LUNARAY
BLOCKCHAINSECURITY

SMART CONTRACT SECURITY AUDIT REPORT

For RealPerp

21 October 2023

lunaray.co

Table of Contents

1. Overview

On Oct 19, 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 Oct 21, 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:
4BC10C1A4D14AAB2E60D1F2E49DB1F8ADDBB95B1C8ED0B5AB746023133F311AC

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	VaultPriceFeedV2.sol, FastPriceFeedV2.sol

2.2 Audit Range

Smart contract file name and corresponding SHA256：

Name	SHA256
VaultPriceFeedV2.sol	B7A2C2A4673A67C67233ACC84E8EC77E9021023C17C306CA1C7B02E215C4B414
FastPriceFeedV2.sol	90392899E7822EC761000DA78CDED65BB28DB19026AC516E795E1414B7F7B7CE

3. Project contract details

3.1 Contract Overview

VaultPriceFeedV2 Contract

The contract is primarily used to obtain price data for different tokens. The contract provides multiple price sources, including Pyth, AMM and carefully maintained price sources by the project, which can be enabled or disabled based on configuration parameters set by the administrator.

FastPriceFeedV2 Contract

The main implemented functionality of the contract is to provide token price data and to allow adding extended information such as cumulative price change data. It also allows managing update policies for price data, including time intervals and delays, as well as configuring extension parameters for price updates. In addition, the contract supports enabling/disabling price data extensions such as price variance and price monitoring, and allows administrators to manage signers, updaters, and other associated contracts. Specific functions include price data management, price extension information logging, price calculation, price update method setting, configuration parameter provisioning and administrative privilege control.

3.2 Contract details

VaultPriceFeedV2 Contract

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

	bool _includeAmmPrice	
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

FastPriceFeedV2 Contract

Name	Parameter	Attributes
setSigner	address _account bool _isActive	onlyGov
setUpdater	address _account bool _isActive	onlyGov
setFastPriceEvents	address _fastPriceEvents	onlyGov
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
setPricesWithBitsAndExecute	address _positionRouter uint256 _priceBits uint256 _timestamp uint256 _endIndexForIncreasePositions uint256 _endIndexForDecreasePositions uint256 _maxIncreasePositions uint256 _maxDecreasePositions	onlyUpdater
setPricesWithBitsAndExecuteLimitSwapOrder	address _positionManager uint256 _priceBits uint256 _timestamp address _account uint256 _orderIndex address payable _feeReceiver	onlyUpdater
setPricesWithBitsAndExecuteIncreaseOrder	address _positionManager uint256 _priceBits uint256 _timestamp address _account uint256 _orderIndex address payable _feeReceiver	onlyUpdater

setPricesWithBitsAndExecuteDecreaseOrder	address _positionManager uint256 _priceBits uint256 _timestamp address _account uint256 _orderIndex address payable _feeReceiver	onlyUpdater
setPricesWithBitsAndLiquidatePosition	address _positionManager uint256 _priceBits uint256 _timestamp address _account address _collateralToken address _indexToken bool _isLong address _feeReceiver	onlyUpdater
disableFastPrice	none	onlySigner
enableFastPrice	none	onlySigner
getPrice	address _token uint256 _refPrice bool _maximise	external
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

4. Audit details

4.1 Findings Summary

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

4.2 Risk distribution

Name	Risk level	Repair status
Administrator Permissions	Low	Acknowledged
Price Control	Info	Resolved
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 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.

```
contract FastPriceFeedV2 is ISecondaryPriceFeed, IFastPriceFeed,
Governable {
    function setSigner(
        address _account,
        bool _isActive
    ) external override onlyGov {...}
    function setUpdater(
        address _account,
        bool _isActive
    ) external override onlyGov {...}
    function setFastPriceEvents(address _fastPriceEvents) external
onlyGov {...}
    function setVaultPriceFeed(
        address _vaultPriceFeed
```

```
    ) external override onlyGov {...}
    function setMaxTimeDeviation(uint256 _maxTimeDeviation) external
onlyGov {...}
    function setPriceDuration(
        uint256 _priceDuration
    ) external override onlyGov {...}
    function setMaxPriceUpdateDelay(
        uint256 _maxPriceUpdateDelay
    ) external override onlyGov {...}
    function setSpreadBasisPointsIfInactive(
        uint256 _spreadBasisPointsIfInactive
    ) external override onlyGov {...}
    function setSpreadBasisPointsIfChainError(
        uint256 _spreadBasisPointsIfChainError
    ) external override onlyGov {...}
    function setMinBlockInterval(
        uint256 _minBlockInterval
    ) external override onlyGov {...}
    function setIsSpreadEnabled(
        bool _isSpreadEnabled
    ) external override onlyGov {...}
    function setLastUpdatedAt(uint256 _lastUpdatedAt) external
onlyGov {...}
    function setTokenManager(address _tokenManager) external
onlyTokenManager {...}
    function setMaxDeviationBasisPoints(
        uint256 _maxDeviationBasisPoints
    ) external override onlyTokenManager {...}
    function setMaxCumulativeDeltaDiffs(
        address[] memory _tokens,
        uint256[] memory _maxCumulativeDeltaDiffs
    ) external override onlyTokenManager {...}
    function setPriceDataInterval(
        uint256 _priceDataInterval
    ) external override onlyTokenManager {...}
    function setMinAuthorizations(
        uint256 _minAuthorizations
    ) external onlyTokenManager {...}
}
```

- **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.

4.3.2 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.

According to the contract logic, when some of the administrator configurations in the get-price function and the incoming parameters satisfy the corresponding conditions, it may result in malicious AMM price manipulation.

```
function getPrice(
    address _token,
    uint256 _refPrice,
    bool _maximise
) external view override returns (uint256) {
    if (block.timestamp > lastUpdatedAt.add(maxPriceUpdateDelay)) {
        if (_maximise) {
            return
                _refPrice
                    .mul(
                        BASIS_POINTS_DIVISOR.add(
                            spreadBasisPointsIfChainError
```

```
                        )
                    )
                    .div(BASIS_POINTS_DIVISOR);
        }
        return
            _refPrice
                .mul(
                    BASIS_POINTS_DIVISOR.sub(spreadBasisPointsIfChai
nError)
                )
                .div(BASIS_POINTS_DIVISOR);
    }
    if (block.timestamp > lastUpdatedAt.add(priceDuration)) {
        if (_maximise) {
            return
                _refPrice
                    .mul(
                        BASIS_POINTS_DIVISOR.add(
                            spreadBasisPointsIfInactive
                        )
                    )
                    .div(BASIS_POINTS_DIVISOR);
        }
        return
            _refPrice
                .mul(BASIS_POINTS_DIVISOR.sub(spreadBasisPointsIfIna
ctive))
                .div(BASIS_POINTS_DIVISOR);
    }
    uint256 fastPrice = prices[_token];
    if (fastPrice == 0) {
        return _refPrice;
    }
    uint256 diffBasisPoints = _refPrice > fastPrice
        ? _refPrice.sub(fastPrice)
        : fastPrice.sub(_refPrice);
    diffBasisPoints = diffBasisPoints.mul(BASIS_POINTS_DIVISOR).div(
        _refPrice
    );
```

```
    // create a spread between the _refPrice and the fastPrice if
the maxDeviationBasisPoints is exceeded
    // or if watchers have flagged an issue with the fast price
    bool hasSpread = !favorFastPrice(_token) ||
        diffBasisPoints > maxDeviationBasisPoints;
    if (hasSpread) {
        // return the higher of the two prices
        if (_maximise) {
            return _refPrice > fastPrice ? _refPrice : fastPrice;
        }
        // return the lower of the two prices
        return _refPrice < fastPrice ? _refPrice : fastPrice;
    }
    return fastPrice;
}
```

- **Safety advice**

It is recommended that the direct use of a single unprocessed price data source be avoided at the contract level, and data integration based on multiple price sources is recommended, e.g., the use of weighted average prices, weighted median prices, or other means of integrating price data.

- **Repair Status**

REALPERP has Resolved.

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.

https://lunaray.co

https://github.com/lunaraySec

https://twitter.com/lunaray_Sec

http://t.me/lunaraySec