SECURITY AUDIT OF

KELP DELTA NEUTRAL RESTAKING VAULT

Public Report

Aug 29, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Aug 29, 2024. We would like to thank the Harmonix Finance for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Kelp Delta Neutral Restaking Vault. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Kelp Delta Neutral Restaking Vault

The Kelp Restaking Delta Neutral Vault on Harmonix is a decentralized finance (DeFi) product designed to provide a balanced, low-risk investment strategy. It leverages restaking mechanisms and a delta-neutral approach to minimize exposure to market volatility while generating returns. This vault is tailored for users seeking stable yields in the crypto space without significant risk from market fluctuations.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the Kelp Delta Neutral Restaking Vault.

It was conducted on commit `0c76341d553b7ce48b012da7bc4792ddbd3bb61d` from git repository *https://github.com/harmonixfi/core-smart-contract*

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
8416f6c044c1d84517f233e5dd1b12a306262f9142aa7ee9b2b4b26104890c20	contracts/vaults/restakingDeltaNeutral/Base/BaseDeltaNeutralVault.sol
ef1872cbc11d88d290e64c5fa737dd1f79b9b54e0c8ffdbfff92481866aa6ff8	contracts/vaults/restakingDeltaNeutral/Base/BaseSwapVault.sol
90edaefd83087e83ef73cc5b8862a0f3e851693cafd413bebaf647325f211e98	contracts/vaults/restakingDeltaNeutral/Base/strategies/BaseRestakingStrategy.sol
3f56b9d33475462b8178324a15df7f6e56e60fb5689a72eb6f35eb2dadc35277	contracts/vaults/restakingDeltaNeutral/Base/strategies/PerpDexStrategy.sol
850b46824adb3693b8e75c5939b6a88d10659c43f1870342a288047479f2312c	contracts/vaults/restakingDeltaNeutral/KelpZircuit/KelpRestakingDeltaNeutralVault.sol
1fc19705a93bcb4cc47b5f60a4fce16d78be6a6e099968aa695983967e4967c2	contracts/vaults/restakingDeltaNeutral/KelpZircuit/strategies/KelpZircuitRestakingStrategy.sol

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.

- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Harmonix Finance acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Harmonix Finance understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Harmonix Finance agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Harmonix Finance will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Harmonix Finance, the final report will be considered fully accepted by the Harmonix Finance without the signature.

2. AUDIT RESULT

2.1. Overview

The Kelp Delta Neutral Restaking Vault are written in Solidity version ^0.8.19 and utilize `OpenZeppelin`'s upgradeable contract libraries, emphasizing modularity and security.

The project enables users to deposit and withdraw funds to earn yield returns. It channels user funds into two main strategies: 50% is staked in the Kelp Zircuit contract, while the remaining 50% is deposited on-chain for investment purposes. The contract earns interest from Kelp and generates profits from these investments, which are then distributed as rewards to all participants in the protocol.

The contract raises decentralization concerns as users need to place their trust in both the protocol and the contract owner. This trust is crucial because the funds are allocated for investment and staking in external protocols.

2.2. Findings

During the audit process, the audit team had identified some vulnerable issues in the contract code.

#	Title	Severity	Status
1	Wrong formula in `syncRestakingBalance` function	CRITICAL	FIXED
2	Unclear logic in calculate `totalShareAmount` in `initiateWithdrawal`	CRITICAL	FIXED
3	Missing mul `performancefee` decimals in `initiateWithdrawal` function	CRITICAL	FIXED
4	Looping through the array in `updateDepositArr` and `updateWithdrawalArr` causes the gas limit to be exceeded	HIGH	ACKNOWLEDGED
5	Not refund excess amount in `swapToWithOutput` function	HIGH	FIXED
6	Issue when `claimFee` too much totalFeePoolAmount	HIGH	FIXED

#	Title	Severity	Status
7	Centralization issue in `syncPerpDexBalance` function	HIGH	ACKNOWLEDGED
8	User-controlled `swapCallData` in `swapTo` function	HIGH	ACKNOWLEDGED
9	Issue when result of `swapProxy` not equal the actual used value	MEDIUM	FIXED
10	Confuse validate logic in `completeWithdrawal` function and user pay more fee.	MEDIUM	FIXED
11	Issue when withdraw non-standard erc20 token in `emergencyShutdown` function.	MEDIUM	FIXED
12	Arbitrary pass `timestamp` in the `acquireManagementFee` function	MEDIUM	ACKNOWLEDGED
13	Centralization issue in `importVaultState` function	MEDIUM	FIXED
14	Add `_gap` Variable to Upgradable Contract	LOW	ACKNOWLEDGED
15	Using `block.chaindid` instead of passing networkdId in initialize function	LOW	ACKNOWLEDGED
16	Unused `withdrawals[msg.sender].profit` state	INFORMATIVE	FIXED
17	Unclear logic in `getUserVaultState` function	INFORMATIVE	FIXED

2.2.1. CRITICAL - Wrong formula in `syncRestakingBalance` function

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/strategies/BaseRestakingStrategy.sol

The formula in the `syncRestakingBalance` function is incorrect. The `restakingState.totalBalance` is value of USDC converted from all assets in the vault. But the formula is use the restakingToken value.

```
function closePosition(uint256 ethAmount) external nonReentrant {
    //...
    restakingState.unAllocatedBalance += actualUsdcAmount; // @VerichainsAuditor:
unAllocatedBalance stored USDC value
    //...
    }
    function syncRestakingBalance() external nonReentrant {
        uint256 ethAmount = restakingToken.balanceOf(address(this)) *
swapProxy.getPriceOf(address(restakingToken), address(ethToken)) / 1e18;
        restakingState.totalBalance = restakingState.unAllocatedBalance + ethAmount *
swapProxy.getPriceOf(address(restakingToken), address(ethToken)) / 1e18; //
@VerichainsAuditor: `ethAmount * swapProxy.getPriceOf(address(restakingToken),
address(ethToken)) / 1e18` is the value in `restakingToken` value not the USDC value like
`restakingState.unAllocatedBalance`
    }
```

UPDATES

- ***Aug 29, 2024***: The issue has been fixed in the commit `0c76341d553b7ce48b012da7bc4792ddbd3bb61d`.

RECOMMENDATION

Change value from restakingToken to USDC value in the formula.

2.2.2. CRITICAL - Unclear logic in calculate `totalShareAmount` in `initiateWithdrawal` function

Positions:

- `contracts/vaults/restakingDeltaNeutral/Base/BaseDeltaNeutralVault.sol`

Wrong formula in `totalShareAmount`, it will cause the issue when the vault.param.decimals is not 6. The `totalShareAmount` should be `ShareMath.sharesToAsset(depositReceipt.shares, pps, vaultParams.decimals)`;

```
function initiateWithdrawal(uint256 shares) external nonReentrant {
        DepositReceipt storage depositReceipt = depositReceipts[msg.sender];
        require(depositReceipt.shares >= shares, "INVALID_SHARES");
        require(withdrawals[msg.sender].shares == 0, "INVALID_WD_STATE");
        uint256 pps = _getPricePerShare();
        uint256 totalShareAmount = depositReceipt.shares * pps / 1e6; //
@VerichainsAuditor: Wrong formula here, error when vault.param.decimals is not 6
        //...
        withdrawals[msg.sender].withdrawAmount = ShareMath.sharesToAsset(shares, pps,
vaultParams.decimals);
        //...
    }
```

RECOMMENDATION

The `currentAmount` in `getUserVaultState` function should be updated with the correct formula.

UPDATES

- ***Aug 29, 2024***: The issue has been fixed in the commit `0c76341d553b7ce48b012da7bc4792ddbd3bb61d`.

2.2.3. CRITICAL - Missing mul `performancefee` decimals in `initiateWithdrawal` function.

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/BaseDeltaNeutralVault.sol

When initiate withdrawal, the performanceFee is calculated with the wrong formula. The `performanceFee` should be multiplied by `1e12` to represent the decimals.

```
function initiateWithdrawal(uint256 shares) external nonReentrant {
        ...
        uint256 performanceFee = withdrawProfit > 0 ? (withdrawProfit *
vaultParams.performanceFeeRate) / 1e14 : 0;// @VerichainsAuditor: Missing mul
performancefee decimals
        ...

    function setFeeRates(
        uint256 _performanceFeeRate,
        uint256 _managementFeeRate
    ) external {
    _auth(ROCK_ONYX_ADMIN_ROLE);

    require(_performanceFeeRate <= 100, "INVALI_RATE");
    require(_managementFeeRate <= 100, "INVALID_RATE");
    vaultParams.performanceFeeRate = _performanceFeeRate; // @VerichainsAuditor: the
performanceFeeRate is not contain the decimals
    vaultParams.managementFeeRate = _managementFeeRate;
    emit FeeRatesUpdated(_performanceFeeRate, _managementFeeRate);

    function _getManagementFee(uint256 timestamp) internal view returns (uint256) {
        uint256 perSecondRate = vaultParams.managementFeeRate * 1e12 / (365 * 86400) + 1;
        // @VerichainsAuditor: the managementFeeRate has the same logic with
performanceFeeRate, and it mul with 1e12 for represent the base decimals
        uint256 period = timestamp - vaultState.lastUpdateManagementFeeDate;
        return ((_totalValueLocked() - vaultState.withdrawPoolAmount) * perSecondRate *
period) / 1e14;
    }
```

RECOMMENDATION

The `performanceFeeRate` should be multiplied by `1e12` to represent the decimals.

UPDATES

- ***Aug 29, 2024***: The issue has been fixed in the commit `0c76341d553b7ce48b012da7bc4792ddbd3bb61d`.

2.2.4. HIGH - Looping through the array in `updateDepositArr` and `updateWithdrawalArr` causes the gas limit to be exceeded

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/BaseDeltaNeutralVault.sol

The `updateDepositArr` and `updateWithdrawalArr` functions loop through the array of deposit and withdrawal receipts to update the state of the contract. As the length of these arrays grows over time, the gas consumption of the loop can exceed the block gas limit, causing the functions to fail. This can lead to several functions failing to execute when uses these function.

```
function updateDepositArr(DepositReceipt memory depositReceipt) internal {
    for (uint256 i = 0; i < depositReceiptArr.length; i++) {
        if (depositReceiptArr[i].owner == msg.sender) {
            depositReceiptArr[i].depositReceipt = depositReceipt;
            return;
        }
    }
    depositReceiptArr.push(DepositReceiptArr(msg.sender, depositReceipt));
    }
    function updateWithdrawalArr(Withdrawal memory withdrawal) internal {
        for (uint256 i = 0; i < withdrawalArr.length; i++) {
            if (withdrawalArr[i].owner == msg.sender) {
                withdrawalArr[i].withdrawal = withdrawal;
                return;
            }
        }
        withdrawalArr.push(WithdrawalArr(msg.sender, withdrawal));
    }
```

RECOMMENDATION

Redesign the contract to avoid looping through the array. Instead, use a mapping to store the deposit and withdrawal receipts.

UPDATES

- ***Aug 29, 2024***: The issue has been acknowledged by the development team.

2.2.5. HIGH - Not refund excess amount in `swapToWithOutput` function

Positions:

- contracts/extensions/Uniswap/Uniswap.sol

When running in BASE_NETWORK, the `swapToWithOutput` function does not refund the excess amount to the user. This can lead to a loss of funds for the user.

```
function swapToWithOutput(
    address recipient,
    address tokenIn,
    uint256 amountOut,
    address tokenOut,
    uint24 poolFee
) external returns (uint256) {
    //...

    if(network == BASE_NETWORK){
    IUniSwapRouterOnBase.ExactOutputSingleParams memory wdParams =
IUniSwapRouterOnBase.ExactOutputSingleParams({
        tokenIn: tokenIn,
        tokenOut: tokenOut,
        fee: poolFee,
        recipient: recipient,
        amountOut: amountOut,
        amountInMaximum: amountInMaximum,
        sqrtPriceLimitX96: 0
    });
    return swapRouterOnBase.exactOutputSingle(wdParams); // @VerichainsAuditor: The excess
amount is not refund to the user
    }
    IUniSwapRouter.ExactOutputSingleParams memory params =
IUniSwapRouter.ExactOutputSingleParams({
        tokenIn: tokenIn,
        tokenOut: tokenOut,
        fee: poolFee,
        recipient: recipient,
        deadline: block.timestamp,
        amountOut: amountOut,
        amountInMaximum: amountInMaximum,
        sqrtPriceLimitX96: 0
    });

    uint256 amountIn = swapRouter.exactOutputSingle(params);
    if (amountIn < amountInMaximum) {
    TransferHelper.safeApprove(tokenIn, address(swapRouter), 0);
    TransferHelper.safeTransfer(tokenIn, msg.sender, amountInMaximum - amountIn);
    }
return amountIn;
}
```

RECOMMENDATION

The `swapToWithOutput` function should refund the excess amount to the user to prevent loss of funds.

UPDATES

- ***Aug 29, 2024***: The issue has been fixed in the commit `0c76341d553b7ce48b012da7bc4792ddbd3bb61d`.

2.2.6. HIGH - Issue when `claimFee` too much totalFeePoolAmount

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/BaseDeltaNeutralVault.sol

Within the if statement, `vaultState.totalFeePoolAmount` is set to 0 before the transfer occurs. Consequently, the subsequent safeTransfer statement using this value will always transfer 0 tokens.

```
function claimFee(address receiver, uint256 amount) external nonReentrant {
        _auth(ROCK_ONYX_ADMIN_ROLE);

        if (amount > vaultState.totalFeePoolAmount) {
            vaultState.totalFeePoolAmount = 0;
            TransferHelper.safeTransfer(vaultParams.asset, receiver,
vaultState.totalFeePoolAmount); // @VerichainsAuitor: It will transfer 0 token.
            return;
        }

        vaultState.totalFeePoolAmount -= amount;
        TransferHelper.safeTransfer(vaultParams.asset, receiver, amount);
    }
```

2.2.7. HIGH - Centralization issue in `syncPerpDexBalance` function

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/strategies/PerpDexStrategy.sol

Description:

The `syncPerpDexBalance` function allows the admin to change the `perpDexState` state variable. With this logic, the profit and loss of the project can be manipulated by the admin.

```
function syncPerpDexBalance(uint256 totalBalance, uint256 totalProfit) external {
        _auth(ROCK_ONYX_ADMIN_ROLE);
        perpDexState.totalBalance = totalBalance;
        perpDexState.totalProfit = totalProfit;
    }
```

UPDATES

- ***Aug 29, 2024***: The issue has been acknowledged by the development team.

2.2.8. HIGH - User-Controlled `swapCallData` in `swapTo` Function

Positions:

- `contracts/lib/BaseSwapAggregator.sol`#swapTo()

The `swapTo` function presents a significant vulnerability by allowing users to directly control the `swapCallData` without any validation, which is executed through a low-level `.call` to the `exchangeAddress`. Since users can pass any arbitrary data as `swapCallData`, this opens up the potential for executing unintended or malicious operations on the `exchangeAddress`. If the `exchangeAddress` is compromised or intentionally malicious, it could result in the unauthorized transfer of tokens or other harmful actions within the contract.

RECOMMENDATION

It is recommended to implement a validation mechanism to ensure that the `swapCallData` is safe and does not contain any malicious or unintended operations. This can be achieved by implementing a whitelist of approved `exchangeAddresses` or by restricting the types of operations that can be executed through the `swapCallData`.

UPDATES

- ***Aug 29, 2024***: The issue has been acknowledged by the development team.

2.2.9. MEDIUM - Issue when result of `swapProxy` not equal the actual used value

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/strategies/BaseRestakingStrategy.sol

The `openPosition` and `closePosition` functions in the contract directly use the return value from the `swapProxy` contract to update the state of USDC used/received. This approach is vulnerable to potential inconsistencies if the `swapProxy` contract's logic is modified or the returned value is influenced by custom tokens. In such cases, the returned value may not accurately reflect the actual USDC amount used or received, leading to incorrect state updates.

```
function openPosition(uint256 ethAmount) external nonReentrant {
        _auth(ROCK_ONYX_OPTIONS_TRADER_ROLE);
        require(restakingState.unAllocatedBalance > 0, "INSUFICIENT_BALANCE");

        usdcToken.approve(address(swapProxy), restakingState.unAllocatedBalance);
        uint256 usedUsdAmount = swapProxy.swapToWithOutput(
            address(this),
            address(usdcToken),
            ethAmount,
            address(ethToken),
            getFee(address(usdcToken), address(ethToken))
        ); // @VerichainsAuditor: `usedUsdAmount` is the return value from `swapProxy` may
not the actual swap value.

        restakingState.unAllocatedBalance -= usedUsdAmount;
        depositToRestakingProxy(ethAmount);

        emit PositionOpened(usedUsdAmount, ethAmount);
}
```

RECOMMENDATION

The function should store the balance before and after the swap, and then calculate the difference to update the state.

UPDATES

- ***Aug 29, 2024***: The issue has been fixed in the commit `0c76341d553b7ce48b012da7bc4792ddbd3bb61d`.

2.2.10. MEDIUM - Confuse validate logic in `completeWithdrawal` function and user pay more fee.

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/BaseDeltaNeutralVault.sol

In `completeWithdrawal` function, the logic in `require` statement and the next statement (ternary statement) make the logic confusion.

With the current constraint, when the `withdrawalPoolAmount > vault.withdrawPoolAmount` and `withdrawlPoolAmount < vault.withdrawPoolAmount + feeAmount`. User only receive the `vaultState.withdrawPoolAmount` but the fee user must pay is based on the `withdrawalPoolAmount`(the bigger one). As result, user must pay fee more than the fee they should pay.

```
function completeWithdrawal(uint256 shares) external nonReentrant {
        require(withdrawals[msg.sender].shares >= shares, "INVALID_SHARES");
        uint256 withdrawAmount = (shares * withdrawals[msg.sender].withdrawAmount) /
withdrawals[msg.sender].shares;
        uint256 performanceFee = (shares * withdrawals[msg.sender].performanceFee) /
withdrawals[msg.sender].shares;
        uint256 feeAmount = performanceFee + networkCost;
        vaultState.totalFeePoolAmount += feeAmount;

        require( vaultState.withdrawPoolAmount > withdrawAmount - feeAmount,
"EXCEED_WD_POOL_CAP"); // @VerichainsAuditor: Unclear logic here
        withdrawAmount = vaultState.withdrawPoolAmount < withdrawAmount ?
vaultState.withdrawPoolAmount : withdrawAmount; // @VerichainsAuditor: Unclear logic here
        vaultState.withdrawPoolAmount -= withdrawAmount;
        // end migration
    }
```

RECOMMENDATION

The require statement should `require(vaultState.withdrawPoolAmount >= withdrawAmount)` and the ternary statement should be removed.

UPDATES

- ***Aug 29, 2024***: The issue has been fixed in the commit `0c76341d553b7ce48b012da7bc4792ddbd3bb61d`.

2.2.11. MEDIUM - Issue when withdraw non-standard erc20 token in `emergencyShutdown` function.

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/BaseDeltaNeutralVault.sol

In the `emergencyShutdown` function, using `IERC20(tokenAddress).transfer(receiver, amount)` could cause a revert if the token at tokenAddress is not a standard ERC20 token. To prevent this, it's recommended to use IERC20(tokenAddress).safeTransfer(receiver, amount) instead.

```
function emergencyShutdown(
        address receiver,
        address tokenAddress,
        uint256 amount
    ) external nonReentrant {
        _auth(ROCK_ONYX_ADMIN_ROLE);

        IERC20(tokenAddress).transfer(receiver, amount);
    }
```

UPDATES

- ***Aug 29, 2024***: The issue has been fixed in the commit `0c76341d553b7ce48b012da7bc4792ddbd3bb61d`.

2.2.12. MEDIUM - Arbitrary pass `timestamp` in the `acquireManagementFee` function

Positions:

- `contracts/vaults/restakingDeltaNeutral/KelpZircuit/KelpRestakingDeltaNeutralVault.sol`#acquireManagementFee()

Description:

The admin position has the ability to call the `acquireManagementFee` method with any timestamp. Furthermore, the fees amount is determined using the delta between the given timestamp and the latest management fee timestamp. The admin can supply an arbitrary timestamp to the function and collect the management fee for that date. This could result in a loss of funds for users.

```
function acquireManagementFee(uint256 timestamp) external nonReentrant {
        _auth(ROCK_ONYX_ADMIN_ROLE);

        uint256 feeAmount = _getManagementFee(timestamp);
        require(feeAmount <= _totalValueLocked(), "INVALID_ACQUIRE_AMOUNT");

        vaultState.totalFeePoolAmount += _acquireFunds(feeAmount);
        vaultState.lastUpdateManagementFeeDate = block.timestamp;
    }
```

Recommendation:

The fee amount should be calculated based on the current timestamp. The admin should not be able to pass an arbitrary timestamp to the function.

UPDATES

- ***Aug 29, 2024***: The issue has been acknowledged by the development team.

2.2.13. MEDIUM - Centralization issue in `importVaultState` function

Positions:

- `contracts/vaults/restakingDeltaNeutral/KelpZircuit/KelpRestakingDeltaNeutralVault.sol`#importVaultState()

Description:

The `importVaultState` function allows the admin to change all state variables of the contract. This function is only accessible by the admin. This function can be used to manipulate the state of the contract and can cause a change balance of users.

```
function importVaultState(
        DepositReceiptArr[] calldata _depositReceiptArr,
        WithdrawalArr[] calldata _withdrawalArr,
        VaultParams calldata _vaultParams,
        VaultState calldata _vaultState,
        EthRestakingState calldata _ethRestakingState,
        PerpDexState calldata _perpDexState
    ) external {
        _auth(ROCK_ONYX_ADMIN_ROLE);

        depositReceiptArr = _depositReceiptArr;
        for (uint256 i = 0; i < _depositReceiptArr.length; i++) {
            depositReceipts[_depositReceiptArr[i].owner] = _depositReceiptArr[i]
                .depositReceipt;
        }

        withdrawalArr = _withdrawalArr;
        for (uint256 i = 0; i < _withdrawalArr.length; i++) {
            withdrawals[_withdrawalArr[i].owner] = _withdrawalArr[i].withdrawal;
        }

        vaultParams = _vaultParams;
        vaultState = _vaultState;
        restakingState = _ethRestakingState;
        perpDexState = _perpDexState;
    }
```

Recommendation:

For compliance with the decentralization technique, delete the importVaultState function from the contract.

UPDATES

- ***Aug 29, 2024***: The issue has been acknowledged by the development team.

2.2.14. LOW - Add _gap Variable to Upgradable Contract

Positions:

- RockOnyxAccessControl.sol
- contracts/vaults/restakingDeltaNeutral/Base/BaseSwapVault.sol

Storage gaps are a convention for reserving storage slots in a base contract, allowing future versions of that contract to use up those slots without affecting the storage layout of child contracts.

RECOMMENDATION

We recommend adding _gap variable to the contract or use AccessControlUpgradeable, it will be helpful for the future developments. Following the guide by the Openzeppelin: *https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps*

UPDATES

- ***Aug 29, 2024***: The issue has been acknowledged by the development team.

2.2.15. LOW - Using block.chaindid instead of passing networkdId in initialize function

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/strategies/BaseRestakingStrategy.sol

The current practice of passing networkId in the initialize function of the BaseRestakingStrategy contract is susceptible to human error.

We recommend using block.chainid instead of passing networkId in the initialize function. It will help to avoid the human error in the future.

UPDATES

- ***Aug 29, 2024***: The issue has been acknowledged by the development team.

2.2.16. INFORMATIVE - Unused `withdrawals[msg.sender].profit` state.

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/BaseDeltaNeutralVault.sol

The `initializeWithdrawal` function in the contract sets the `withdrawals[msg.sender].profit` state variable, but this variable is not referenced or used anywhere else within the contract's code.

RECOMMENDATION

The `withdrawals[msg.sender].profit` state should be removed from the contract.

UPDATES

- ***Aug 29, 2024***: The issue has been fixed in the commit `0c76341d553b7ce48b012da7bc4792ddbd3bb61d`.

2.2.17. INFORMATIVE - Unclear logic in `getUserVaultState` function.

Positions:

- contracts/vaults/restakingDeltaNeutral/Base/BaseDeltaNeutralVault.sol

The variable currently named `profit` within the function can be misleading, as the formula suggests it calculates the percentage return on investment (ROI) rather than the absolute profit amount. To improve clarity, it should be renamed to `ROI` or `profitPercentage`. The same principle applies to the `loss` variable, which should be renamed to `lossPercentage`.

UPDATES

- ***Aug 29, 2024***: The issue has been fixed in the commit `0c76341d553b7ce48b012da7bc4792ddbd3bb61d`.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Aug 29, 2024*	Public Report	Verichains Lab

Table 2. Report versions history