LUNARAY
BLOCKCHAINSECURITY

SMART CONTRACT SECURITY AUDIT REPORT

For EDE elp-3

3 June 2023

lunaray.co

Table of Contents

1. Overview

On May 26, 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 Jun 3, 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:

54B2DEF91F3374CD49BD50BC2FB9B5B9A42129E430CFA8D6C25ED2616BF098DB

2. Background

2.1 Project Description

Project name	El Dorado Exchange ELP-3
Contract type	Spot and perpetual social trading
Code language	Solidity
Public chain	BNB Chain
Project website	https://www.ede.finance/
Contract file	Vault.sol VaultUtils.sol RouterSign.sol VaultPriceFeed.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
Vault.sol	1A4347733C84A38D121A6304B81D59D6B545871519C69A22C0C4C5E34F11AE16
VaultUtils.sol	B2EC77DE87FE72640FA44680D46DEE688C44B3F742A6C4D6940FDD3209C70417
RouterSign.sol	17B328CF5434EBE8038E99F88EFC234AE9A6A6812FA9C5EE36389FBDE9CB9ADD
VaultPriceFeed.sol	30794653C36271A747DE006FEECC2CA562361CB287A30488C85136D2A380A826

3. Project contract details

3.1 Contract Overview

Vault Contract

The Valut contract is the base contract for the whole system and is mainly used by other contracts. The roles of Owner, Manager, liquidator and normal user are called.

The Owner role can be called to initialize the contract, set the interface contract address, set the management mode, set the administrator liquidator, update the token address and quantity, and set the token address and quantity.

The Owner role can call the contract initialization function, set the interface contract address, set the management mode, set the administrator liquidator, update the token address and quantity, set the interest rate, set the Token configuration, etc.; the Manager role can call the contract for buying and selling.

The contracts that can be called by the Manager role are buy and sell USDX, calculate reserves, add and subtract positions, calculate rewards, etc.; the functions that can be called by the user are set

The functions that can be called by users are setting up Router, exchanging tokens, checking token information, etc.; the authority to call the clearing function is set by the Owner, when the Owner sets the clearing status to private

When the owner sets the clearing status to private mode, only the liquidator can execute the clearing operation; when the clearing status is set to non-private mode, all users can act as the liquidator to execute the clearing.

The new version of the code adds a swap function to the The new version of the code adds permission restrictions to the swap function so that only roles with permission can call it, and adds a new variable, tradingRec, to calculate the price impact when getting prices.

VaultUtils Contract

The ValitUtils contract is a complementary contract to the Valut contract, with the main functions of setting various interest rates, checking position information, obtaining

The main functions of the ValitUtils contract are to set various interest rates, check the position information, get the interest rate, calculate the commission for buying and selling USDX or exchange, check the liquidation, etc. If the liquidation criteria are not met, the liquidation fee is returned directly.

If the liquidation fee does not exceed the collateral, then some of the collateral will be liquidated and the remaining collateral will be set as the collateral limit.

If the liquidation cost exceeds the collateral, all collateral will be liquidated. In the new version, some of the privileged functions have been modified and the mechanism of validating data such as transaction size in the function of validating increased positions has been adjusted, and the getLiqPrice function has been refactored.

RouterSign Contract

This contract is a routing contract, mainly used for the invocation of transactions in smart contracts, including operations such as position increase, position decrease, and token exchange. Among them, the router contract implements trading operations by calling the vault contract, and obtains market price information through the priceFeed contract. In addition, some administrative functions are defined for administrator account operations. The function of this contract is to implement trading operations in the smart contract and provide the corresponding administrative functions. The main difference between this contract and the Router contract is that adding and reducing a position through this contract requires passing a specified price and updating the signature data of the price.

RouterSign Contract

The contract implements a Token price acquisition mechanism, providing a set of adjustable price setting parameters, as well as price conversion functions (e.g. `tokenToUsdUnsafe`, `usdToTokenUnsafe`) and some public view functions (e.g. `getPrimaryPrice` and `getPrice`). The contract is based on the Pyth and Server Oracle price settings, and the corresponding ERC20 token prices can be calculated. In addition, the contract provides some settings and administrator privileges, such as specifying Pyth Address, configuring tokens, setting maximum price adjustment limits, etc. The main function of the contract price acquisition source is based on the pyth price source, and according to the administrator's configuration to decide whether to fine-tune the price, and return the final price; Server Oracle's price source is mainly used to determine whether the pyth price is within the price difference set by the administrator, but does not affect the final price.

3.2 Contract details

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	onlyManager

	uint256 _usdxAmount	
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	private

	address _receiver	
_reduceCollateral	bytes32 _key uint256 _collateralDelta uint256 _sizeDelta uint256 _price	private
_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
_validateBufferAmoun t	address _token	private
_increaseUsdxAmount	address _token uint256 _amount	private
_decreaseUsdxAmount	address _token uint256 _amount	private
_increaseReservedAm ount	address _token uint256 _amount	private
_decreaseReservedAm ount	address _token uint256 _amount	private
_validate	bool _condition uint256 _errorCode	private

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

VaultUtils Contract

Name	Parameter	Attributes
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	public

	bool _isLong	
getLiqPrice	bytes32 _key	public
getPositionsInfo	uint256 _start uint256 _end	public
getNextAveragePrice	uint256 _size uint256 _averagePrice uint256 _nextPrice uint256 _sizeDelta bool _isIncrease	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
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 uint256 _taxBasisPoints bool _increment	public
_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

VaultPriceFeed Contract

Name	Parameter	Attributes
getPrice	address _token	external
queryPriceFeed	bytes32 id	external
priceFeedExists	bytes32 id	external
getValidTimePeriod	none	external
getPrice	bytes32 id	external
getPriceUnsafe	bytes32 id	external
setServerOracle	address _pyth address _serverOra	onlyOwner
setGap	uint256 _priceSafetyTimeGap uint256 _stopTradingPriceGap	onlyOwner

setAdjustment	address _token bool _isAdditive uint256 _adjustmentBps	onlyOwner
setSpreadBasisPoints	address _token uint256 _spreadBasisPoints	onlyOwner
isSupportToken	address _token	public
priceTime	address _token	external
_getCombPrice	address _token bool _maximise bool _addAdjust	internal
_addBasisSpread	address _token uint256 _price bool _max bool _addAdjust	internal
getConvertedPyth	address _token	public
getPythPrice	address _token	public
getPrimaryPrice	address _token	public
tokenToUsdUnsafe	address _token uint256 _tokenAmount bool _max	public
usdToTokenUnsafe	address _token uint256 _usdAmount bool _max	public

RouterSign Contract

Name	Parameter	Attributes
initialize	address _vault address _weth address _priceFeed address _esbt	onlyOwner
setESBT	address _esbt	onlyOwner
setPriceFeed	address _priceFeed	onlyOwner
setVault	address _vault	onlyOwner
withdrawToken	address _account address _token uint256 _amount	onlyOwner
sendValue	Address _receiver uint256 _amount	onlyOwner
_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_receiver	private
_vaultSwap	address _tokenIn address _tokenOut uint256 _minOut address _receiver	private
_sender	none	private

4. Audit details

4.1 Findings Summary

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

4.2 Risk distribution

Name	Risk level	Repair status
Administrator Permissions	Low	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
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 sendValue(address payable _receiver, uint256 _amount) external
onlyOwner {
    _receiver.sendValue(_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 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**

4.3.3 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**

4.3.4 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**

4.3.5 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**

4.3.6 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**

4.3.7 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**

4.3.8 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**

4.3.9 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**

4.3.10 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**

4.3.11 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**

4.3.12 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**

4.3.13 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**

4.3.14 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**

4.3.15 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**

4.3.16 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**

4.3.17 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**

4.3.18 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**

4.3.19 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**

4.3.20 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**

4.3.21 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.

LUNARAY
BLOCKCHAINSECURITY

https://lunaray.co

https://github.com/lunaraySec

https://twitter.com/lunaray_Sec

http://t.me/lunaraySec