain	Arbitrum
Project website	https://www.ede.finance/
Contract file	RewardRouter.sol PositionManager.sol Router.sol PositionRouter.sol VaultPriceFeedV21Fast.sol VaultUtils.sol Vault.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

Smart contract file name and corresponding SHA256：

Name	SHA256
RewardRouter.sol	8F491D0896054A0FD8C8A9478A38141206B0804AD07CBEB65BEDB112170AE904
PositionManager.sol	4D9B515AF3E88BF04EE59AF05CBEB244C6D5EAF2CED92C6E39A5E1BCE19A7BD3
Router.sol	5757EBABB52BD6D6070D56B732D80912859A175E79BF176527A155951120F162
PositionRouter.sol	F6B2C96B8380DCC18FF7EDAFFDE77A4AFE193F680095C56932C611335FCB1162
VaultPriceFeedV21Fast.sol	745DE0C41B50D2AC114D1FEE5194CD5D794D6B9ED5AC83D7E212AE1A938DBB49
VaultUtils.sol	2D9C780098E1FF578065F706CA69B79EE8738E39AC1F05B0015CC17B87540021
Vault.sol	F084EBA9AD40CBD5114D287D8198565B9709DD3560F341E325743E6D19A631E5

3. Project contract details

3.1 Contract Overview

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.

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.

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.

increase and reduce positions through ETH, reduce positions and convert them into user-specified tokens or ETH and so on.

VaultPriceFeedV21Fast Contract

The VaultPriceFeedV21Fast contract mainly provides a price feed mechanism and is called by other contracts to query token prices. The main functions of the contract include the Owner setting the price update time, token address, Token configuration, etc.; The Updater role can update token prices and perform corresponding position increase/decrease operations for price changes. The contract also implements a large number of signature-related pure functions; users can query token prices, obtain initial prices, obtain on-chain prices, obtain recent prices, obtain secondary prices, etc.

Vault 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.

VaultUtils 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.

3.2 Contract details

RewardRouter Contract

Name	Parameter	Attributes
initialize	address _rewardToken address _eusd address _weth address _pricefeed uint256 _base_fee_point	onlyOwner
setRewardToken	address _rewardToken	onlyOwner
setPriceFeed	address _token bool _status	onlyOwner
adjustProfit	address _account uint256 val_1 uint256 val_2	onlyOwner
setESBT	address _esbt	onlyOwner
setBaseFeePoint	uint256 _base_fee_point	onlyOwner
setCooldownDuration	uint256 _setCooldownDuration	onlyOwner
setTokenConfig	address _token uint256 _token_decimal address _elp_n bool _isStable	onlyOwner
delToken	address _token address _elp_n	onlyOwner
setSwapToken	address _token bool _status	onlyOwner
setELPn	address _elp_n uint256 _elp_n_weight address _stakedELPnVault uint256 _elp_n_decimal address _stakedElpTracker	onlyOwner
clearELPn	address _elp_n	onlyOwner
withdrawToken	address _token	onlyOwner

	address _account uint256 _amount	
stakedELPnAmount	none	external
stakeELPn	address _elp_n uint256 _elpAmount	external
unstakeELPn	address _elp_n 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
claimableESBTEUSD	address _account	external
claimESBTEUSD	none	public
_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
swapCollateral	none	public
sellEUSD	address _token uint256 _EUSDamount	public
sellEUSDNative	uint256 _EUSDamount	public
_sellEUSD	address _account address _token uint256 _EUSDamount	internal
getELPnList	none	external
getEUSDPoolInfo	none	external
getEUSDCollateralDetail	none	external

PositionRouter Contract

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

cancelDecreasePosition	bytes32 _key address payable _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	onlyOwner
setLiquidator	address _account bool _isActive	onlyOwner
setPartner	address _account bool _isActive	onlyOwner
setInLegacyMode	bool _inLegacyMode	onlyOwner
decreasePosition	address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong address _receiver uint256 _price	onlyPartnersOrLegacyMode
decreasePositionETH	address _collateralToken address _indexToken uint256 _collateralDelta uint256 _sizeDelta bool _isLong address payable _receiver uint256 _price	onlyPartnersOrLegacyMode
liquidatePosition	address _account address _collateralToken address _indexToken bool _isLong address _feeReceiver	onlyLiquidator
executeSwapOrder	address _account uint256 _orderIndex address payable _feeReceiver	onlyOrderKeeper
executeIncreaseOrder	address _account uint256 _orderIndex address payable _feeReceiver	onlyOrderKeeper

executeDecreaseOrder	address _account uint256 _orderIndex address payable _feeReceiver	onlyOrderKeeper

Vault Contract

Name	Parameter	Attributes
initialize	address _usdx address _priceFeed uint8 _baseMode	onlyOwner
setVaultUtils	address _vaultUtils	onlyOwner
setVaultStorage	address _vaultStorage	onlyOwner
setESBT	address _eSBT	onlyOwner
setManager	address _manager bool _isManager	onlyOwner
setIsSwapEnabled	bool _isSwapEnabled	onlyOwner
setPriceFeed	address _priceFeed	onlyOwner
setRouter	address _router bool _status	onlyOwner
setUsdxAmount	address _token uint256 _amount bool _increase	onlyOwner
setTokenConfig	address _token uint256 _tokenDecimals uint256 _tokenWeight uint256 _maxUSDAmount bool _isStable bool _isFundingToken bool _isTradingToken	onlyOwner
clearTokenConfig	address _token	onlyOwner
upgradeVault	address _newVault address _token uint256 _amount	onlyOwner
buyUSDX	address _token address _receiver	onlyManager
sellUSDX	address _token address _receiver uint256 _usdxAmount	onlyManager
claimFeeToken	address _token	onlyManager

claimFeeReserves	none	onlyManager
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	bytes32 key uint256 _collateralDelta uint256 _sizeDelta address _receiver	private
liquidatePosition	address _account address _collateralToken address _indexToken bool _isLong address _feeReceiver	external
directPoolDeposit	address _token	external
tradingTokenList	none	external
fundingTokenList	none	external
claimableFeeReserves	none	external
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
tokenDecimals	address _token	public
getPositionStructByKey	bytes32 _key	public
getPositionStruct	address _account address _collateralToken address _indexToken bool _isLong	public
getTokenBase	address _token	public
getTradingRec	address _token	public
isFundingToken	address _token	public
isTradingToken	address _token	public
getTradingFee	address _token	public
getUserKeys	address _account uint256 _start uint256 _end	external
getKeys	uint256 _start uint256 _end	external
updateRate	address _token	public
_swap	address _tokenIn address _tokenOut address _receiver	private
_reduceCollateral	bytes32 _key	private

	uint256 _collateralDelta uint256 _sizeDelta uint256 _price	
_validatePosition	uint256 _size uint256 _collateral	private
_collectSwapFees	address _token uint256 _amount uint256 _feeBasisPoints	private
_collectMarginFees	bytes32 _key uint256 _sizeDelta	private
_collectFeeResv	address _account address _collateralToken uint256 _marginFees uint256 _feeTokens	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
_validate	bool _condition uint256 _errorCode	private
_updateGlobalSize	bool _isLong address _indexToken	private

	uint256 _sizeDelta uint256 _price bool _increase	
_delPosition	address _account bytes32 _key	private
_increaseGuaranteedUsd	address _token uint256 _usdAmount	private
_decreaseGuaranteedUsd	address _token uint256 _usdAmount	private

Router Contract

Name	Parameter	Attributes
setESBT	address _esbt	onlyOwner
setValidateContract	bool _valid	onlyOwner
setInfoCenter	address _infCenter	onlyOwner
addPlugin	address _plugin	onlyOwner
removePlugin	address _plugin	onlyOwner
withdrawToken	address _account address _token uint256 _amount	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	external

	address _receiver uint256 _price	
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 uint256 _collateralDelta uint256 _sizeDelta bool _isLong address _receiver uint256 _price	private
_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

VaultPriceFeedV21Fast Contract

Name	Parameter	Attributes
adjustmentBasisPoints	address _token	external
isAdjustmentAdditive	address _token	external
setAdjustment	address _token bool _isAdditive uint256 _adjustmentBps	external
setSpreadBasisPoints	address _token uint256 _spreadBasisPoints	external
getOrigPrice	address _token	external
priceVariancePer1Million	address _token	external
getPrimaryPrice	address _token bool _maximise	external
increasePositionRequestKeysStart	none	external
decreasePositionRequestKeysStart	none	external
executeIncreasePositions	uint256 _count address payable _executionFeeReceiver	external
executeDecreasePositions	uint256 _count address payable _executionFeeReceiver	external
getRequestQueueLengths	none	external
setPriceMethod	uint8 _setT	onlyOwner
setPriceVariance	uint256 _priceVariance	onlyOwner
setSafePriceTimeGap	uint256 _gap	onlyOwner
setAdjustment	address _token bool _isAdditive uint256 _adjustmentBps	onlyOwner
setSpreadBasisPoints	address _token uint256 _spreadBasisPoints	onlyOwner
_getCombPrice	address _token	internal

	bool _maximise	
getOrigPrice	address _token	public
getChainlinkPrice	address _token bool _max	public
getPrimaryPrice	address _token bool _maximise	public
setUpdater	address _account bool _isActive	onlyOwner
setSignPrefixCode	address _updater uint256 _setCode	onlyOwner
setTimeTolerance	uint256 _tol	onlyOwner
setTokenChainlinkConfig	address _token address _chainlinkContract bool _isStrictStable	onlyOwner
addPositionRouter	address _positionRouter	onlyOwner
VerifyMessage	bytes32 _hashedMessage uint8 _v bytes32 _r bytes32 _s	public
splitSignature	bytes sig	public
recoverSigner	bytes32 _ethSignedMessageHash bytes _signature	public

VaultUtils Contract

Name	Parameter	Attributes
priceVariancePer1Million	address _token	external
setMaxProfitRatio	uint256 _setRatio	onlyOwner
setSpreadBasis	address _token uint256 _spreadBasis uint256 _maxSpreadBasis uint256 _minSpreadCalUSD	onlyOwner
setMaxGlobalSize	address _token uint256 _amountLong uint256 _amountShort	onlyOwner
setTradingLimit	address _token uint256 _maxShortSize uint256 _maxLongSize uint256 _maxSize uint256 _maxRatio uint256 _countMinSize	onlyOwner
setOnlyRouterSwap	bool _onlyRS	onlyOwner
setLiquidator	address _liquidator bool _isActive	onlyOwner
setInPrivateLiquidationMode	bool _inPrivateLiquidationMode	onlyOwner
setPremiumRate	uint256 _premiumBasisPoints int256 _posIndexMaxPoints int256 _negIndexMaxPoints uint256 _maxPremiumBasisErrorUSD	onlyOwner
setFundingRate	uint256 _fundingRateFactor uint256 _stableFundingRateFactor	onlyOwner
setMaxLeverage	uint256 _maxLeverage	onlyOwner
setTaxRate	uint256 _taxMax uint256 _taxTime	onlyOwner
getLatestFundingRatePerSec	address _token	public
hRateToSecRate	uint256 _comRate	public

hRateToSecRateInt	int256 _comRate	public
getLatestLSRate	address _token	public
updateRate	address _token	public
getNextIncreaseTime	uint256 _prev_time uint256 _prev_size uint256 _sizeDelta	public
validateIncreasePosition	address _collateralToken address _indexToken uint256 _size uint256 _sizeDelta bool _isLong	external
validateDecreasePosition	VaultMSData.Position _position uint256 _sizeDelta uint256 _collateralDelta	external
getPositionKey	address _account address _collateralToken address _indexToken bool _isLong uint256 _keyID	public
getPositionInfo	address _account address _collateralToken address _indexToken bool _isLong	public
getPositionsInfo	uint256 _start uint256 _end	public
getNextAveragePrice	uint256 _size uint256 _averagePrice uint256 _nextPrice uint256 _sizeDelta bool _isIncrease	public
getInitialPosition	address _account address _collateralToken address _indexToken uint256 _sizeDelta bool _isLong uint256 _price	public

getPositionNextAveragePrice	uint256 _size uint256 _averagePrice uint256 _nextPrice uint256 _sizeDelta bool _isIncrease	public
calculateTax	uint256 _profit uint256 _aveIncreaseTime	public
validateLiquidation	bytes32 _key bool _raise	public
validateLiquidationPar	address _account address _collateralToken address _indexToken bool _isLong bool _raise	public
_validateLiquidation	VaultMSData.Position position bool _raise	public
getPositionImpactRatio	address _token uint256 _size	public
getImpactedPrice	address _token uint256 _sizeDelta uint256 _price bool _isLong	public
getFundingFee	VaultMSData.Position _position VaultMSData.TradingFee _tradingFee	public
getPremiumFee	VaultMSData.Position _position VaultMSData.TradingFee _tradingFee	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	public

	uint256 _taxBasisPoints bool _increment	
_validate	bool _condition uint256 _errorCode	private
getTradingTax	address _token	public
getTradingLimit	address _token	public
tokenUtilization	address _token	public
getTargetUsdxAmount	address _token	public
validLiq	address _account	public

4. Audit details

4.1 Findings Summary

Severity	Found	Resolved	Acknowledged
High	**0**	**0**	0
Medium	0	0	0
Low	1	0	1
Info	2	1	1

4.2 Risk distribution

Name	Risk level	Repair status
Administrator Permissions	Low	Acknowledged
Dead Code	Info	Resolved
Redundant codes	Info	Acknowledged
Variable Override	No	normal
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
Reentry attack	No	normal
Integer Overflow	No	normal

4.3 Risk audit details

4.3.1 Administrator Permissions

- **Risk description**

Currently in the contract, only the Owner administrator can set contract-related parameters, which may affect the stability of the project market when the administrator is maliciously manipulated or the private key is leaked.

```
function withdrawToken(
    address _account,
    address _token,
    uint256 _amount
) external onlyOwner{
    IERC20(_token).safeTransfer(_account, _amount);
}

function setUsdxAmount(
    address _token,
    uint256 _amount,
    bool _increase
) external override onlyOwner {
    if (_increase) _increaseUsdxAmount(_token, _amount);
    else _decreaseUsdxAmount(_token, _amount);
}
```

- **Safety advice**

It is recommended to use multi-signature contracts to control administrator privileg es, or destroy administrator privileges after the contract is chained.

- **Repair Status**

EL DORADO EXCHANGE has Acknowledged.

4.3.2 Dead Code

- **Risk description**

Dead code refers to code that will never be executed. It may be due to logical or programming errors or it may be outdated and no longer used code. In Solidity contracts, dead code may result in increased gas costs.

When calculating the value of premiumFee, since useNegativeRate is always false, the function returns 0 when _accumPremiumRate is less than 0. Therefore, the vaultUtils.getPremiumFee() function always returns a value that is not less than 0, so the else if(_premiumFee < 0) function may not be executed.

```
function getPremiumFee(VaultMSData.Position memory _position, VaultMSDa
ta.TradingFee memory _tradingFee) public view override returns (int256)
 {
    if (_position.size == 0 || _position.lastUpdateTime == 0)
        return 0;
    // VaultMSData.TradingFee memory _tradingFee = vault.getTradingFee
(_position.indexToken);
    int256 _accumPremiumRate = _position.isLong ? _tradingFee.accumulat
iveLongRateSec : _tradingFee.accumulativeShortRateSec;
    int256 _useFeePerSec  = _position.isLong ? _tradingFee.longRatePerS
ec : _tradingFee.shortRatePerSec;
    _accumPremiumRate += _useFeePerSec * int256((block.timestamp.sub(_t
radingFee.latestUpdateTime)));
    _accumPremiumRate -= _position.entryPremiumRateSec;
    if (!useNegativeRate && _accumPremiumRate < 0)
        _accumPremiumRate = 0;
    return int256(_position.size) * _accumPremiumRate / int256(VaultMSD
ata.PRC_RATE_PRECISION);
}

function _collectMarginFees(bytes32 _key, uint256 _sizeDelta)
    private
    returns (int256)
{
    VaultMSData.Position storage _position = positions[_key];
    int256 _premiumFee = vaultUtils.getPremiumFee(
        _position,
        tradingFee[_position.indexToken]
    );
    _position.accPremiumFee += _premiumFee;
```

```
    if (_premiumFee > 0) {
        // uint256 tokenAmount = usdToTokenMin(_position.collateralToke
n, uint256(_premiumFee));
        // _increasePoolAmount(_position.collateralToken, tokenAmount);

        //for poolAmount: decrease & increase
        _validate(_position.collateral >= uint256(_premiumFee), 29);
        _decreaseGuaranteedUsd(
            _position.collateralToken,
            uint256(_premiumFee)
        );
        _position.collateral = _position.collateral.sub(
            uint256(_premiumFee)
        );
    } else if (_premiumFee < 0) {
        // uint256 tokenAmount = usdToTokenMin(_position.collateralToke
n, uint256(-_premiumFee));
        // _decreasePoolAmount(_position.collateralToken, tokenAmount);

        _increaseGuaranteedUsd(
            _position.collateralToken,
            uint256(-_premiumFee)
        );
        _position.collateral = _position.collateral.add(
            uint256(-_premiumFee)
        );
    }
    emit CollectPremiumFee(
        _position.account,
        _position.size,
        _position.entryPremiumRateSec,
        _premiumFee
    );
    uint256 feeUsd = vaultUtils.getPositionFee(
        _position,
        _sizeDelta,
        tradingFee[_position.indexToken]
    );
    _position.accPositionFee = _position.accPositionFee.add(feeUsd);
    uint256 fuFee = vaultUtils.getFundingFee(
        _position,
        tradingFee[_position.collateralToken]
    );
    _position.accFundingFee = _position.accFundingFee.add(fuFee);
    feeUsd = feeUsd.add(fuFee);
    uint256 feeTokens = usdToTokenMin(_position.collateralToken, feeUs
d);
```

```
    _validate(_position.collateral >= feeUsd, 29);
    _decreaseGuaranteedUsd(_position.collateralToken, feeUsd);
    _position.collateral = _position.collateral.sub(feeUsd);
    _decreasePoolAmount(_position.collateralToken, feeTokens);
    _collectFeeResv(
        _position.account,
        _position.collateralToken,
        feeUsd,
        feeTokens
    );
    emit CollectMarginFees(_position.collateralToken, feeUsd, feeToken
s);
    return _premiumFee + int256(feeUsd);
}
```

- **Safety advice**

One way to fix dead code is to simply delete unnecessary code. However, before dele ting dead code, it's best to carefully check the code and make sure it won't have any negative impact.

- **Repair Status**

EL DORADO EXCHANGE has Acknowledged.

4.3.3 Redundant codes

- **Risk description**

Code unrelated to the business may result in unnecessary processing fees.

```
function getInitialPosition(address _account, address _collateralToken,
 address _indexToken, uint256 _sizeDelta, bool _isLong, uint256 _price)
 public override view returns (VaultMSData.Position memory){
    VaultMSData.Position memory position;
    position.account = _account;
    position.averagePrice = _price;
    position.aveIncreaseTime = block.timestamp;
    position.collateralToken = _collateralToken;
    position.indexToken = _indexToken;
    position.isLong = _isLong;
    return position;
}
```

```
function getLiqPrice(bytes32 /*_key*/) public view override returns (in
t256){
    // (uint256 size, uint256 collateral, uint256 averagePrice, uint256
 entryFundingRate, , , , ) =vault.getPositionByKey(_key);
    // if (size < 1) return size;
    // uint256 _fees = getFundingFee(positionsOrig[_key].account,positi
onsOrig[_key].collateralToken, positionsOrig[_key].indexToken, position
sOrig[_key].isLong, size, entryFundingRate);
    // _fees = _fees.add(getPositionFee(positionsOrig[_key].account, po
sitionsOrig[_key].collateralToken, positionsOrig[_key].indexToken, posi
tionsOrig[_key].isLong, size));
    // _fees = _fees.add(liquidationFeeUsd);
    // uint256 _maxLevCon = size.mul(BASIS_POINTS_DIVISOR).div(maxLever
age);
    // uint256 _tmpDelta = _maxLevCon > _fees ?_maxLevCon :  _fees;
    // _tmpDelta = averagePrice.mul(collateral.sub(_tmpDelta)).div(siz
e);
    // return positionsOrig[_key].isLong ? averagePrice.sub(_tmpDelta)
 : averagePrice.add(_tmpDelta);
    return 0;
}
```

- **Safety advice**

When going live in a production environment, code that is unrelated to the business logic should be removed.

- **Repair Status**

EL DORADO EXCHANGE has Acknowledged.

4.3.4 Variable Override

- **Risk description**

In Solidity, contract variables are stored in the order they are declared. So when a contract is upgraded using a proxy contract and the new contract changes the position of variables or modifies the position of previous variables, it may cause unexpected variable overwriting.

- **Audit Results : Passed**

4.3.5 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**

4.3.6 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**

4.3.7 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**

4.3.8 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**

4.3.9 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**

4.3.10 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**

4.3.11 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**

4.3.12 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**

4.3.13 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**

4.3.14 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**

4.3.15 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**

4.3.16 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**

4.3.17 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**

4.3.18 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**

4.3.19 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**

4.3.20 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**

4.3.21 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**

4.3.22 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**

4.3.23 Reentry Attack

- **Risk Description**

An attacker constructs a contract containing malicious code at an external address in the Fallback function When the contract sends tokens to this address, it will call the malicious code. The call.value() function in Solidity will consume all the gas he receives when it is used to send tokens, so a re-entry attack will occur when the call to the call.value() function to send tokens occurs before the actual reduction of the sender's account balance. The re-entry vulnerability led to the famous The DAO attack.

- **Audit Results : Passed**

4.3.24 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**

9. 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