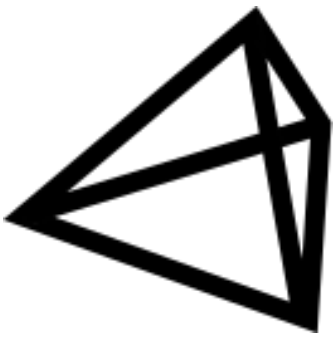


Derivio

Audit

Presented by:

OtterSec contact@osec.io

Woosun Song procfs@osec.io

Nicholas R. Putra nicholas@osec.io

Contents

© 2023 Otter Audits LLC. All Rights Reserved.

01 | Executive Summary

Overview

Derivio engaged OtterSec to perform an assessment of the `derivio-core` program. This assessment was conducted between May 24th and June 14th, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 14 findings in total.

In particular, we identified issues regarding reentrancy (OS-DRV-ADV-00), depletion of ether due to missing funds validation (OS-DRV-ADV-01), and an incomplete order cancel implementation (OS-DRV-ADV-02). In addition, we addressed implementation errors that render multiple core functionalities unusable (OS-DRV-ADV-03, OS-DRV-ADV-05, OS-DRV-ADV-06, OS-DRV-ADV-07).

We also made recommendations to address an approval to an unchecked address (OS-DRV-SUG-00), an out-of-order business logic (OS-DRV-SUG-01), around gas optimization (OS-DRV-SUG-02, OS-DRV-SUG-03, OS-DRV-SUG-04), and code maturity (OS-DRV-SUG-05).

© 2023 Otter Audits LLC. All Rights Reserved.

02 | Scope

The source code was delivered to us in a git repository at github.com/raindexv/derivio-core. The initial audit was performed on b8ec035.

During the audit process, Derivio implemented improvements for intermediate findings in a454f15. In addition to the initial audit, we have conducted supplementary audits for this version.

A brief description of the programs is as follows.

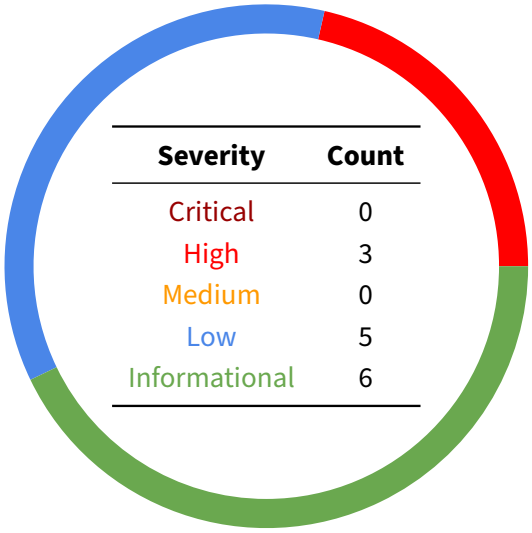
Name	Description
manager	The manager contract provides an interface for end-users and makers to configure and interact with pools. It implements the following functionalities: • Pyth price feed updates and retrieval • Order placement, execution, and cancellation • Option size increase and reduction
pool	Pools implement core business logic for deposits, exchanges, and derivatives. It implements the following functionalities: • Deposits: Users deposit funds into the pool in return for pool tokens, which they may stake to mine rewards. • Exchanges: Users may convert between assets, where oracle prices determine the exchange ratio. • Derivatives: Pools offer options and perpetual exchanges. Both products heavily rely on oracle prices to determine position states.
staking	The staking contract distributes rewards proportional to the amount and duration of the stake. Users may deposit stake share tokens into pools for exchanges and derivatives.

© 2023 Otter Audits LLC. All Rights Reserved.

03 | Findings

Overall, we reported 14 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	3
Medium	0
Low	5
Informational	6

© 2023 Otter Audits LLC. All Rights Reserved.

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-DRV-ADV-00	High	Resolved	A compromised maker may appropriate funds in the manager through reentrancy in `executeOrder`.
OS-DRV-ADV-01	High	Resolved	The manager contract naively presumes the provision of ether in `payable`, `updatePrices`, and `create*Order`. When exploited, this may deplete ether.
OS-DRV-ADV-02	High	Resolved	`cancelOrder` does not return collateral and execution fees provided in order creation, resulting in permanent loss of funds.
OS-DRV-ADV-03	Low	Resolved	`_updateERC20Balance` improperly handles tokens with `address(0)` by using `IERC20(address).balanceOf` for balance tracking instead of the native `balance`, which causes execution to revert unconditionally.
OS-DRV-ADV-04	Low	Resolved	`transferOut` and `transferOutUnwrapped` violate the Check-Effects-Interactions pattern which makes them potentially susceptible to reentrancy.
OS-DRV-ADV-05	Low	Resolved	`createPoolSwapOrder` marks the order type as `POOl_DEPOSIT` instead of `POOL_SWAP`.
OS-DRV-ADV-06	Low	Resolved	`getOracleAssets` exhibits an out-of-bounds read, resulting in an unconditional revert.
OS-DRV-ADV-07	Low	Resolved	`createClaimDepositOrder` does not have `payable` keyword.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-ADV-00 [high] | Reentrancy Due To A Low-Level Call

Description

The `executeOrder` function performs execution fee compensation by transferring ether through a low-level call. The subsequent storage updates perform in a manner that resembles a direct violation of the Check-Effects-Interactions pattern. This violation leaves the facet vulnerable to reentrancy.

contracts/manager/facets/order/ExecuteOrderFacet.sol SOLIDITY

```
 function executeOrder(
    bytes32 key,
    uint256 timestamp,
    OracleUpdates calldata oracleUpdates
) external {
    require(_hasRole(LibAccessControl.ORDER_KEEPER_ROLE, msg.sender));

    timestamp = LibTimestamp.processTimestamp(timestamp);

    OrderState storage orderState = LibStorage.orderState();

    Order memory order = orderState.orders[key];

    // require at least some seconds passed
    require(timestamp >= order.timestamp + 1, "not yet executable");

    // Compensate execution fee (unwrap ETH)
    LibManagerTransfer.transferOut({
        token: LibStorage.globalConfig().weth,
        amount: order.executionFee,
        recipient: msg.sender
    });

    if (order.orderType == OrderType.PERP_INCREASE) {
        LibPerpIncreaseOrder.executePerpIncreaseOrder(
            key,
            timestamp,
            oracleUpdates
        );
    } else if (order.orderType == OrderType.PERP_DECREASE) {
        /* other branch cases */
    }
}
```

Exploiting this vulnerability may only happen under the assumption of a compromised keeper. Thus, we consider this issue to be a centralization risk rather than a theft of funds.

Remediation

Perform execution fee compensation after all storage updates are performed.

© 2023 Otter Audits LLC. All Rights Reserved.

Patch

Fixed in 8252097.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-ADV-01 [high] | Missing Funds Validation

Description

The manager contract fails to validate the provision of ether. An adversary may exploit this to deplete ether. When creating orders, users must pay execution fees, which cover Pyth price feed update fees. However, the protocol fails to verify two critical conditions.

First, it does not ensure that the amount of execution fees paid is adequate to cover the Pyth fees. Second, it neglects to verify whether `msg.value` matches the execution fees specified in the order. Due to the absence of these checks, a malicious actor may exploit the system by repeatedly invoking fee-taking operations without making the necessary payments, resulting in a depletion of ether from the manager contract.

One example of such an attack is to invoke `updatePrices` repeatedly to drain ether. `updatePrices` is public and does not perform any access control checks. However, it sends ether to the Pyth contract without checking if the caller provided sufficient funds.

contracts/manager/facets/oracle/OracleFacet.sol SOLIDITY

```
function updatePrices(
    OracleUpdates calldata updates,
    address[] calldata tokens,
    address base,
    address quote
) external payable returns (PriceDataList memory priceList) {
    PriceData[] memory pythPrices = updatePythPrices(
        updates.pythUpdateData,
        tokens
    );
    return newPriceDataList({prices: pythPrices, base: base, quote: quote});
}
```

Remediation

Verify two essential conditions in the protocol: firstly, that the execution fees are greater than or equal to the Pyth fees, and secondly, that successful payment of the execution fees. Additionally, uphold the invariant that the manager contract holds no ether.

Patch

Fixed in 870521d by checking the balances before and after the order process, validating the provision of funds, rather than checking `msg.value`.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-ADV-02 [high] | Incomplete Cancel Order Implementation

Description

`cancelOrder` only cleans up relevant storage variables without returning collateral and execution fees paid by the order's creator.

contracts/manager/facets/order/ExecuteOrderFacet.sol SOLIDITY

```
    function cancelOrder(bytes32 key) external {
        OrderState storage orderState = LibStorage.orderState();
        Order memory order = orderState.orders[key];

        require(order.owner == msg.sender, "not owner");
        require(order.orderType != OrderType.INVALID, "not active");

        if (order.orderType == OrderType.PERP_INCREASE) {
            LibPerpIncreaseOrder.cleanupPerpIncreaseOrder(key);
        } else if (order.orderType == OrderType.PERP_DECREASE) {
            LibPerpDecreaseOrder.cleanupPerpDecreaseOrder(key);
        } else if (order.orderType == OrderType.POOL_DEPOSIT) {
            LibPoolDepositOrder.cleanupPoolDepositOrder(key);
        } else if (order.orderType == OrderType.POOL_REDEEM) {
            LibPoolRedeemOrder.cleanupPoolRedeemOrder(key);
        } else if (order.orderType == OrderType.POOL_SWAP) {
            LibPoolSwapOrder.cleanupPoolSwapOrder(key);
        } else if (order.orderType == OrderType.OPTION_OPEN) {
            LibOptionOpenOrder.cleanupOptionOpenOrder(key);
        } else {
            revert("unsupported order type");
        }

        orderState.orderKeys.remove(key);
        delete orderState.orders[key];

        emit OrderCancelled(key);
    }
```

Remediation

Extend the functionality of `cancelOrder` so it returns collateral and execution fees.

Patch

Fixed in 274f9b0.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-ADV-03 [low] | Incorrect Handling Of Balance Mappings

Description

`_updateERC20Balance` contains an implementation error that overlooks a specific corner case involving ether as the asset. In this case, when the `token` address is zero, the function calls `IERC20(address).balanceOf(address(this))`, which will revert unconditionally due to calling a zero address. The correct implementation should use `address(this).balance` to capture the ether balance correctly.

contracts/common/lib/LibTransfer.sol SOLIDITY

```
    function _updateERC20Balance(address token) private {
        state().tokenBalance[token] = IERC20(token).balanceOf(address(this));
    }
```

Remediation

Extend `_updateERC20Balance` to handle `address(0)` correctly.

Patch

Fixed in b69afbb.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-ADV-04 [low] | Reentrancy Due to ERC20 Extensions

Description

`transferOut` and `transferOutUnwrapped` do not follow the Check-Effects-Interactions (CEI) pattern, which may potentially result in reentrancy. Reentrancy may occur if the specified `token` extends the functionalities of ERC-20, such as ERC-777. ERC-777 allows callbacks to be invoked upon token transfers, which becomes a source of interleaving, as in low-level calls. Currently, this reentrancy may not be exploited in favor of the attacker, but we recommend addressing this issue in case of future code additions.

contracts/common/lib/LibTransfer.sol SOLIDITY

```
    function transferOut(
        address token,
        address recipient,
        uint256 amount
    ) internal {
        IERC20(token).safeTransfer(recipient, amount);
        _updateERC20Balance(address(0));
    }
```

contracts/common/lib/LibTransfer.sol SOLIDITY

```
    function transferOutUnwrapped(
        address token,
        address recipient,
        uint256 amount,
        address weth
    ) internal {
        if (token == weth) {
            IWETH(token).withdraw(amount);
            _updateERC20Balance(token);

            // No need to update ETH balance - same amount unwrapped as the amount
    ↪  sent

            (bool success, ) = recipient.call{ value: amount }("");
            require(success, "ETH transfer failed");
        } else {
            IERC20(token).safeTransfer(recipient, amount);
            _updateERC20Balance(token);
        }
    }
```

Remediation

There are several options available to address reentrant code. One approach is to impose restrictions on the types of tokens whitelisted by the protocol, thereby disallowing potentially hazardous tokens like ERC777.

© 2023 Otter Audits LLC. All Rights Reserved.

Additionally, modifying the code to strictly adhere to the CEI pattern can be effective. For example, in the `transferOut` function, the `_updateERC20Balance` should be called before `safeTransfer`.

If considering the inclusion of riskier tokens, it is worth considering more robust reentrancy protection measures. One viable option is the introduction of reentrancy guards. However, it is important to note that reentrancy guards can lead to increased gas consumption. Fortunately, the forthcoming introduction of EIP-1153(Transient Storage Operations) is expected to alleviate this concern.

Patch

Fixed in b69afbb.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-ADV-05 [low] | Implementation Error In Swap Order Creation

Description

`createPoolSwapOrder` exhibits an implementation error where the posted `orderType` is `POOL_DEPOSIT` instead of `POOL_SWAP`, reverting all swap orders upon execution.

contracts/manager/lib/order/LibPoolSwapOrder.sol SOLIDITY

```
    /// @notice Create a pool deposit order
    function createPoolSwapOrder(
        CreatePoolSwapOrder memory params
    ) internal returns (bytes32) {
    ...
        LibStorage.orderState().orders[key] = Order({
            orderType: OrderType.POOL_DEPOSIT,
            owner: msg.sender,
            timestamp: LibTimestamp.blockTimestamp(),
            executionFee: params.executionFee
        });
    ...
    }
```

Remediation

Change `orderType` to `OrderType.POOL_SWAP` as shown below:

contracts/manager/lib/order/LibPoolSwapOrder.sol SOLIDITY

```
    /// @notice Create a pool deposit order
    function createPoolSwapOrder(
        CreatePoolSwapOrder memory params
    ) internal returns (bytes32) {
    ...
        LibStorage.orderState().orders[key] = Order({
            orderType: OrderType.POOL_SWAP,
            owner: msg.sender,
            timestamp: LibTimestamp.blockTimestamp(),
            executionFee: params.executionFee
        });
    ...
    }
```

Patch

Fixed in ae817c2.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-ADV-06 [low] | Out Of Bound Read

Description

`getOracleAssets` exhibits an out-of-bounds read on the `collateralTokens` array.

contracts/manager/facets/option/OptionGlobalFacet.sol SOLIDITY

```
    /// @notice Get array of all collateral tokens + indexAsset
    function getOracleAssets(
        address indexAsset
    ) internal view returns (address[] memory) {
        EnumerableSet.AddressSet storage collateralTokens = LibStorage
            .optionsGlobalConfig()
            .collateralTokens;
        uint256 length = 1 + collateralTokens.length();
        address[] memory assets = new address[](length);
        for (uint256 i = 0; i < length; i++) {
            assets[i] = collateralTokens.at(i);
        }
        assets[length - 1] = indexAsset;
        return assets;
    }
```

The `collateralTokens` array will be indexed from 0 to `collateralTokens.length`, with both ends included. The last iteration will cause an unconditional revert as the read is out of bounds by one.

Remediation

Remove the last iteration step as follows:

contracts/manager/facets/option/OptionGlobalFacet.sol SOLIDITY

```
    /// @notice Get array of all collateral tokens + indexAsset
    function getOracleAssets(
        address indexAsset
    ) internal view returns (address[] memory) {
        EnumerableSet.AddressSet storage collateralTokens = LibStorage
            .optionsGlobalConfig()
            .collateralTokens;
        uint256 length = collateralTokens.length();
        address[] memory assets = new address[](length + 1);
        for (uint256 i = 0; i < length; i++) {
            assets[i] = collateralTokens.at(i);
        }
        assets[length] = indexAsset;
        return assets;
    }
```

© 2023 Otter Audits LLC. All Rights Reserved.

Patch

Fixed in 5d5cc46.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-ADV-07 [low] | Non-Payable Callee Of Payable Function

Description

`createClaimDepositOrder` is non-payable, but it calls `createPoolDepositOrder` which is payable. This implementation error prevents users from sending the required execution fees and renders the `createClaimDepositOrder` function unusable.

contracts/staking/facets/StakingRewardsFacet.sol SOLIDITY

```
    /// @notice Claim sender's reward, then create deposit & stake order for
    ↪  receiver
    function createClaimDepositOrder(
        CreateClaimDepositOrder memory params
    ) public {
    ...
    }
```

Remediation

Make `createClaimDepositOrder` payable.

contracts/staking/facets/StakingRewardsFacet.sol SOLIDITY

```
    /// @notice Claim sender's reward, then create deposit & stake order for
    ↪  receiver
    function createClaimDepositOrder(
        CreateClaimDepositOrder memory params
    ) public payable {
    ...
    }
```

Patch

Fixed in 4e942ab.

© 2023 Otter Audits LLC. All Rights Reserved.

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may lead to security issues in the future.

ID	Description
OS-DRV-SUG-00	`createClaimDepositOrder` does not validate the inputted `param.router` address.
OS-DRV-SUG-01	`_previewSwap` performs balance checks on the input asset before increasing it.
OS-DRV-SUG-02	`checkTokensForOptions` has unused parameters.
OS-DRV-SUG-03	`previewMultiSwap`'s `tokenPath` parameter may be optimized to be of type `calldata`.
OS-DRV-SUG-04	`_previewSwap` contains a redundant asset whitelist check.
OS-DRV-SUG-05	Contracts mandate the invocation of `recordTransferOutETH` after transferring ether, which is prone to human error.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-SUG-00 | Missing Address Checks

Description

The `createClaimDepositOrder` function does not validate the user-provided `param.router` address. This oversight results in the approval of `rewardToken` on an arbitrary address. The amount of clearance is limited to the amount of rewards claimable by `msg.sender`, resulting in this issue having minimal security implications. However, we recommend implementing address checks in case of future code additions.

contracts/staking/facets/StakingRewardsFacet.sol SOLIDITY

```
    /// @notice Claim sender's reward, then create deposit & stake order for
    ↪  receiver
    function createClaimDepositOrder(
        CreateClaimDepositOrder memory params
    ) public {
        uint256 rewardAmount = claim(address(this));

        StakingRewardsConfig storage config = LibStorage.stakingRewardsConfig();
        address token = config.assetToken;
        address rewardToken = config.rewardToken;

        IERC20(rewardToken).safeIncreaseAllowance(params.router, rewardAmount);
        ICreateOrder(params.router).createPoolDepositOrder(
            CreatePoolDepositOrder({
                executionFee: params.executionFee,
                amountIn: rewardAmount,
                inToken: rewardToken,
                swapPoolPath: new address[](0),
                swapTokenPath: new address[](0),
                minSwapOut: 0,
                pool: token,
                asset: rewardToken,
                receiver: params.receiver,
                minShares: params.minShares,
                stake: params.stake
            })
        );
    }
```

Remediation

Implement validation for `param.router`.

Patch

Fixed in 4e942ab.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-SUG-01 | Out Of Order Validation Logic

Description

`_previewSwap` exhibits an implementation error where `fromAssetAmount` is validated before it is updated. All swaps whose `fromAssetAmount` is larger than the initial balance of the `fromAsset`'s pool balance will be reverted upon execution even though the `pool` has enough `toAsset`'s balance for the needed `toAssetAmount`.

contracts/pool/facets/pool/PoolSwapFacet.sol SOLIDITY

```
    /// @notice Returns the amount of `toAsset` that would be received (after fees)
    /// @param whitelistPrices base must be `fromAsset`, quote must be `toAsset`
    function _previewSwap(
        address fromAsset,
        address toAsset,
        uint256 fromAssetAmount,
        PriceDataList memory whitelistPrices
    )
        internal
        view
        returns (uint256 toAssetAmount_, uint256 feeToAssetAmount_)
    {
        if (!poolConfig().assetWhitelist.contains(fromAsset)) {
            revert AssetNotWhitelisted();
        }
        if (!poolConfig().assetWhitelist.contains(toAsset)) {
            revert AssetNotWhitelisted();
        }
        if (fromAsset == toAsset) {
            revert("fromAsset == toAsset");
        }

        require(
            poolState().assetBalances[fromAsset] >= fromAssetAmount,
            "insufficient fromAsset"
        );

        (uint256 rawToAssetAmount, uint256 usd) = convertAssetToAsset(
            fromAsset,
            toAsset,
            fromAssetAmount,
            whitelistPrices
        );

        // calculate & subtract swap fee
        uint256 feeFraction = LibSwap.calculateSwapFeeFractionForAssetSwap(
            fromAsset,
            toAsset,
            usd,
            whitelistPrices
        );
```

© 2023 Otter Audits LLC. All Rights Reserved.

```
        uint256 feeToAssetAmount = (rawToAssetAmount * feeFraction).divUp(1e18);
        uint256 toAssetAmount = rawToAssetAmount - feeToAssetAmount;

        require(
            poolState().assetBalances[toAsset] >= toAssetAmount,
            "insufficient toAsset"
        );

        return (toAssetAmount, feeToAssetAmount);
    }
```

Remediation

Remove the `fromAssetAmount` check from `_previewSwap`.

Patch

Fixed in c24da22.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-SUG-02 | Unused Parameters

Description

`checkTokensForOptions` currently includes two unused parameters, `indexAsset` and `isLong`. Unused parameters contribute to increased code size and gas consumption. Therefore, remove these parameters from the function to optimize the codebase and reduce unnecessary resource consumption.

contracts/pool/lib/option/OptionUtils.sol SOLIDITY

```
    function checkTokensForOptions(
        address collateralToken,
        address indexAsset,
        bool isLong
    ) internal view {
        if (!poolConfig().assetWhitelist.contains(collateralToken)) {
            revert AssetNotWhitelisted();
        }
    }
```

Remediation

Remove the `indexAsset` and `isLong` parameters.

Patch

Fixed in 89c8af6.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-SUG-03 | Demote Parameter Type To Calldata

Description

`previewMultiSwap`'s `tokenPath` parameter is currently defined as a `memory` type. Change this type to `calldata` for gas optimization.

contracts/manager/lib/LibSwapRouter.sol SOLIDITY

```
    /// @notice Preview route swap through multiple pools
    /// @param tokenPath Exclusive of inToken and outToken
    function previewMultiSwap(
        uint256 amountIn,
        address inToken,
        address[] memory tokenPath,
        address outToken,
        address[] calldata poolPath,
        OracleUpdatesPreview calldata oracleUpdatesPreview
    ) internal view returns (uint256[] memory amounts, uint256[] memory fees) {
        ...
    }
```

Remediation

Change the `tokenPath` type to `address[] calldata`.

contracts/manager/lib/LibSwapRouter.sol SOLIDITY

```
    /// @notice Preview route swap through multiple pools
    /// @param tokenPath Exclusive of inToken and outToken
    function previewMultiSwap(
        uint256 amountIn,
        address inToken,
        address[] calldata tokenPath,
        address outToken,
        address[] calldata poolPath,
        OracleUpdatesPreview calldata oracleUpdatesPreview
    ) internal view returns (uint256[] memory amounts, uint256[] memory fees) {
        ...
    }
```

Patch

Fixed in 53bbcf9.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-SUG-04 | Redundant Checks

Description

`_previewSwap` contains an asset whitelist check. However, `convertAssetToAsset`, called by `_previewSwap`, also performs the same check. Remove one of these checks for the sake of gas optimization.

contracts/manager/lib/LibSwapRouter.sol SOLIDITY

```
    /// @notice Returns the amount of `toAsset` that would be received (after fees)
    /// @param whitelistPrices base must be `fromAsset`, quote must be `toAsset`
    function _previewSwap(
        address fromAsset,
        address toAsset,
        uint256 fromAssetAmount,
        PriceDataList memory whitelistPrices
    )
        internal
        view
        returns (uint256 toAssetAmount_, uint256 feeToAssetAmount_)
    {
        if (!poolConfig().assetWhitelist.contains(fromAsset)) {
            revert AssetNotWhitelisted();
        }
        if (!poolConfig().assetWhitelist.contains(toAsset)) {
            revert AssetNotWhitelisted();
        }
        if (fromAsset == toAsset) {
            revert("fromAsset == toAsset");
        }

        require(
            poolState().assetBalances[fromAsset] >= fromAssetAmount,
            "insufficient fromAsset"
        );

        (uint256 rawToAssetAmount, uint256 usd) = convertAssetToAsset(
            fromAsset,
            toAsset,
            fromAssetAmount,
            whitelistPrices
        );
        ...
    }
```

contracts/pool/facets/pool/PoolSwapFacet.sol SOLIDITY

```
    /// @inheritdoc IPoolSwap
    function convertAssetToAsset(
        address fromAsset,
```

© 2023 Otter Audits LLC. All Rights Reserved.

```
        address toAsset,
        uint256 fromAssetAmount,
        PriceDataList memory pairPrices
    ) public view returns (uint256 toAssetAmount_, uint256 usd_) {
        if (!poolConfig().assetWhitelist.contains(fromAsset)) {
            revert AssetNotWhitelisted();
        }
        if (!poolConfig().assetWhitelist.contains(toAsset)) {
            revert AssetNotWhitelisted();
        }
        ...
    }
```

Remediation

Remove one of the asset whitelist checks.

Patch

Fixed in 0654c36.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-DRV-SUG-05 | Fix Error Prone Code Pattern

Description

The pool and manager contracts require developers to call `recordTransferOutETH` for every ether transfer, which may become a potential cause of human error in future code additions. The protocol validates the provision of user funds by comparing balances before and after operations.

If the contract does not call `recordTransferOutETH` after transferring ether, the variable designated for maintaining ether balances will be outdated. For future deposits, the balance snapshot will be larger than the actual value, resulting in an underestimation of the provided funds.

contracts/common/lib/LibTransfer.sol SOLIDITY

```
    /// Record transfer out ETH. MUST be called after transferring out ETH with
    ↪  msg.value
    function recordTransferOutETH() internal {
        _updateERC20Balance(address(0));
    }
```

contracts/common/lib/LibTransfer.sol SOLIDITY

```
    /// Record transferred in ETH
    function recordTransferInETH() internal returns (uint256 amount_) {
        uint256 balanceBefore = state().tokenBalance[address(0)];
        uint256 balanceAfter = address(this).balance;
        state().tokenBalance[address(0)] = balanceAfter;
        return balanceAfter - balanceBefore;
    }
```

Then, any user transferring the execution fee will, due to the outdated balance, have the `balanceBefore` value be higher than the actual balance, resulting in the returned amount being lower than the ETH the user sends.

Remediation

Implement a wrapper that transfers ether and balances updates within a single function. The wrapper reduces the plausibility of human error since manual insertion of balance updates will not be necessary.

Patch

Fixed in b69afbb.

© 2023 Otter Audits LLC. All Rights Reserved.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities that immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority or access control validation • Improperly designed economic incentives leading to loss of funds
High	Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities that could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation

© 2023 Otter Audits LLC. All Rights Reserved.

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of sum, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.

© 2023 Otter Audits LLC. All Rights Reserved.