LUNARAY
BLOCKCHAINSECURITY

SMART CONTRACT SECURITY AUDIT REPORT

For RealPerp

15 November 2023

lunaray.co

Table of Contents

1. Overview

On Nov. 14, 2023, the security team of Lunaray Technology received the security audit request of the **REALPERP project**. The team completed the audit of the **REALPERP smart contract** on Nov. 15, 2023. During the audit process, the security audit experts of Lunaray Technology and the REALPERP 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 REALPERP 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 REALPERP smart contract security audit: **Passed**

Audit Report Hash:

BD072CE5D480A339DAF7229925D333DA9AD4F4F51797ACCFF17F6DF637A461FE

2. Background

2.1 Project Description

Project name	RealPerp
Contract type	Decentralized exchange
Code language	Solidity
Public chain	Manta Pacific
Project website	http://realperp.com
Introduction	Realperp is a decentralized swap and perpetual DEX on Manta Pacific, offering diverse trading options and high liquidity for blue chip crypto assets. It's designed for traders prioritizing capital control and superior trading experiences. With low swap fees and minimal price impact trades, Realperp enhances its efficient trading environment through unique multi-asset pools that facilitate trading and generate earnings for liquidity providers from market making, swap fees, and leverage trading.
Contract file	RewardRouterOrderBook.sol, VaultPriceFeedV3.sol FastPriceFeedV3.sol PriceWrapper.sol

2.2 Audit Range

Smart contract file name and corresponding SHA256:

Based on repository commit: 5b6e1b564f025d9eb562c05513534e48d338bc4e

Name	SHA256
RewardRouterOrderBook.sol	1C7A293C694EED19D370403D1CE1511C9B5D18D6DAE49A1E2D15DB098C6A8F0C
VaultPriceFeedV3.sol	63CBEF79D9F2A8999AD9439849CA105475ABCF6044D7B664B42FD4A88BC52020
FastPriceFeedV3.sol	E8EAC555A8E12BB67F53038BA42E2DDACFFF2C7CEE66D6E704188155C9DC144A
PriceWrapper.sol	5F9ADFA1937443003AC124A754BD09CF48C345CE906A7697106D7E201E59915F

3. Project contract details

3.1 Contract Overview

RewardRouterOrderBook Contract

The contract implements an order book that handles mint, stake, unstake and redeem operations for the user. The contract allows the user to perform transactions in the RealPerp protocol. This includes creating and executing mint and stake GLP requests, and creating and executing unstake and redeem GLP requests. It also includes functions to set the minimum execution cost, set the delay value, and set the execution cost receiver.

VaultPriceFeedV3 Contract

The contract implements the configuration and management of the price source, and the contract contains the logic for price calculation, including the maintenance of price adjustments for the project side. In addition, the contract also includes the setting and management of parameters such as price threshold, price space, price deviation, etc. Compared with the old version, the new version removes the Pyth price source and uses off-chain price data instead, and the price source is mainly the project side backend server.

FastPriceFeedV3 Contract

The main function implemented in this contract is to provide token price data and allow for the addition of extended information such as cumulative price change data, in short the contract is a prediction machine contract with price data provided by the project. The new version adds a function for liquidity order execution and the ability to maintain the latest price of a given token via PriceWrapper.

PriceWrapper Contract

The contract implements a storage contract for price data, stores prices and confidence intervals for individual tokens, provides an interface for external contracts to query prices, and restricts only authorized roles to update price data.

3.2 Contract details

RewardRouterOrderBook Contract

Name	Parameter	Attributes
initialize	address _router address _weth address _glpManager address _glp address _feeGlpTracker address _stakedGlpTracker uint256 _minExecutionFee	onlyGov
setMinExecutionFee	uint256 _minExecutionFee	onlyKeeper
setKeeper	address _keeper bool _isActive	onlyGov
createMintAndStakeGlpRequest	address _token uint256 _amountIn uint256 _minUsdg uint256 _minGlp uint256 _executionFee bool _shouldWrap	external
createUnstakeAndRedeemGlpRequest	address _tokenOut uint256 _glpAmount uint256 _minOut address _receiver uint256 _executionFee bool _withdrawETH	external
_createMintAndStakeGlpRequest	address _account address _token uint256 _amountIn uint256 _minUsdg uint256 _minGlp uint256 _executionFee bool _shouldWrap	internal
_createUnstakeAndRedeemGlpRequest	address _account address _tokenOut	internal

	uint256 _glpAmount uint256 _minOut address _receiver uint256 _executionFee bool _withdrawETH	
executeMintAndStakeGlpRequests	uint256 _endIndex address payable _executionFeeReceiver	onlyKeeper
executeUnstakeAndRedeemGlpRequests	uint256 _endIndex address payable _executionFeeReceiver	onlyKeeper
executeMintAndStakeGlpRequest	bytes32 _key address payable _executionFeeReceiver	public
executeUnstakeAndRedeemGlpRequest	bytes32 _key address payable _executionFeeReceiver	public
cancelMintAndStakeGlpRequest	bytes32 _key address payable _executionFeeReceiver	public
cancelUnstakeAndRedeemGlpRequest	bytes32 _key address payable _executionFeeReceiver	public
setDelayValues	uint256 _maxBlockDelay uint256 _maxTimeDelay	onlyGov
_validateExecution	uint256 _requestBlockNumber uint256 _requestBlockTime address _account	internal
_validateCancellation	uint256 _requestBlockNumber uint256 _requestBlockTime address _account	internal
_validateExecutionOrCancellation	uint256 _requestBlockNumber uint256 _requestBlockTime address _account	internal
_transferInETH	none	private
_storeMintAndStakeGlpRequest	MintAndStakeGlpRequest _request	internal
_storeUnstakeAndRedeemGlpRequest	UnstakeAndRedeemGlpRequest _request	internal
getRequestKey	address _account uint256 _index	public

_transferOutETHWithGasLimitFallbackToWeth	uint256 _amountOut address payable _receiver	internal
setEthTransferGasLimit	uint256 _ethTransferGasLimit	onlyGov
approve	address _token address _spender uint256 _amount	private
withdrawToken	address _token address _account uint256 _amount	onlyGov
getRequestQueueLengths	none	external

VaultPriceFeedV3 Contract

Name	Parameter	Attributes
setGov	address _gov	onlyGov
setPriceHandler	address _handler bool active	onlyGov
setPythFlags	address _pythFlags	onlyGov
setAdjustment	address _token bool _isAdditive uint256 _adjustmentBps	onlyGov
setUseV2Pricing	bool _useV2Pricing	onlyGov
setIsAmmEnabled	bool _isEnabled	onlyGov
setIsPriceEnabled	bool _isEnabled	onlyPriceHandler
setIsSecondaryPriceEnabled	bool _isEnabled	onlyGov
setSecondaryPriceFeed	address _secondaryPriceFeed	onlyGov
setTokens	address _btc address _eth address _bnb	onlyGov
setPairs	address _bnbBusd address _ethBnb	onlyGov

	address _btcBnb	
setSpreadBasisPoints	address _token uint256 _spreadBasisPoints	onlyGov
setSpreadThresholdBasisPoints	uint256 _spreadThresholdBasisPoints	onlyGov
setFavorPrimaryPrice	bool _favorPrimaryPrice	onlyGov
setPriceSampleSpace	uint256 _priceSampleSpace	onlyGov
setMaxStrictPriceDeviation	uint256 _maxStrictPriceDeviation	onlyGov
setTokenConfig	address _token address _priceFeed uint256 _priceDecimals bool _isStrictStable	onlyGov
getPriceV1	address _token bool _maximise bool _includeAmmPrice	public
getPriceV2	address _token bool _maximise bool _includeAmmPrice	public
getAmmPriceV2	address _token bool _maximise uint256 _primaryPrice	public
getLatestPrimaryPrice	address _token	public
getPrimaryPrice	address _token bool _maximise	public
getSecondaryPrice	address _token uint256 _referencePrice bool _maximise	public
getAmmPrice	address _token	public
getPairPrice	address _pair bool _divByReserve0	public

FastPriceFeedV3 Contract

Name	Parameter	Attributes
setSigner	address _account bool _isActive	onlyGov
setUpdater	address _account bool _isActive	onlyGov
setFastPriceEvents	address _fastPriceEvents	onlyGov
setPriceWrapper	address _priceWrapper	onlyTokenManager
setVaultPriceFeed	address _vaultPriceFeed	onlyGov
setMaxTimeDeviation	uint256 _maxTimeDeviation	onlyGov
setPriceDuration	uint256 _priceDuration	onlyGov
setMaxPriceUpdateDelay	uint256 _maxPriceUpdateDelay	onlyGov
setSpreadBasisPointsIfInactive	uint256 _spreadBasisPointsIfInactive	onlyGov
setSpreadBasisPointsIfChainError	uint256 _spreadBasisPointsIfChainError	onlyGov
setMinBlockInterval	uint256 _minBlockInterval	onlyGov
setIsSpreadEnabled	bool _isSpreadEnabled	onlyGov
setLastUpdatedAt	uint256 _lastUpdatedAt	onlyGov
setTokenManager	address _tokenManager	onlyTokenManager
setMaxDeviationBasisPoints	uint256 _maxDeviationBasisPoints	onlyTokenManager
setPriceDataInterval	uint256 _priceDataInterval	onlyTokenManager
setMinAuthorizations	uint256 _minAuthorizations	onlyTokenManager
setPricesWithBits	uint256 _priceBits uint256 _timestamp	onlyUpdater
disableFastPrice	none	onlySigner
enableFastPrice	none	onlySigner
getPrice	address _token uint256 _refPrice	external

	bool _maximise	
favorFastPrice	address _token	public
getPriceData	address _token	public
_setPricesWithBits	uint256 _priceBits uint256 _timestamp	private
_setPrice	address _token uint256 _price address _vaultPriceFeed address _fastPriceEvents	private
_setPriceData	address _token uint256 _refPrice uint256 _cumulativeRefDelta uint256 _cumulativeFastDelta	private
_emitPriceEvent	address _fastPriceEvents address _token uint256 _price	private
_setLastUpdatedValues	uint256 _timestamp	private

PriceWrapper Contract

Name	Parameter	Attributes
setHandler	address _handler bool _isActive	onlyGov
setPrice	address[] memory _tokens uint[] memory _price uint[] memory _conf	onlyHandler
getMaxPrice	address token	external
getMinPrice	address token	external
getMedianPrice	address token	external

4. Audit details

4.1 Findings Summary

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

4.2 Risk distribution

Name	Risk level	Repair status
Price Control	Medium	Acknowledged
Administrator Permissions	Low	Acknowledged
Logic design flaw	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
Denial of Service	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 Price Control

- **Risk description**

Price manipulation usually refers to the practice of some large investors or traders of buying or selling large quantities of a particular asset to influence the price of that asset and then capitalizing on the price change to make a profit. This behavior undermines the fairness of the market and creates an uneven playing field for smaller investors.

The price source is adjusted from the previous Pyth price + off-chain price (maintained by the project side) to the off-chain price (maintained by the project side), whose price increase right is 100% controlled by the back-end service of the project side, so there is a possibility of price manipulation caused by hacker attacks or inside ghosts of the project side, which may lead to malicious attacks.

```
function getPrimaryPrice(
    address _token,
    bool _maximise
) public view override priceEnabled returns (uint256) {
    address priceFeedAddress = priceFeeds[_token];
    require(
        priceFeedAddress != address(0),
        "VaultPriceFeed: invalid price feed"
    );
    if (pythFlags != address(0)) {
        bool isRaised = IPythFlags(pythFlags).getFlag(
            FLAG_MANTA_SEQ_OFFLINE
        );
        if (isRaised) {
            // If flag is raised we shouldn't perform any critical
operations
            revert("Pyth feeds are not being updated");
        }
    }
    IPriceWrapper priceFeed = IPriceWrapper(priceFeedAddress);
    uint256 price = 0;
    if (_maximise) {
```

```
        price = priceFeed.getMaxPrice(_token);
    } else {
        price = priceFeed.getMinPrice(_token);
    }
    require(price > 0, "VaultPriceFeed: could not fetch price");
    // normalise price precision
    uint256 _priceDecimals = priceDecimals[_token];
    return price.mul(PRICE_PRECISION).div(10 ** _priceDecimals);
}
```

- **Safety advice**

It is recommended that relatively credible price sources on the chain be used at the same time to prevent 100% price control.

- **Repair Status**

REALPERP has Acknowledged.

4.3.2 Administrator Permissions

- **Risk description**

Contract administrator privileges are people who can change the status of a contract or perform certain sensitive operations. If administrator privileges are abused or hacked, the contract may be subject to several risks. First, a hacker may be able to obtain the administrator's private key, thus hijacking the administrator's account and thus having full control of the contract. The hacker can change the status of the contract or perform illegal operations at will, causing a serious loss of contract assets. Second, the administrator may intentionally or unintentionally disclose his or her private key, causing others to gain access. Hackers may obtain the administrator's private key through social engineering or trickery, and obtain a large amount of sensitive contract information, which can then be used to carry out attacks. Third, administrators may use their privileges to perform improper operations. For example, an administrator may change the status of a contract for financial gain, or change the execution rules of a contract to gain additional control, and so on. These actions may also lead to loss of contract assets or illegitimate domination.

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

- **Safety advice**

To ensure contract security, contract administrators must take a variety of measures to protect contracts, such as avoiding the use of EOA addresses or reasonable private key management, such as multi-signature wallets or multi-signature contracts for privilege control and other actions. Or remove administrator privileges after the project is launched.

- **Repair Status**

REALPERP has Acknowledged.

4.3.3 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**

4.3.4 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**

4.3.5 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**

4.3.6 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**

4.3.7 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**

4.3.8 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**

4.3.9 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**

4.3.10 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**

4.3.11 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**

4.3.12 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**

4.3.13 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**

4.3.14 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**

4.3.15 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**

1. 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_co

http://t.me/lunaraySec