Zellic

Y2K Finance

Smart Contract Security Assessment

October 30, 2023

Prepared for:

Last Oracle, 0xHarbs, 3xHarry, Crypwalk

Y2K Finance

Prepared by:

Katerina Belotskaia and Nipun Gupta

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for Y2K Finance from August 16th to September 4th, 2023. During this engagement, Zellic reviewed Y2K Finance's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Do users receive correct amounts when depositing, queuing deposits, withdrawing, queueing withdrawals, and claiming emissions?
- Could funds be blocked due to incomplete Stargate/LayerZero bridge transactions?
- Are there problems with incorrect balance management?

1.2 Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

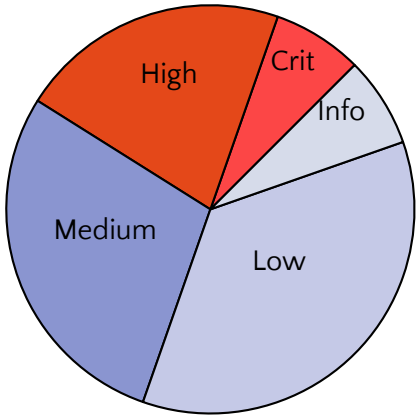
1.3 Results

During our assessment on the scoped Y2K Finance contracts, we discovered 14 findings. One critical issue was found. Three were of high impact, four were of medium impact, five were of low impact, and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Y2K Finance's benefit in the Discussion section (4) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	1
High	3
Medium	4
Low	5
Informational	1



2 Introduction

2.1 About Y2K Finance

Y2K Finance is an binary option protocol designed to create a marketplace for tail risks in DeFi, including pegged assets, directional markets, and real-world events that could pose a risk to the DeFi ecosystem. Market participants have the ability to robustly hedge or speculate on the risk of a particular market.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general.

We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3 Scope

The engagement involved a review of the following targets:

Y2K Finance Contracts

Repositories	https://github.com/Y2K-Finance/earthquake-xchain https://github.com/Y2K-Finance/strategy-vaults/
Versions	earthquake-xchain: `8dda65630e55c1b8eba7d134c77835d352d9343a` strategy-vaults: `2031cc8e3660bab0b2daf86f50111c1aa8b1e5e7`

Programs	bridgeController.sol swapController.sol vaultController.sol curve.sol uniswapV2.sol uniswapV2Dest.sol uniswapV3.sol uniswapV3Dest.sol zapDest.sol zapFrom.sol ERC4626.sol QueueContract.sol StrategyVault.sol HookChecker.sol PositionSizer.sol VaultGetter.sol HookAave.sol SwapRouter.sol HookAaveFixYield.sol
Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of four person-weeks. The assessment was conducted over the course of four calendar weeks.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia, Engineer
kate@zellic.io

Nipun Gupta, Engineer
nipun@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

August 16, 2023	Kick-off call
August 16, 2023	Start of primary review period
September 4, 2023	End of primary review period
September 8, 2023	Draft report delivered

3 Detailed Findings

3.1 Emissions can be claimed multiple times

- **Target**: StrategyVault
- **Category**: Business Logic
- **Likelihood**: High
- **Severity**: Critical
- **Impact**: Critical

Description

The function `claimEmissions` can be used by users to claim Y2K emissions. This function uses the correct vault balance of users to calculate the accumulated emissions and then subtracts the emission debt to find out the amount of emission tokens to be transferred to the users.

```
function claimEmissions(address receiver)
    external returns (uint256 emissions)
{
    int256 accEmissions = int256(
        (balanceOf[msg.sender] * accEmissionPerShare) / PRECISION
    );
    emissions = uint256(accEmissions - userEmissionDebt[msg.sender]);
    userEmissionDebt[msg.sender] = accEmissions;
    if (emissions > 0) emissionToken.safeTransfer(receiver, emissions);
    emit EmissionsClaimed(msg.sender, receiver, emissions);
}
```

A user can also transfer their vault tokens to another account after calling `claimEmissions`. As the emission debt is not transferred along with the vault balance, they can call `claimEmissions` again using their other account and claim these emissions again.

This process can be repeated multiple times, effectively draining all the emission tokens from the StrategyVault contract.

Impact

All the emission tokens can be drained out of the contract.

Recommendations

While transferring tokens using the functions `transfer` and `transferFrom`, it is important to update the `userEmissionDebt` mapping using the function `_updateUserEmissions`.

To do this, override the `_transfer` function, which is called in both `transfer` and `transferFrom` functions, to add the following additional logic.

```
function _transfer(address sender, address recipient, uint256 amount)
internal virtual override {
    _updateUserEmissions(sender,amount,false);
    _updateUserEmissions(recipient,amount,true);
    super._transfer(sender, recipient, amount);
}
```

Remediation

The issue was fixed in commit dd5470d and 7a57688.

3.2 The value of `queuedWithdrawalTvl` can be artificially inflated

- **Target**: StrategyVault
- **Category**: Business Logic
- **Likelihood**: High
- **Severity**: High
- **Impact**: High

Description

The value of `queuedWithdrawalTvl` can be artificially inflated, which might revert the transactions calling `fetchDeployAmounts` or `deployPosition` in StrategyVault and `_borrow` in the `HookAaveFixYield` and `HookAave` contracts.

When a user calls `requestWithdrawal`, the value of `totalQueuedShares[deployId]` is increased by the amount of shares. The user can then transfer their funds to another wallet and call `requestWithdrawal` again, which would increase the `totalQueuedShares[deployId]` for the second time.

This can be repeated multiple times to artificially increase the value of `totalQueuedShares[deployId]`. When the owner closes this position using `closePosition`, this value will be added to `queuedWithdrawalTvl`, thus increasing its value more than intended.

If the value of `queuedWithdrawalTvl` becomes greater than `totalAssets()` after a successful exploit, it will revert the function call `fetchDeployAmounts` and `deployPosition` in the StrategyVault contract due to integer underflow. This would also revert any call to `availableUnderlying`, which is called in `_borrow` in the hook contract.

Impact

Certain function calls would revert, and new positions cannot be deployed.

Recommendations

When a user requests withdrawal using the `requestWithdrawal`, these funds should not be allowed to be transferred to other wallets.

An additional check can be implemented in the `_transfer` function that checks that no more than `balanceOf[sender] - withdrawQueue[sender].shares` are transferred from the sender's address.

```
function _transfer(address sender, address recipient, uint256 amount)
internal virtual override {
    require(balanceOf[sender] - withdrawQueue[sender].shares >
        amount,"Not enough funds");
```

```
        _updateUserEmissions(sender,amount,false);
        _updateUserEmissions(recipient,amount,true);
        super._transfer(sender, recipient, amount);
    }
```

Remediation

The issue was fixed in commit 11f6797.

3.3 The lack of token addresses' verification

- **Target**: zapFrom
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: High
- **Impact**: High

Description

The `permitSwapAndBridge` and `swapAndBridge` functions allow users to perform the swap, and after that, bridge the resulting tokens to another chain using Stargate, which is a decentralised bridge and exchange building on top of the Layer Zero protocol.

These functions have the following parameters: `swapPayload`, `receivedToken`, and `_srcPoolId`. The `swapPayload` parameter contains all the necessary data for the swap, including the `path` array with the list of tokens involved in the swap process. It is assumed that the final address of the token participating in the swap will be used for cross-chain swap.

The `receivedToken` token address will be used by the `_bridge` function for assigning the approval for the stargateRouter contract.

Besides that, the user controls the `_srcPoolId` parameter, which determines the `pool` address, which is associated with a specific token and will hold the assets of the tokens that will be transferred from the current contact inside stargateRouter. However, there is no verification that these three addresses – `receivedToken`, the last address in `path` and `pool.token()` – match each other.

When using native tokens, the user should pass the `wethAddress` address as `receivedToken` because before `_bridge`, the necessary amount of tokens should be withdrawn from the weth contract. After that, the `receivedToken` will be rewritten to zero address. Currently there is no verification that the `receivedToken` is not zero initially.

```
function swapAndBridge(
    uint amountIn,
    address fromToken,
    address receivedToken,
    uint16 srcPoolId,
    uint16 dstPoolId,
    bytes1 dexId,
    bytes calldata swapPayload,
    bytes calldata bridgePayload
) external payable {
    _checkConditions(amountIn);
```

```
    ERC20(fromToken).safeTransferFrom(msg.sender, address(this),
    amountIn);

    uint256 receivedAmount;
    if (dexId != 0x05) {
        receivedAmount = _swap(dexId, amountIn, swapPayload);
    } else {
        ERC20(fromToken).safeApprove(balancerVault, amountIn);
        receivedAmount = _swapBalancer(swapPayload);
    }

    if (receivedToken == wethAddress) {
        WETH(wethAddress).withdraw(receivedAmount);
        receivedToken = address(0);
    }

    _bridge(
        receivedAmount,
        receivedToken,
        srcPoolId,
        dstPoolId,
        bridgePayload
    );
}

function _bridge(
    uint amountIn,
    address fromToken,
    uint16 srcPoolId,
    uint16 dstPoolId,
    bytes calldata payload
) private {
    if (fromToken == address(0)) {
/* NOTE: If sending after swap to ETH then msg.value will be < amountIn as
    it only contains the fee
    If sending without swap msg.value will be > amountIn as it contains
    both fee + amountIn
**/
        uint256 msgValue = msg.value > amountIn
            ? msg.value
            : amountIn + msg.value;
```

```
    IStargateRouter(stargateRouterEth).swapETHAndCall{value:
    msgValue}( ... );
     ...
    }
...
}
```

Impact

Due to the lack of verification that the `receivedToken` address matches the last address in the `path` array and `pool.token()` address, users are able to employ any token address as the `receivedToken`. This potentially allows them to successfully execute cross-chain swaps using tokens owned by the contract.

In instances where a user initially sets the `receivedToken` address to the zero address, the required amount of tokens will not be withdrawn from the weth contract. Consequently, the contract will attempt to transfer to the stargateRouter contract the funds present in its balance before the transaction took place.

In both scenarios, if the contract possesses any tokens, they can be utilized instead of the tokens received during the execution of the swap.

This also leads to a problem in the `_bridge` function during `msgValue` calculation. When the `fromToken` (`receivedToken` from `swapAndBridge`) is not the outcome of a swap, users can specify any `amountIn` as the result of a swap involving a different token. This `amountIn` value will then be used as the ETH value. Consequently, if the contract holds other funds, they will be sent to stargateRouterEth along with the user's fee.

Recommendations

We recommend to add the check that the `receivedToken` and the last address in `path` and `pool.token()` match each other — and also that the `receivedToken` address is not equal to zero address.

Remediation

This issue has been acknowledged by Y2K Finance, and a fix was implemented in commit 5f79149.

3.4 The lack of verification of the `payload` data

- **Target**: zapFrom
- **Category**: Coding Mistakes
- **Likelihood**: High
- **Severity**: High
- **Impact**: High

Description

Within the functions `bridge`, `permitSwapAndBridge`, and `swapAndBridge`, there is a lack of validation for the `payload` or `bridgePayload` data provided by users, which is transmitted to the stargateRouter contract for subsequent transmission to the destination chain.

The `sgReceive` function expects that `_payload` will include the `receiver` address, the vault's epoch `id`, and the `vaultAddress`. However, if the data type mismatches the expected format, the refund process using the `_stageRefund` function will not occur as the function call will result in a revert.

```
function sgReceive(
    uint16 _chainId,
    bytes memory _srcAddress,
    uint256 _nonce,
    address _token,
    uint256 amountLD,
    bytes calldata _payload
) external payable override {
    if (msg.sender ≠ stargateRelayer && msg.sender ≠ stargateRelayerEth)
        revert InvalidCaller();
    (address receiver, uint256 id, address vaultAddress) = abi.decode(
        _payload,
        (address, uint256, address)
    );

    if (id == 0) return _stageRefund(receiver, _token, amountLD);
    if (whitelistedVault[vaultAddress] ≠ 1)
        return _stageRefund(receiver, _token, amountLD);
    bool success = _depositToVault(id, amountLD, _token, vaultAddress);
    if (!success) return _stageRefund(receiver, _token, amountLD);

    receiverToVaultToIdToAmount[receiver][vaultAddress][id] += amountLD;
    emit ReceivedDeposit(_token, address(this), amountLD);
}
```

Impact

The absence of proper payload validation exposes the system to potential issues, as incorrect or malformed payloads could cause the subsequent `sgReceive` function call from the zapDest contract to revert. Such reverts could lead to locked funds and hinder the expected behavior of the system.

Recommendations

Instead of accepting raw payload data from users, we recommend encoding the payload data directly inside the functions `bridge`, `permitSwapAndBridge`, and `swapAndBridge`. This ensures that the payload is created according to the expected format and reduces the likelihood of incorrect payloads causing reverts of calls in the destination contract.

If the payload must be provided by users, we recommend to implement robust input validation mechanisms to ensure that only valid and properly formatted payloads are accepted.

Remediation

This issue has been acknowledged by Y2K Finance, and a fix was implemented in commit 56a1461.

3.5 Incorrect loop implementation in the function `clearQueuedDeposits`

- **Target**: StrategyVault
- **Category**: Coding Mistakes
- **Likelihood**: Medium
- **Severity**: Low
- **Impact**: Low

Description

The function `clearQueuedDeposits` is used to clear a fixed amount of deposits in the queue. This function loops through the `queueDeposits` mapping and pops the last element of the array while minting shares to the expected receivers in that mapping.

The issue is that the array indexing used to access `queueDeposits` is incorrect because the array index will be out of bound in many cases. Shown below is the relevant part of the code:

```
function clearQueuedDeposits(
    uint256 queueSize
) external onlyOwner returns (uint256 pulledAmount) {
    // ...
    for (uint256 i = depositLength - queueSize; i < queueSize; ) {
        QueueDeposit memory qDeposit = queueDeposits[queueSize - i - 1];
        uint256 shares = qDeposit.assets.mulDivDown(
            cachedSupply,
            cachedAssets
        );
```

Impact

In many cases the function might revert.

Recommendations

Consider changing the code to the following:

```
function clearQueuedDeposits(
    uint256 queueSize
) external onlyOwner returns (uint256 pulledAmount) {
    // ...
    for (uint256 i = depositLength; i > depositLength - queueSize; ) {
```

```
        QueueDeposit memory qDeposit = queueDeposits[i - 1];
        // ...
        unchecked {
            i--;
        }
```

Remediation

The issue was fixed in commit cf415dd.

3.6 Lack of data validation for `trustedRemoteLookup`

- **Target**: zapDest
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Informational
- **Impact**: Informational

Description

The current implementation of the `lzReceive` function lacks checks to verify the validity of the data stored in `trustedRemoteLookup[_srcChainId]` and `_srcAddress` bytes.

If `trustedRemoteLookup[_srcChainId]` is not set and `_srcAddress` is zero bytes, the result of the check `if (keccak256(_srcAddress) ≠ keccak256(trustedRemoteLookup[_srcChainId]))` will be true because `keccak256("") == keccak256("")`.

```
function lzReceive(
    uint16 _srcChainId,
    bytes memory _srcAddress,
    uint64 _nonce,
    bytes memory _payload
) external override {
    if (msg.sender ≠ layerZeroRelayer) revert InvalidCaller();
    if (
        keccak256(_srcAddress) ≠
        keccak256(trustedRemoteLookup[_srcChainId])
    ) revert InvalidCaller();
     ...
}
```

Impact

The issue currently has no security impact, because it is not expected that the layerZeroRelayer contract will send an empty `_srcAddress`. But limiting a contract's attack surface is a crucial way to mitigate future risks.

Recommendations

To ensure data consistency and avoid potential issues, it is recommended to add the following checks:

- `trustedRemoteLookup[_srcChainId] > 0`

- `_srcAddress.length == trustedRemoteLookup[_srcChainId].length`

An example of such checks can be found in the implementation provided by LayerZero Labs here.

Remediation

This issue has been acknowledged by Y2K Finance, and a fix was implemented in commit 32eaca8.

3.7 Array out-of-bound exception in `_removeVaults`

- **Target**: StrategyVault
- **Category**: Coding Mistakes
- **Likelihood**: Medium
- **Severity**: Low
- **Impact**: Low

Description

The function `_removeVaults` is a helper function that removes vaults from the `vaultList`. While removing a vault from the middle of the array, it is intended to replace the vault at the last index with the vault to be removed and pop the last vault.

The index of the last element of the array should be `removeCount - 1` (where `removeCount = vaults.length`), but the function is using the last element as `removeCount` — due to which it will revert because it would access element out-of-bounds of the array.

Shown below is the relevant part of the code:

```
function _removeVaults(
    address[] memory vaults
) internal returns (address[] memory newVaultList) {
    // ...
    } else {
        if (vaults.length > 1) {
            vaults[j] = vaults[removeCount];
            delete vaults[removeCount];
        } else delete vaults[j];
        removeCount--;
    }
    // ...
```

Impact

The `_removeVaults` function would revert in certain cases.

Recommendations

Use the correct last array index `removeCount - 1` instead of `removeCount`:

```
function _removeVaults(
    address[] memory vaults
```

```
) internal returns (address[] memory newVaultList) {
    // ...
    } else {
        if (vaults.length > 1) {
            vaults[j] = vaults[removeCount];
            delete vaults[removeCount];
            vaults[j] = vaults[removeCount - 1];
            delete vaults[removeCount - 1];
    } else delete vaults[j];
    removeCount--;
    }
    // ...
```

Remediation

The issue was fixed in commit fd2a6f3.

3.8 The function `_removeVaults` returns early

- **Target**: StrategyVault
- **Category**: Coding Mistakes
- **Likelihood**: Medium
- **Severity**: Low
- **Impact**: Low

Description

The function `_removeVaults` is a helper function that removes vaults from the `vaultList`. While removing the vaults, it runs two loops, but the return statement is inside the first loop, due to which this function returns after the first iteration of the first loop. The intended functionality is to return after both the loops are finished.

Impact

The `_removeVaults` function would return early, and all the vaults will not be removed from the list as intended.

Recommendations

Move the two lines outside of the loop:

```
function _removeVaults(
    address[] memory vaults
) internal returns (address[] memory newVaultList) {
    uint256 removeCount = vaults.length;
    newVaultList = vaultList;

    for (uint256 i; i < newVaultList.length; ) {
        for (uint j; j < removeCount; ) {
            if (vaults[j] == newVaultList[i]) {
                // Deleting the removeVault from the list
                if (j == removeCount) {
                    delete vaults[j];
                    removeCount--;
                } else {
                    if (vaults.length > 1) {
                        vaults[j] = vaults[removeCount];
                        delete vaults[removeCount];
                    } else delete vaults[j];
                    removeCount--;
                }
```

```
                // Deleting the vault from the newVaultList list
                if (
                    newVaultList[i]
== newVaultList[newVaultList.length - 1]
                ) {
                    delete newVaultList[i];
                } else {
                    newVaultList[i]
= newVaultList[newVaultList.length - 1];
                    delete newVaultList[newVaultList.length - 1];
                }
            }
            unchecked {
                j++;
            }
        }
        unchecked {
            i++;
        }

        vaultList = newVaultList;
        return newVaultList;
    }
        vaultList = newVaultList;
        return newVaultList;
}
```

Remediation

The issue was fixed in commit 945734b and 6bd136c.

3.9 The `weightStrategy` range violation

- **Target**: StrategyVault
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

The `weightStrategy` global variable determines the weight strategy used when deploying funds and can take one of three values:

1. for equal weight
2. for fixed weight
3. for threshold weight

However, the `setWeightStrategy` function allows the owner of the contract to set this value to a number less than or equal to `strategyCount()`, which is equal to 4.

```
function setWeightStrategy(
    uint8 weightId,
    uint16 proportion,
    uint256[] calldata fixedWeights
) external onlyOwner {
    ...
    if (weightId > strategyCount()) revert InvalidWeightId();
    ...
    weightStrategy = weightId;
    weightProportion = proportion;
    vaultWeights = fixedWeights;
    emit WeightStrategyUpdated(weightId, proportion, fixedWeights);
}

function strategyCount() public pure returns (uint256) {
    return 4;
}
```

Impact

If the `weightStrategy` is set to 4, the `fetchWeights` function will revert because there is a check that this value cannot be more than 3. As a result, the `deployPosition` function,

which is called by the owner of the contract, will also revert, preventing the owner from deploying funds to Y2K vaults.

Recommendations

We recommend to change the condition from > to ≥.

```
function setWeightStrategy(
    uint8 weightId,
    uint16 proportion,
    uint256[] calldata fixedWeights
) external onlyOwner {
    ...
    if (weightId > strategyCount()) revert InvalidWeightId();
    if (weightId ≥ strategyCount()) revert InvalidWeightId();
    ...
}
```

Remediation

The issue was fixed in commit 2248d6f.

3.10 Incompatibility with USDT token

- **Target**: VaultController
- **Category**: Business Logic
- **Likelihood**: Medium
- **Severity**: Medium
- **Impact**: Medium

Description

While depositing ERC-20 tokens to the vault, the contract first approves the token to the vault using `safeApprove` from the solmate library and then calls `deposit` on the earthquake vault in a try-catch. The code is as follows:

```
function _depositToVault(
        uint256 id,
        uint256 amount,
        address inputToken,
        address vaultAddress
    ) internal returns (bool) {
        if (inputToken == sgEth) {
            try
                IEarthquake(vaultAddress).depositETH{value: amount}(
                    id,
                    address(this)
                )
            {} catch {
                return false;
            }
        } else {
            ERC20(inputToken).safeApprove(address(vaultAddress), amount);
            try
                IEarthquake(vaultAddress).deposit(id, amount,
    address(this))
            {} catch {
                return false;
            }
        }
        return true;
    }
```

If the call to `deposit` on the earthquake vault fails, it would be caught using the catch statement and the function would simply return false. In this case, the approval would

not be decreased as the tokens would not be transferred to the earthquake vault. If this token is USDT, subsequent calls to `safeApprove` will revert, as USDT's `approve` function reverts if the current allowance is nonzero.

Impact

USDT deposits to the earthquake vault might fail in case any deposit to the vault fails.

Recommendations

Consider changing the code to the following:

```
function _depositToVault(
        uint256 id,
        uint256 amount,
        address inputToken,
        address vaultAddress
    ) internal returns (bool) {
        if (inputToken == sgEth) {
            try
                IEarthquake(vaultAddress).depositETH{value: amount}(
                    id,
                    address(this)
                )
            {} catch {
                return false;
            }
        } else {
            ERC20(inputToken).safeApprove(address(vaultAddress), 0);

            ERC20(inputToken).safeApprove(address(vaultAddress), amount);
            try
                IEarthquake(vaultAddress).deposit(id, amount,
    address(this))
            {} catch {
                return false;
            }
        }
        return true;
    }
```

Remediation

This issue has been acknowledged by Y2K Finance, and a fix was implemented in commit d17e221.

3.11 Conversion between different units does not account for token decimals

- **Target**: HookAave, HookAaveFixYield
- **Category**: Business Logic
- **Likelihood**: Medium
- **Severity**: Medium
- **Impact**: Medium

Description

The functions `_borrow` and `_repay` in the hook contracts are used to borrow and repay to Aave.

Taking an example of `_repay`, this function calculates the amount to be repaid using `balanceOf` on the variable debt token as well as the current balance of borrow tokens using `balanceOf` on the borrow token.

If the amount to be repaid is greater than the current balance of borrow tokens, the function `_swapForMissingBorrowToken` withdraws the deposit token and swaps these tokens to borrow tokens to repay the amount to Aave.

The amount to be withdrawn is calculated by the following code:

```
function _swapForMissingBorrowToken(
        address borrowToken,
        uint256 amountNeeded
    ) internal {
        ERC20 depositToken = strategyDepositToken;
        uint256 exchangeRate = (aaveOracle.getAssetPrice(borrowToken) *
            105e16) / aaveOracle.getAssetPrice(address(depositToken));
        uint256 amountToWithdraw = ((exchangeRate * amountNeeded) / 1e18);

        _withdraw(amountToWithdraw, false);
        _swap(amountToWithdraw, depositToken, 1);
    }
```

Although this would work if both tokens are of the same decimals, there would be an issue if these tokens (depositToken and borrowToken) are of different decimals.

For example, if borrowToken is ETH and depositToken is USDC, and the `amountNeeded` is 100 ETH, assuming the price of ETH to be $1,200, the value of `amountToWithdraw` would be calculated as 126,000e18 whereas it should be 126,000e6.

The same issue is also present in the `_repay` function.

Impact

Incorrect decimal conversion might lead to incorrect values during `_borrow` and `_repay`.

Recommendations

Take into account the decimals for all the tokens while such conversions take place.

Remediation

The issue was fixed in commits 80da566 and 0db93f7.

3.12 Malicious users can profit due to temporary exchange rate fluctuations

- **Target**: StrategyVault
- **Category**: Business Logic
- **Likelihood**: Low
- **Severity**: Medium
- **Impact**: Medium

Description

When a position is closed by calling `closePosition`, the deposit queue is cleared by pulling funds from the queue contract using the function `_pullQueuedDeposits`. The function `_pullQueuedDeposits` is only called when the length of `queueDeposits` is less than `maxQueuePull`.

If the length of this `queueDeposits` array is greater than `maxQueuePull`, the queue is first reduced using the function `clearQueuedDeposits`. The relevant part of the code is shown below:

```
function closePosition() external onlyOwner {
    if (!fundsDeployed) revert FundsNotDeployed();

    // ...
    fundsDeployed = false;
    // ...
    uint256 queueLength = queueDeposits.length;
    if (queueLength > 0 && queueLength < maxQueuePull)
        _pullQueuedDeposits(queueLength);
}
```

There may be a scenario where either the owner forgets to call the `clearQueuedDeposits` function before `closePosition` or a malicious user front-runs the owner's `closePosition` call to increase the length of the queue such that `_pullQueuedDeposits` is not called. In both these cases, the queue will not be cleared, but `fundsDeployed` would be set to false.

If the owner later tries to clear the queue by calling the function `clearQueuedDeposits` multiple times, the exchange rate would temporarily fluctuate. This is due to a bug in the function `clearQueuedDeposits`.

While clearing part of the queue, the function pulls all the funds from the QueueContract. At this time, the `totalSupply` is only increased by a small amount, but `totalAssets` is increased by a large amount. The exchange rate would again reach back to the

expected amount when the entire queue is cleared, but between the calls to `clearQueuedDeposits`, the exchange rate is incorrect. A malicious user can profit by calling `withdraw` between these calls as they would receive more assets than they should.

Impact

A malicious user can sell shares at higher price than expected.

Recommendations

In the function `clearQueuedDeposits`, only the assets that are removed from the queue should be pulled from the QueueContract.

Remediation

The issue was fixed in commit 86e24fe.

3.13 Incorrect `weights` calculation

- **Target**: PositionSizer
- **Category**: Coding Mistakes
- **Likelihood**: Medium
- **Severity**: Medium
- **Impact**: Medium

Description

The function `_thresholdWeight` performs a calculation of `weights` for a set of vaults based on their return on investment (ROI) compared to a threshold value. However, during the process of identifying valid vaults, the `validIds` array is populated with both valid indexes and zeros, which leads to unintended behavior.

The second loop iterates over this array to calculate `weights` only until `validCount`. But `validCount` is less than the actual `validIds` size. So the `weights` will be calculated only for the first `validCount` elements from the `validIds` array, regardless of whether they are valid indexes or zeros.

```
function _thresholdWeight(
    address[] memory vaults,
    uint256[] memory epochIds
) internal view returns (uint256[] memory weights) {
    ...
    for (uint256 i; i < vaults.length; ) {
        uint256 roi = _fetchReturn(vaults[i], epochIds[i], marketIds[i]);
        if (roi > threshold) {
            validCount += 1;
            validIds[i] = i;
        }
        unchecked {
            i++;
        }
    }
    ...
    uint256 modulo = 10_000 % validCount;
    for (uint j; j < validCount; ) {
        uint256 location = validIds[j];
        weights[location] = 10_000 / validCount;
        if (modulo > 0) {
            weights[location] += 1;
            modulo -= 1;
        }
```

```
            unchecked {
                j++;
            }
        }
    }
```

Impact

This behavior leads to missing `weights` calculations for a portion of the valid vaults.

Recommendations

We recommend correcting the second loop so that it iterates the entire length of the `validIds` array and counts the `weights` only if the `validIds[j]` is not zero.

Remediation

The issue was fixed in commit d9ee9d3.

3.14 Incorrect return value in `fetchEpochIds` in case of invalid vaults

- **Target**: VaultGetter
- **Category**: Coding Mistakes
- **Likelihood**: Medium
- **Severity**: Medium
- **Impact**: Low

Description

The function `fetchEpochIds` is used to get the list of `epochIds`, `validVaults`, and `vaultType` for the vaults that are active.

The function loops through the vault's array, calls the function `epochValid` for each array, and returns the `epochId`, `vaultType`, and a boolean `valid` that describes if the vault is valid or not. When a vault is invalid — in other words, `valid = false` — the counter `i` is increased but the `validCount` is not. If any vault is invalid, there would be a mismatch between the returned arrays.

```
function fetchEpochIds(
        address[] memory vaults
    )
        public
        view
        returns (
            uint256[] memory epochIds,
            address[] memory validVaults,
            uint256[] memory vaultType
        )
    {
        uint256 validCount;
        epochIds = new uint256[](vaults.length);
        validVaults = new address[](vaults.length);
        vaultType = new uint256[](vaults.length);

        for (uint256 i = 0; i < vaults.length; ) {
            IEarthquake vault = IEarthquake(vaults[i]);

            bool valid;
            (valid, epochIds[i], vaultType[i]) = epochValid(vault);
            unchecked {
                i++;
```

```
            }

            if (!valid) {
                continue;
            }

            validVaults[validCount] = address(vault);
            unchecked {
                validCount++;
            }
        }
    }
```

Impact

If the `weightStrategy` used is 3 (threshold), the function `_thresholdWeight` would revert in `VaultGetter.getRoi` as this function would try to call `totalSupply` on an `address(0)`. The function `_thresholdWeight` is internally called in `deployPosition`; therefore, deploying new positions might fail.

Recommendations

Revert the function `fetchEpochIds` if a vault in the list is invalid.

Remediation

The issue was fixed in commit 8e8d33e.

4 Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1 Variable naming suggestion

The contract zapDest is the cross-chain bridge receiver for Y2K vaults that implements the function `lzReceive`. The LayerZero Endpoint will invoke this function to deliver the message on the destination. In this contract, the global variable `layerZeroRelayer` is used to store the address of the LayerZero Endpoint that will call this function. We suggest to rename this variable as `layerZeroEndpoint` to avoid confusion.

This issue has been acknowledged by Y2K Finance, and a fix was implemented in commit 8b8d6ad7.

4.2 Documentation contains additional parameter that is not included in the code

The definition of payload in the comments suggests that it contains another encoded parameter named `depositType`, but this parameter is not included in the code. We suggest to update the documentation to reflect the definition of `payload` the same as what is being used in the code.

4.3 The function `_swapUniswapV2` can be rewritten as it only expects one token swap

The `_swapUniswapV2` function in uniswapV2Dest is used to swap one token for another using Uniswap V2 pools. The current implementation of this function is almost identical to the `_swapUniswapV2` defined in uniswapV2.

While the function in uniswapV2 was designed to swap multiple tokens, the function in uniswapV2Dest is just expected to swap one. Though there are no security concerns here, we suggest to simplify the current implementation of `_swapUniswapV2` in uniswapV2Dest to avoid complexity.

This issue has been acknowledged by Y2K Finance, and a fix was implemented in commit 859022b.

4.4 LayerZero configuration

We recommend implementing a set of functions that enable the configuration of a generic LayerZero user application in both the ZapFrom and ZapDest contracts.

```
// @notice set the send() LayerZero messaging library version to _version
// @param _version - new messaging library version
function setSendVersion(uint16 _version) external override onlyOwner {
    lzEndpoint.setSendVersion(_version);
}

// @notice set the lzReceive() LayerZero messaging library version to
    _version
// @param _version - new messaging library version
function setReceiveVersion(uint16 _version) external override onlyOwner {
    lzEndpoint.setReceiveVersion(_version);
}

// @notice Only when the UA needs to resume the message flow in blocking
    mode and clear the stored payload
// @param _srcChainId - the chainId of the source chain
// @param _srcAddress - the contract address of the source contract at the
    source chain
function forceResumeReceive(uint16 _srcChainId,
    bytes calldata _srcAddress) external override onlyOwner {
    lzEndpoint.forceResumeReceive(_srcChainId, _srcAddress);

function getConfig(
    uint16 _version,
    uint16 _chainId,
    address,
    uint _configType
) external view returns (bytes memory) {
    return lzEndpoint.getConfig(_version, _chainId, address(this),
    _configType);
}

// @notice set the configuration of the LayerZero messaging library of the
    specified version
// @param _version - messaging library version
// @param _chainId - the chainId for the pending config change
```

```
// @param _configType - type of configuration. every messaging library
    has its own convention.
// @param _config - configuration in the bytes. can encode arbitrary
    content.
function setConfig(
    uint16 _version,
    uint16 _chainId,
    uint _configType,
    bytes calldata _config
) external override onlyOwner {
    lzEndpoint.setConfig(_version, _chainId, _configType, _config);
}
```

This issue has been acknowledged by Y2K Finance, and fixes were implemented in the following commits:

- 37187ae1
- c0df7ed0

4.5 Use Non-blocking pattern instead of blocking pattern in lzReceive

LayerZero provides ordered delivery of messages from a given sender to a destination chain, i.e. srcUA-> dstChain. Therefore, when a transaction fails on the destination chain (eg: if `lzReceive` reverts for some reason), the channel between the source and destination is blocked.

If it isn't necessary to preserve the sequential nonce property then the non-blocking pattern should be used. An example implementation of non-blocking pattern by LayerZero is here: https://github.com/LayerZero-Labs/solidity-examples/blob/main/contracts/lzApp/NonblockingLzApp.sol

This issue has been acknowledged by Y2K Finance, and a fix was implemented in commit c0df7ed0.

4.6 Use reentrancy guards in deposit and withdraw functions

While our examination of the contracts did not reveal any instances of reentrancy scenarios, we strongly advise the implementation of reentrancy guards as a precautionary measure. The following functions lack protection against reentrancy:

- ZapFrom: The `swapAndBridge`, `permitSwapAndBridge` functions.
- ZapDest: The `withdraw` function.
- StrategyVault: The `deposit`, `withdraw`, `claimEmissions` and `withdrawFromQueue` function.

This recommendation is based on the potential for certain tokens to incorporate callbacks during transfers, which can introduce vulnerabilities if not adequately safeguarded against.

The issue was fixed in commit 46d5cba.

5 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

> Please note that our threat model was based on commits 8dda6563 and 2031cc8e, which represents a specific snapshot of the codebase. Therefore, it's important to understand that the absence of certain tests in our report may not reflect the current state of the test suite.
>
> During the remediation phase, Y2K Finance took proactive steps to address the findings by adding test cases where applicable in commit 761d1cb4. This demonstrates their dedication to enhancing the code quality and overall reliability of the system, which is commendable.

5.1 Module: ERC4626.sol

Function: `deposit(uint256 assets, address receiver)`

This deposits assets to mint shares in the contract.

Inputs

- `assets`
  - **Control**: Fully controlled.
  - **Constraints**: No constraints.
  - **Impact**: The amount of assets to deposit.
- `receiver`
  - **Control**: Fully controlled.
  - **Constraints**: No constraints.
  - **Impact**: The address to receive the minted shares.

Branches and code coverage (including function calls)

Intended branches

- The function calculates the preview deposit amount from assets and ensures it is not zero.
  - ☑ Test coverage
- The function transfers the assets from the caller to the contract.
  - ☑ Test coverage
- The function mints shares for the receiver.
  - ☑ Test coverage
- The function emits a `Deposit` event.
  - ☑ Test coverage
- The function calls `afterDeposit` to perform any necessary postdeposit actions.
  - ☑ Test coverage

Negative behavior

- The function reverts with `ZERO_SHARES` if the calculated shares are zero.
  - ☐ Negative test

Function call analysis

- `asset.safeTransferFrom(msg.sender, address(this), assets)`
  - **What is controllable?** `msg.sender` and `assets`.
  - **If return value controllable, how is it used and how can it go wrong?** This function call does not return a value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the asset transfer fails, it will revert and the transaction will be rolled back.
- `_mint(receiver, shares)`
  - **What is controllable?** `receiver` and `shares`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the minting of shares fails, it will revert and the transaction will be rolled back.
- `afterDeposit(assets, shares)`
  - **What is controllable?** `assets` and `shares`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the `afterDeposit` function reverts, the entire function would revert.

Function: `withdraw(uint256 shares, address receiver, address owner)`

Withdraws assets by burning shares and transferring assets to the receiver

Inputs

- `shares`
  - **Control**: Fully controlled.
  - **Constraints**: No constraints.
  - **Impact**: The amount of shares to burn for withdrawal.
- `receiver`
  - **Control**: Fully controlled.
  - **Constraints**: No constraints.
  - **Impact**: The address to receive the withdrawn assets.
- `owner`
  - **Control**: Fully controlled.
  - **Constraints**: Owner should either be the caller of the function or should have enough allowance
  - **Impact**: The owner of the shares.

Branches and code coverage (including function calls)

Intended branches

- The function calculates the assets to be withdrawn based on shares.
  ☑ Test coverage
- If the sender is not the owner, it updates the allowance for the sender to spend the owner's shares.
  ☑ Test coverage
- The function calls `beforeWithdraw` to perform any necessary prewithdraw actions.
  ☑ Test coverage
- The function burns the specified shares from the owner's balance.
  ☑ Test coverage
- The function emits a `Withdraw` event.
  ☑ Test coverage
- The function transfers assets to the receiver.
  ☑ Test coverage

Negative behavior

- The function reverts if the sender tries to withdraw more shares than allowed

by their allowance.
  - ☐ Negative test

Function call analysis

- `beforeWithdraw(assets, shares)`
  - **What is controllable?** `assets` and `shares`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the `beforeWithdraw` reverts, the entire function would revert.
- `_burn(owner, shares)`
  - **What is controllable?** `owner` and `shares`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the burning of shares fails, it will revert and the transaction will be rolled back.
- `asset.safeTransfer(receiver, assets)`
  - **What is controllable?** `receiver` and `assets`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the asset transfer fails, it will revert and the transaction will be rolled back.

5.2 Module: HookAaveFixYield.sol

Function: `updateWeeklyAppreciation(uint256 newAppreciation)`

This updates the weekly appreciation — in other words, the proportion of position to use.

Inputs

- `newAppreciation`
  - **Control**: Fully controlled.
  - **Constraints**: Should be greater than zero and less than or equal to 1e12.
  - **Impact**: The new fixed amount.

Branches and code coverage (including function calls)

Intended branches

- Updates the `weeklyAppreciation` with the provided `newAppreciation`.

☑ Test coverage

Negative behavior

- The function reverts if `newAppreciation` is zero.
  ☑ Negative test
- The function reverts if `newAppreciation` is greater than 1e12.
  ☑ Negative test

5.3 Module: HookAave.sol

Function: `afterClose()`

This repays the loan and swaps the excess to aTokens.

Branches and code coverage (including function calls)

Intended branches

- Repays the loan by calling `_repay(borrowToken)`.
  ☑ Test coverage
- Checks for excess tokens and swaps them to deposit tokens.
  ☑ Test coverage
- Transfers any remaining excess tokens to the strategy contract.
  ☑ Test coverage

Negative behavior

- The function reverts if the caller is an address other than the strategy contract.
  ☐ Negative test

Function call analysis

- `_repay(borrowToken)`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `_swap(excess, borrowToken, 0)`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

- `_supply(amountOut, false)`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `afterDeposit(uint256 amount)`

This deposits the amount into Aave (from deposits in strategy vault).

Inputs

- `amount`
  - **Control**: Fully controlled.
  - **Constraints**: No constraints.
  - **Impact**: The amount to deposit.

Branches and code coverage (including function calls)

Intended branches

- Calls the `_supply` function with the provided `amount` and `true` as arguments.
  ☑ Test coverage
- Emits an `AaveSupply` event.
  ☑ Test coverage

Negative behavior

- The function reverts if the caller is an address other than strategy.
  ☑ Negative test

Function call analysis

- `_supply(amount, true)`
  - **What is controllable?** `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `beforeDeploy()`

This borrows the max amount from Aave and sends it to the strategyVault.

Branches and code coverage (including function calls)

Intended branches

- The function calls `_borrow()` to borrow the max amount from Aave and transfers the borrowed amount to `msg.sender`.
  - ☑ Test coverage

Negative behavior

- The function reverts if `_borrow()` fails.
  - ☐ Negative test
- The function reverts if the caller is an address other than strategy.
  - ☐ Negative test

Function call analysis

- `_borrow()`
  - **What is controllable?** No external control.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is not controllable, but its impact is significant as it represents the borrowed amount from Aave.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `strategyBorrowToken.safeTransfer(msg.sender, borrowAmount)`
  - **What is controllable?** `msg.sender`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `beforeWithdraw(uint256 amount)`

This withdraws the amount from Aave (from withdrawals in strategy vault).

Inputs

- `amount`
  - **Control**: Fully controlled.
  - **Constraints**: No constraints.
  - **Impact**: The amount to withdraw.

Branches and code coverage (including function calls)

Intended branches

- Calls the `_withdraw` function with the provided `amount` and `true` as arguments.
  ☑ Test coverage
- Emits an `AaveWithdraw` event.
  ☑ Test coverage

Negative behavior

- The function reverts if the caller is an address other than strategy.
  ☐ Negative test

Function call analysis

- `_withdraw(amount, true)`
  - **What is controllable?** `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `setStrategyVault(address _strategy, address _borrowToken, address _lpToken)`

This sets the strategy vault info.

Inputs

- `_strategy`
  - **Control**: Fully controlled.
  - **Constraints**: Must not be the zero address.
  - **Impact**: The address of the strategy vault.
- `_borrowToken`
  - **Control**: Fully controlled.
  - **Constraints**: Must not be the zero address.
  - **Impact**: The address of the borrow token.
- `_lpToken`
  - **Control**: Fully controlled.
  - **Constraints**: Must not be the zero address.
  - **Impact**: The address of the LP token.

Branches and code coverage (including function calls)

Intended branches

- The global storage variables `strategyInUse`, `strategyDepositToken`, `strategyBorrowToken`, and `strategyLPToken` are updated properly.
  ☑ Test coverage

Negative behavior

- The function reverts if `_strategy`, `_borrowToken`, or `_lpToken` is the zero address.
  ☑ Test coverage

Function call analysis

- `ERC20 _depositToken = IStrategyVault(_strategy).asset();`
  - **What is controllable?** `_strategy`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is assigned to `_depositToken`, and its use is not controllable. The function is used to retrieve the asset of the strategy vault.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, it indicates that `_strategy` is not a valid strategy vault contract.

Function: `updatePath(IRouter.Path path, bool updatingAll)`

This updates the path information used for swaps.

Inputs

- `path`
  - **Control**: Fully controlled.
  - **Constraints**: The `toAmountMin` fields in `path.BToD` and `path.DToB` must be greater than zero.
  - **Impact**: The new path info.
- `updatingAll`
  - **Control**: Fully controlled.
  - **Constraints**: No constraints.
  - **Impact**: If true, it will update all path info; if false, it will only update `toAmountMin` and the `updated` timestamp.

Branches and code coverage (including function calls)

Intended branches

- The function updates the path info — either all fields or just `toAmountMin` and `updated`.
  ☑ Test coverage

Negative behavior

- The function reverts if `toAmountMin` fields in `path.BToD` or `path.DToB` are zero.
  ☑ Negative test
- The function reverts if `updatingAll` is true and the route info or token placement is invalid.
  ☑ Negative test

5.4 Module: QueueContract.sol

Function: `transferToQueue(address caller, uint256 amount)`

This transfers asset from the caller to the QueueContract.

Inputs

- `caller`
  - **Control**: Fully controlled.
  - **Constraints**: No constraints.
  - **Impact**: The caller of the function.
- `amount`
  - **Control**: Fully controlled.
  - **Constraints**: No constraints.
  - **Impact**: The amount to transfer.

Branches and code coverage (including function calls)

Intended branches

- The function checks if the `msg.sender` has a valid asset and transfers the specified amount.
  ☑ Test coverage
- The function emits a `QueueDeposit` event.
  ☑ Test coverage

Negative behavior

- The function reverts if the asset is `address(0)` or if the `transferFrom` operation fails.

☑ Negative test

Function call analysis

- `asset.transferFrom(caller, address(this), amount)`
  - **What is controllable?** `caller` and `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `transferToStrategy()`

This transfers assets from the QueueContract to the caller (strategyVault).

Branches and code coverage (including function calls)

Intended branches

- The function checks if the caller's vault has a nonzero balance and transfers the balance.
  ☑ Test coverage
- The function emits a `DepositsCleared` event.
  ☑ Test coverage

Negative behavior

- The function reverts if the vault balance is zero or if the transfer operation fails.
  ☑ Negative test

Function call analysis

- `depositAsset[msg.sender].transfer(msg.sender, vaultBalance)`
  - **What is controllable?** `msg.sender` and `vaultBalance`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

5.5 Module: StrategyVault.sol

Function: `claimEmissions(address receiver)`

This can be called to any state. If `accEmissions` is less than `userEmissionDebt[msg.sender]`, the calculated `emissions` value can be a huge number as result of cast `int256` to

`uint256`.

Inputs

- `receiver`
  - **Constraints**: No.
  - **Impact**: The receiver of emissionToken.

Branches and code coverage (including function calls)

Intended branches

- claim before withdraw tokens
  ☑ Test coverage
- claim after withdraw tokens
  ☑ Test coverage

Function call analysis

- `emissionToken.safeTransfer(receiver, emissions)`
  - **What is controllable?**: receiver
  - **If return value controllable, how is it used and how can it go wrong?**: n/a
  - **What happens if it reverts, reenters, or does other unusual control flow?**: can revert if the balance of contract less than `emissions` value

Function: `clearQueuedDeposits(uint256 queueSize)`

The function allows the contract owner to clear the fixed amount of deposits in the end of queue. Also it mints the appropriate amount of shares for receivers from queue and transfer deposited tokens from QueueContract to this contract.

Inputs

- `queueSize`
  - **Constraints**: No checks.
  - **Impact**: The amount of elements of `queueDeposits` will be deleted.

Branches and code coverage (including function calls)

Intended branches

- The `queueSize` is equal to `queueDeposits.length`.
  ☑ Test coverage

- `queueSize` is less than `queueDeposits.length`.
  - ☐ Test coverage

Negative behavior

- `queueSize` is more than `queueDeposits.length`.
  - ☐ Negative test
- `queueSize` is zero.
  - ☐ Negative test

Function call analysis

- `_updateUserEmissions(qDeposit.receiver, shares, true)`
  - **What is controllable?** The `qDeposit.receiver` value from `queueDeposits` — `shares` is calculated here.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem.
- `_mint(qDeposit.receiver, shares);`
  - **What is controllable?** The `qDeposit.receiver` value from `queueDeposits` — `shares` is calculated here.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem.
- `queueContract.transferToStrategy();`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** The function transfers full queue balance to the strategy, but it should only transfer the `pulledAmount`.
- `asset.safeApprove(address(hook.addr), pulledAmount);`
  - **What is controllable?** The `pulledAmount` value is calculated here — the full amount of assets tokens cleared from queue. The `hook.addr` is set by the owner of the contract.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem. The function is called if `hook.command.shouldCallAfterDeposit() == true`.
- `hook.addr.afterDeposit(pulledAmount);`
  - **What is controllable?** The `pulledAmount` value is calculated here — the full amount of assets tokens cleared from queue.

- **If return value controllable, how is it used and how can it go wrong?** N/A.
- **What happens if it reverts, reenters, or does other unusual control flow?** `hook.command.shouldCallAfterDeposit() == true`.

Function: `closePosition()`

The `fundsDeployed` should be `true`. After the call is set to `false`, `deploymentId` is increased. This allows the owner of the contract to trigger close position and withdraw funds from Y2K vaults.

Branches and code coverage (including function calls)

Negative behavior

- `fundsDeployed` is false.
  ☑ Negative test
- Second call after successful closing of position.
  ☑ Negative test

Function call analysis

- `hook.addr.beforeClose();`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** This function is called if `hook.command.shouldCallBeforeClose() == true`. The `hook.addr` is controlled by the owner of the contract.
- `_closePosition(position)`
  - **What is controllable?** Nothing controllable directly by caller; the `position` data was filled during the `deployPosition` call.
  - **If return value controllable, how is it used and how can it go wrong?** The function returns nothing.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Withdraws funds from vaults — can revert due to problems with withdrawal in external contract.
- `_transferAssets(hook.addr.afterCloseTransferAssets())`
  - **What is controllable?** Nothing controllable directly by the caller. The `afterCloseTransferAssets` function returns the list of tokens.
  - **If return value controllable, how is it used and how can it go wrong?** The function returns nothing.
  - **What happens if it reverts, reenters, or does other unusual control flow?** This function is called if `hook.command.shouldTransferAfterClose() == tru`

e. The function transfers asset tokens from the returned list to the receiver (`hook.addr`). The current balance of contract passed as `value` argument to the `safeTransfer(IERC20 token, address to, uint256 value)` function, so there will not be a situation that there are not enough tokens to transfer.

- `hook.addr.afterClose()`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** The function returns nothing.
  - **What happens if it reverts, reenters, or does other unusual control flow?** This function is called if `hook.command.shouldCallAfterClose() == true`. The `hook.addr` is controlled by the owner of the contract.
- `previewRedeem(totalQueuedShares[deploymentId])`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** Return the amount of asset token for the corresponding amount of shares, which are added to the queue for withdrawal for current `depolyId`. If `totalQueuedShares[deploymentId]` is calculated incorrecly, the resulting asset value also will be wrong.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem.
- `_pullQueuedDeposits(queueLength)`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem.

Function: `deployPosition()`

The `fundsDeployed` should be `false`. After the call, `fundsDeployed` = true.

Branches and code coverage (including function calls)

Negative behavior

- The `fundsDeployed` is true.
  ☑ Negative test
- Second call after successful deploy of position
  ☑ Negative test

Function call analysis

- `hook.addr.beforeDeploy()`

  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** This function is called if `hook.command.shouldCallBeforeDeploy() == true`. In the case of the HookAave contract, this function borrows the max amount from Aave and sends it to the strategyVault. It can revert if the `strategyBorrowToken` balace of the hook.addr contract is less than `borrowAmount`.
- `fetchDeployAmounts()`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** With incorrect matching of elements between lists.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem.
- `_deployPosition(vaults, epochIds, amounts, vaultType);`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Deposit tokens to `vaults`. It can revert if balance of `asset` token is not enough.
- `hook.addr.afterDeploy()`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** This function is called if `hook.command.shouldCallAfterDeploy() == true`.

Function: `deposit(uint256 assets, address receiver)`

If `fundsDeployed` is `true`, the deposit will be done over `_queueDeposit`. It allows any caller to perform deposit of assets tokens and receive the shares. In case `fundsDeployed == true`, then the funds will be placed in the deposit queue.

Inputs

- `assets`
  - **Constraints**: The caller should own this amount of asset tokens.
  - **Impact**: The amount of asset tokens will be deposited.
- `receiver`
  - **Constraints**: N/A.
  - **Impact**: The receiver of shares.

Branches and code coverage (including function calls)

Intended branches

- The balance of the receiver's shares has increased properly.
  ☑ Test coverage
- The asset balance of `msg.sender` decreased by `assets` amount.
  ☑ Test coverage

Negative behavior

- The balance of `msg.sender` is less than `assets`.
  ☑ Negative test

Function call analysis

- `_queueDeposit(receiver, assets)`
  - **What is controllable?** `receiver` and `assets`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** The function is called in case of `fundsDeployed == true` after the `deployPosition` function call by owner. Will revert if `assets` is less than `minDeposit`, if `queueContract` has not been approved from `msg.sender`.
- `asset.safeTransferFrom(msg.sender, address(this), assets);`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Can revert if current contract does not have an approve from `msg.sender` and if balance of `msg.sender` is less than `assets` amount.
- `_mint(receiver, shares);`
  - **What is controllable?** `receiver`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem.
- `hook.addr.afterDeposit(assets)`
  - **What is controllable?** `assets`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Can reenter but without negative impact because the function is called after all states changes.

Function: `requestWithdrawal(uint256 shares)`

This cannot be called if `fundsDeployed` is false. It allows the owner of shares to add shares to the queue for withdrawal.

Inputs

- `shares`
  - **Constraints**: The amount of shares should be less than or equal to the rest of shares of `msg.sender`, which were not added to the queue.
  - **Impact**: The amount of shares will be added to the queue for withdrawal.

Branches and code coverage (including function calls)

Intended branches

- `withdrawQueue[msg.sender].shares` was increased by shares amount.
  ☑ Test coverage
- `totalQueuedShares[deployId]` was increased by shares amount.
  ☑ Test coverage

Negative behavior

- Shares amount is more than `balanceOf[msg.sender] - withdrawQueue[msg.sender].shares`.
  ☑ Negative test
- The `fundsDeployed` is false.
  ☑ Negative test

Function: `setWeightStrategy(uint8 weightId, uint16 proportion, uint256[] fixedWeights)`

A restricted `onlyOwner` function. There are not checks of `fixedWeights` values if `weightId == 1` or 4. Also, `fixedWeights` can be an empty list.

Inputs

- `weightId`
  - **Constraints**: `0 < weightId ≤ 4`, but it should be `0 < weightId < 4`.
  - **Impact**: defines the strategy to use for weights. If `weightId == 3`, `fixedWeights` contains `marketIds` and threshold. If `weightId == 2`, then `fixedWeights` contains the custom weights. If `weightId == 1`, then `equalWeight`, that is, all available funds will be equally distributed between the vaults.
- `proportion`

  - **Constraints**: `proportion ≤ 9999`.
  - **Impact**: Proportion of vault funds to use in each deployment to strategy.
- `fixedWeights`
  - **Constraints**: If `weightId == 2`, then the sum of weights should not be more than 10,000.
  - **Impact**: Update the global `vaultWeights`, which is used by `PositionSizer._thresholdWeight` and `PositionSizer._fixedWeight`.

Branches and code coverage (including function calls)

Intended branches

- `weightStrategy`, `weightProportion`, and `vaultWeights` are updated properly.
  ☑ Test coverage

Negative behavior

- `weightId == 0`.
  ☑ Negative test
- `proportion > 9_999`.
  ☑ Negative test
- `weightId > strategyCount()`.
  ☑ Negative test
- `weightId == 2` and `fixedWeights.length ≠ vaultList.length`.
  ☑ Negative test
- `weightId == 3` and `fixedWeights.length ≠ vaultList.length + 1`.
  ☑ Negative test

Function: `unrequestWithdrawal(uint256 shares)`

This can be called only before `closePosition` because after close position, the `deployId` will be increased.

Inputs

- `shares`
  - **Constraints**: The shares cannot be more than the shares from `withdrawQueue[msg.sender]`.
  - **Impact**: The amount of shares for withdrawal.

Branches and code coverage (including function calls)

Intended branches

- The withdrawal is performed properly.
  ☑ Test coverage

Negative behavior

- The current `deployId` ≠ `withdrawQueue[msg.sender].queue[length - 1].deploymentId`.
  ☑ Negative test
- Shares > `item.shares`.
  ☑ Negative test

Function: `updateActiveList(address[] vaults, UpdateAction updateAction)`

A restricted `onlyOwner` function.. Allows to update, delete `vaultList`, add new vaults, and remove vaults.

Inputs

- `vaults`
  - **Constraints**: The vault's list will be checked by the `checkVaultsValid` function. The vault addresses, `_vault.asset()`, `_vault.controller()`, `_vault.treasury()`, `_vault.emissionsToken()`, and `_vault.counterPartyVault()` should not be zero.
  - **Impact**: The new vault's addresses.
- `updateAction`
  - **Constraints**: Can be one of type — `DeleteVaults`, `AppendVaults`, `ReplaceVaults`, or `DeleteVaults`.
  - **Impact**: The type of action.

Branches and code coverage (including function calls)

Intended branches

- Check that the `vaultList` has updated properly after every type of action.
  ☑ Test coverage

Negative behavior

- Check a vaults list that contains, in addition to the valid vault addresses, also zero addresses.
  ☑ Negative test

Function call analysis

- `_appendVaults(vaults)`
  - **What is controllable?** `vaults`.
  - **If return value controllable, how is it used and how can it go wrong?** Return the new vault list with appended elements.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem.
- `_replaceVaults(vaults)`
  - **What is controllable?** `vaults`.
  - **If return value controllable, how is it used and how can it go wrong?** Return the new vault list with new elements.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem.
- `_removeVaults(vaults)`
  - **What is controllable?** `vaults`.
  - **If return value controllable, how is it used and how can it go wrong?** Return the new vault list without removed elements.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem.

Function: `withdrawFromQueue(address receiver, address owner)`

This can be called only after `requestWithdrawal`. It allows the owner or user who has allowance to withdraw asset tokens that have been queued for withdrawal.

Inputs

- `receiver`
  - **Constraints**: No verification, but if `msg.sender` is not an owner, `allowance[owner][msg.sender]` should be more than or equal to `shares`.
  - **Impact**: The address of the receiver of asset tokens.
- `owner`
  - **Constraints**: `withdrawQueue[owner].shares` should not be zero.
  - **Impact**: The owner of shares.

Branches and code coverage (including function calls)

Intended branches

- The assets tokens were transferred properly and shares were burned.
  ☐ Test coverage

Negative behavior

- `withdrawQueue[owner].shares == 0`.
  ☑ Negative test
- `msg.sender` != owner and allowance < shares.
  ☐ Negative test

Function call analysis

- `_previewQueuedWithdraw(owner)`
  - **What is controllable?** `owner`.
  - **If return value controllable, how is it used and how can it go wrong?** Return the full amount of shares in queue and calculate the appropriate amount of asset tokens.
  - **What happens if it reverts, reenters, or does other unusual control flow?** There is no problem.
- `_withdraw(assets, shares, receiver, owner)`
  - **What is controllable?** `receiver` and `owner`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Can revert if asset balance of contract is less than assets amount. Also, it can revert if the owner does not have enough shares.

Function: `withdraw(uint256 shares, address receiver, address owner)`

The user cannot withdraw if `fundsDeployed` is `true`. It allows to withdraw assets tokens from strategy.

Inputs

- `shares`
  - **Constraints**: If `shares` is less than the owner balance, the function will be reverted during `burn`.
  - **Impact**: The amount of shares to withdraw. The asset number will be calculated using the `shares` amount.
- `receiver`
  - **Constraints**: No verification, but if `msg.sender` is not an owner, `allowance[owner][msg.sender]` should be more than or equal to `shares`.
  - **Impact**: The address of the receiver of asset tokens.
- `owner`
  - **Constraints**: `withdrawQueue[owner].shares` should not be zero.
  - **Impact**: The owner of shares.

Branches and code coverage (including function calls)

Intended branches

- The assets tokens were transferred properly and shares were burned.
  ☑ Test coverage

Negative behavior

- `msg.sender` != owner and allowance < shares.
  ☐ Test coverage
- The balance of owner is less than shares.
  ☑ Test coverage

Function call analysis

- `_withdraw(assets, shares, receiver, owner)`
  - **What is controllable?** `receiver` and `owner`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Can revert if asset balance of the contract is less than assets amount. Also, it can revert if the owner does not have enough shares.

5.6 Module: SwapRouter.sol

Function: `swap(PathRoute path, uint256 amount, address receiver)`

This swaps tokens using different routing paths and DEXs.

Inputs

- `path`
  - **Control**: Fully controlled.
  - **Constraints**: The function would revert if `path.route` is not one of these values: 1, 2, or 3.
  - **Impact**: The routing path information.
- `amount`
  - **Control**: Fully controlled.
  - **Constraints**: No constraints.
  - **Impact**: The amount of tokens to swap.
- `receiver`
  - **Control**: Fully controlled.

- **Constraints**: No constraints.
- **Impact**: The address to receive the swapped tokens.

Branches and code coverage (including function calls)

Intended branches

- The function checks the `route` value in `path` and calls the corresponding DEX-specific swap function.
  ☑ Test coverage
- The function returns the amount of swapped tokens.
  ☑ Test coverage

Negative behavior

- The function reverts with `IErrors.InvalidPath()` if the provided route value is not recognized.
  ☐ Negative test

Function call analysis

- `CamelotSwapper._swapOnCamelot(path.bestPath, amount, path.toAmountMin, receiver)`
  - **What is controllable?** `path.bestPath`, `amount`, `path.toAmountMin`, and `receiver`.
  - **If return value controllable, how is it used and how can it go wrong?** This function call returns the amount of tokens received after swapping.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the swap on Camelot fails, it will revert and the transaction will be rolled back.
- `SushiSwapper._swapOnSushi(path.bestPath, amount, path.toAmountMin, receiver)`
  - **What is controllable?** `path.bestPath`, `amount`, `path.toAmountMin`, and `receiver`.
  - **If return value controllable, how is it used and how can it go wrong?** This function call returns the amount of tokens received after swapping.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the swap on SushiSwap fails, it will revert and the transaction will be rolled back.
- `_swapOnUniswapV3(path.bestPath, path.fee, amount, path.toAmountMin, receiver)`
  - **What is controllable?** `path.bestPath`, `path.fee`, `amount`, `path.toAmountMin`, and `receiver`.

- **If return value controllable, how is it used and how can it go wrong?** This function call returns the amount of tokens received after swapping.
- **What happens if it reverts, reenters, or does other unusual control flow?** If the swap on UniswapV3 fails, it will revert and the transaction will be rolled back.

5.7 Module: bridgeController.sol

Function: `_bridgeToSource(byte[1] _bridgeId, address _receiver, address _token, uint256 _amount, uint16 _sourceChainId, byte[] _withdrawPayload)`

This bridges the token to the destination chain via the selected bridge.

Inputs

- `_bridgeId`
  - **Constraints**: Has to be one of 0x01, 0x02 or 0x03.
  - **Impact**: The ID of the bridge to use (e.g., 0x01, 0x02, 0x03).
- `_receiver`
  - **Constraints**: No constraints.
  - **Impact**: The address to receive the bridged tokens.
- `_token`
  - **Constraints**: No constraints.
  - **Impact**: The address of the token to bridge.
- `_amount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of the token to bridge.
- `_sourceChainId`
  - **Constraints**: No constraints.
  - **Impact**: The ID of the chain the token is being bridged from (always calling chain).
- `_withdrawPayload`
  - **Constraints**: Should be encoded in the correct format based on the selected bridge.
  - **Impact**: The payload to decode for the extra inputs for each bridge.

Branches and code coverage (including function calls)

Intended branches

- The function calls the appropriate `_bridgeWith` function based on `_bridgeId`.
  ☑ Test coverage

Negative behavior

- The function reverts if `_bridgeId` is not one of the supported bridge identifiers.
  ☑ Negative test

Function call analysis

- `_bridgeWithCeler(_receiver, _token, _amount, _sourceChainId, abi.decode(_withdrawPayload, (uint256)))`
  - **What is controllable?** `_receiver`, `_token`, `_amount`, `_sourceChainId`, and `_withdrawPayload`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `_bridgeWithHyphen(_receiver, _token, _amount, _sourceChainId)`
  - **What is controllable?** `_receiver`, `_token`, `_amount`, and `_sourceChainId`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `_bridgeWithHop(_receiver, _token, _amount, _sourceChainId, maxSlippage, bonderFee)`
  - **What is controllable?** `_receiver`, `_token`, `_amount`, `_sourceChainId`, `maxSlippage`, and `bonderFee`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `_bridgeWithCeler(address _receiver, address _token, uint256 _amount, uint16 _dstChainId, uint256 maxSlippage)`

This bridges with Celer.

Inputs

- `_receiver`
  - **Constraints**: No constraints.
  - **Impact**: The address to receive the bridged tokens.
- `_token`
  - **Constraints**: No constraints.

  - **Impact**: The address of the token to bridge.
- `_amount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of the token to bridge.
- `_dstChainId`
  - **Constraints**: No constraints.
  - **Impact**: The ID of the chain the token is being bridged to.
- `maxSlippage`
  - **Constraints**: No constraints.
  - **Impact**: The max slippage allowed for the bridge.

Branches and code coverage (including function calls)

Intended branches

- The function approves the `celerBridge` contract to spend `_amount` of `_token`.
  ☑ Test coverage
- The function call `celerBridge.send( … )` succeeds without reverting.
  ☑ Test coverage

Negative behavior

- The function reverts if the approval for `celerBridge` fails.
  ☐ Negative test

Function call analysis

- `ERC20(_token).safeApprove(address(celerBridge), _amount)`
  - **What is controllable?** `_token` and `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `celerBridge.send(_receiver, _token, _amount, _dstChainId, uint64(block.timestamp), uint32(maxSlippage))`
  - **What is controllable?** `_receiver`, `_token`, `_amount`, `_dstChainId`, `block.timestamp`, and `maxSlippage`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `_bridgeWithHop(address _receiver, address _token, uint256 _amount, uint16 _srcChainId, uint256 maxSlippage, uint256 bonderFee)`

This bridges with Hop.

Inputs

- `_receiver`
  - **Constraints**: No constraints.
  - **Impact**: The address to receive the bridged tokens.
- `_token`
  - **Constraints**: No constraints.
  - **Impact**: The address of the token to bridge.
- `_amount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of the token to bridge.
- `_srcChainId`
  - **Constraints**: No constraints.
  - **Impact**: The ID of the chain the token is being bridged from.
- `maxSlippage`
  - **Constraints**: Should be less than or equal to 10,000.
  - **Impact**: The max slippage allowed for the bridge — input of 100 would be 1% slippage.
- `bonderFee`
  - **Constraints**: No constraints.
  - **Impact**: The fee to pay the bonder.

Branches and code coverage (including function calls)

Intended branches

- The function calculates `amountOutMin` correctly based on `maxSlippage`.
  ☑ Test coverage
- The function approves the `bridgeAddress` to spend `_amount` of `_token`.
  ☑ Test coverage
- The function call to `HopBridge` succeeds without reverting.
  ☑ Test coverage

Negative behavior

- The function reverts if `bridgeAddress` is `address(0)`.
  ☐ Negative test

- The function reverts if the approval for `bridgeAddress` fails.
  - ☐ Negative test
- The function reverts if `maxSlippage` is greater than 10,000.
  - ☐ Negative test
- The function reverts if the external call to `HopBridge` reverts.
  - ☐ Negative test

Function call analysis

- `ERC20(_token).safeApprove(bridgeAddress, _amount)`
  - **What is controllable?** `_token` and `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `IHopBridge(bridgeAddress).swapAndSend(_srcChainId, _receiver, _amount, bonderFee, amountOutMin, deadline, (amountOutMin * 998) / 1000, deadline)`
  - **What is controllable?** `_srcChainId`, `_receiver`, `_amount`, `bonderFee`, and `amountOutMin`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is not controlled, but its impact is significant as it bridges the tokens; incorrect values could lead to incorrect token transfers.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `_bridgeWithHyphen(address _receiver, address _token, uint256 _amount, uint16 _srcChainId)`

This bridges with Hyphen.

Inputs

- `_receiver`
  - **Constraints**: No constraints.
  - **Impact**: The address to receive the bridged tokens.
- `_token`
  - **Constraints**: No constraints.
  - **Impact**: The address of the token to bridge.
- `_amount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of the token to bridge.
- `_srcChainId`

  - **Constraints**: No constraints.
  - **Impact**: The ID of the chain the token is being bridged from.

Branches and code coverage (including function calls)

Intended branches

- The function approves the `hyphenBridge` contract to spend `_amount` of `_token`.
  ☑ Test coverage
- The function call `hyphenBridge.depositErc20( … )` succeeds without reverting.
  ☑ Test coverage

Negative behavior

- The function reverts if the approval for `hyphenBridge` fails.
  ☑ Negative test

Function call analysis

- `ERC20(_token).safeApprove(address(hyphenBridge), _amount)`
  - **What is controllable?** `_token` and `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `hyphenBridge.depositErc20(_srcChainId, _token, _receiver, _amount, "Y2K")`
  - **What is controllable?** `_srcChainId`, `_token`, `_receiver`, and `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

5.8 Module: curve.sol

Function: `zapInMulti(byte[] payload)`

This decodes the payload and calls `_multiSwap`.

Inputs

- `payload`
  - **Constraints**: Should decode to the correct format.
  - **Impact**: The data to decode and pass to multiSwap.

Branches and code coverage (including function calls)

Intended branches

- The payload decodes to the correct format, and the internal call to `_multiSwap` succeeds.
  - ☑ Test coverage

Negative behavior

- Revert if payload does not decode to the correct format.
  - ☐ Negative test

Function: `_multiSwap(address[] path, address[] pools, uint256[] iValues, uint256[] jValues, uint256 fromAmount, uint256 toAmountMin)`

Delegates the swap logic for each swap/pair to `_swapEth` or `_swap`.

Inputs

- `path`
  - **Constraints**: No constraints.
  - **Impact**: An array of the tokens being swapped between.
- `pools`
  - **Constraints**: No constraints.
  - **Impact**: An array of Curve pools to swap with.
- `iValues`
  - **Constraints**: No constraints.
  - **Impact**: An array of indexes of the `fromToken` in each Curve pool.
- `jValues`
  - **Constraints**: No constraints.
  - **Impact**: An array of indexes of the `toToken` in each Curve pool.
- `fromAmount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of `fromToken` to swap.
- `toAmountMin`
  - **Constraints**: No constraints.
  - **Impact**: The minimum amount of `toToken` to receive from the swap.

Branches and code coverage (including function calls)

Intended branches

- If any of the path is `address(0)`, use `_swapEth` for swapping tokens; otherwise, use `_swap`.
  ☑ Test coverage

Negative behavior

- Revert if `amountOut == 0`.
  ☑ Negative test

Function: `_swapEth(address fromToken, address toToken, address pool, uint256 i, uint256 j, uint256 fromAmount, uint256 toAmountMin)`

This swaps on Curve with the logic for an ETH pool.

Inputs

- `fromToken`
  - **Constraints**: No constraints.
  - **Impact**: The token being swapped from.
- `toToken`
  - **Constraints**: No constraints.
  - **Impact**: The token being swapped to.
- `pool`
  - **Constraints**: No constraints.
  - **Impact**: The Curve pool being swapped with.
- `i`
  - **Constraints**: No constraints.
  - **Impact**: The index of the `fromToken` in the Curve pool.
- `j`
  - **Constraints**: No constraints.
  - **Impact**: The index of the `toToken` in the Curve pool.
- `fromAmount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of `fromToken` to swap.
- `toAmountMin`
  - **Constraints**: No constraints.
  - **Impact**: The minimum amount of `toToken` to receive from the swap.

Branches and code coverage (including function calls)

Intended branches

- The swap on Curve succeeds with the given parameters.
  ☑ Test coverage

Negative behaviour

- The swap on Curve fails if output tokens are less than `toAmountMin`.
  ☐ Negative test

Function call analysis

- `ERC20(fromToken).safeApprove(pool, fromAmount)`
  - **What is controllable?** `fromToken`, `pool`, and `fromAmount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `ERC20(toToken).balanceOf(address(this))`
  - **What is controllable?** `toToken`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `ICurvePair(pool).exchange(i, j, fromAmount, toAmountMin, false)`
  - **What is controllable?** `pool`, `i`, `j`, `fromAmount`, and `toAmountMin`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `_swapWithCurve(byte[] payload)`

This decodes the payload and routes to `_swap`, `_swapEth`, or `_multiSwap` depending on `swapType`.

Inputs

- `payload`
  - **Constraints**: Should decode to the correct format.
  - **Impact**: The data to decode and pass to the correct function.

Branches and code coverage (including function calls)

Intended branches

- If swapType is 0x01, 0x02, or 0x03, use the correct swap method.
  ☑ Test coverage

Negative behavior

- Revert if `amountOut == 0`.
  - ☐ Negative test
- Revert if `swapType` is not one of 0x01, 0x02 or 0x03.
  - ☑ Negative test

Function: `_swap(address fromToken, address toToken, address pool, int128 i, int128 j, uint256 fromAmount, uint256 toAmountMin)`

This swaps on Curve with the logic for an ERC-20 pool.

Inputs

- `fromToken`
  - **Constraints**: No constraints.
  - **Impact**: The token being swapped from.
- `toToken`
  - **Constraints**: No constraints.
  - **Impact**: The token being swapped to.
- `pool`
  - **Constraints**: No constraints.
  - **Impact**: The Curve pool being swapped with.
- `i`
  - **Constraints**: No constraints.
  - **Impact**: The index of the `fromToken` in the Curve pool.
- `j`
  - **Constraints**: No constraints.
  - **Impact**: The index of the `toToken` in the Curve pool.
- `fromAmount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of `fromToken` to swap.
- `toAmountMin`
  - **Constraints**: No constraints.
  - **Impact**: The minimum amount of `toToken` to receive from the swap.

Branches and code coverage (including function calls)

Intended branches

- The swap on Curve succeeds with the given parameters.
  - ☑ Test coverage

Negative behaviour

- The swap on Curve fails if output tokens are less than `toAmountMin` .
  ☐ Negative test

Function call analysis

- `ERC20(fromToken).safeApprove(pool, fromAmount)`
  - **What is controllable?** `fromToken`, `pool`, and `fromAmount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `ERC20(toToken).balanceOf(address(this))`
  - **What is controllable?** `toToken`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `ICurvePair(pool).exchange(i, j, fromAmount, toAmountMin)`
  - **What is controllable?** `pool`, `i`, `j`, `fromAmount`, and `toAmountMin`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

5.9 Module: swapController.sol

Function: `_swapBalancer(byte[] swapPayload)`

This swaps using the `balancerVault`.

Inputs

- `swapPayload`
  - **Constraints**: No constraints.
  - **Impact**: The payload for the swap — varies by DEX.

Branches and code coverage (including function calls)

Intended branches

- The function calls the `balancerVault` with the provided `swapPayload`.
  ☑ Test coverage
- The function checks if the provided selector matches for a single swap or as-

sumes multiswap if not.
  ☑ Test coverage
- In case of single swap, returns the decoded `uint256` amount of `toToken` received.
  ☑ Test coverage
- In case of multiswap, checks negative asset deltas for received amounts and reverts if none are found.
  ☑ Test coverage

Negative behavior

- The function reverts if the `balancerVault` call is not successful.
  ☐ Negative test

Function call analysis

- `balancerVault.call(swapPayload)`
  - **What is controllable?** `swapPayload`.
  - **If return value controllable, how is it used and how can it go wrong?** Success used for condition; data used for decoding results.
  - **What happens if it reverts, reenters, or does other unusual control flow?** This function call can revert if the `balancerVault` call fails — no reentrancy scenarios.

Function: `_swap(byte[1] dexId, uint256 fromAmount, byte[] swapPayload)`

This uses the `dexId` to route the swap to the correct DEX logic.

Inputs

- `dexId`
  - **Constraints**: Should be one of 0x01, 0x02, 0x03, or 0x04.
  - **Impact**: The `dexId` of the DEX to be used (e.g., 0x01, 0x02, 0x03, or 0x04).
- `fromAmount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of `fromToken` to be swapped.
- `swapPayload`
  - **Constraints**: No constraints.
  - **Impact**: The payload for the swap — varies by DEX.

Branches and code coverage (including function calls)

Intended branches

- The function routes the swap based on `dexId`.
  ☑ Test coverage

Negative behavior

- The function reverts if `dexId` is not one of the supported DEX identifiers.
  ☑ Negative test

Function call analysis

- `_swapUniswapV2(0x01/0x02, fromAmount, swapPayload)`
  - **What is controllable?** 0x01 or 0x02 (from `_swap`), `fromAmount`, and `swapPayload`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `_swapUniswapV3(fromAmount, swapPayload)`
  - **What is controllable?** `fromAmount` and `swapPayload`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `_swapWithCurve(swapPayload)`
  - **What is controllable?** `swapPayload`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

5.10 Module: uniswapV2.sol

Function: `_executeSwap(address[] path, address[] pairs, uint256[] amounts)`

This executes swaps on the UniswapV2 fork.

Inputs

- `path`
  - **Constraints**: No constraints.
  - **Impact**: The array of token addresses to swap between.
- `pairs`
  - **Constraints**: No constraints.

  - **Impact**: The array of pairs to swap through.
- `amounts`
  - **Constraints**: No constraints.
  - **Impact**: The array of amounts to swap with each pair.

Branches and code coverage (including function calls)

Intended branches

- The function correctly swaps tokens through the specified pairs.
  ☑ Test coverage
- The swap logic works as expected for multiple pairs or a single pair,
  ☑ Test coverage

Function call analysis

- `IUniswapPair(pairs[0]).swap(zeroForOne ? 0 : amounts[0], zeroForOne ? amounts[0] : 0, pairs[1], "")`
  - **What is controllable?** `pairs[0]`, `zeroForOne`, `amounts[0]`, and `pairs[1]`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `IUniswapPair(pairs[i]).swap(zeroForOne ? 0 : amounts[i], zeroForOne ? amounts[i] : 0, pairs[i + 1], "")`
  - **What is controllable?** `pairs[i]`, `zeroForOne`, `amounts[i]`, and `pairs[i + 1]`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `IUniswapPair(pairs[pairs.length - 1]).swap(zeroForOne ? 0 : amounts[pairs.length - 1], zeroForOne ? amounts[pairs.length - 1] : 0, address(this), "")`
  - **What is controllable?** `pairs[pairs.length - 1]`, `zeroForOne`, and `amounts[pairs.length - 1]`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `_swapUniswapV2(byte[1] dexId, uint256 fromAmount, byte[] payload)`

This decodes the payload and conducts the swaps.

Inputs

- `dexId`
  - **Constraints**: Should be either 0x01 or 0x02,
  - **Impact**: The ID for the DEX being used (0x01 for UniswapV2, 0x02 for SushiSwap).
- `fromAmount`
  - **Constraints**: No constraints,
  - **Impact**: The amount of the `fromToken` being swapped.
- `payload`
  - **Constraints**: Should be encoded in the correct format — `abi.encode(address[] path, uint256 minAmountOut)`.
  - **Impact**: The encoded payload for the swap.

Branches and code coverage (including function calls)

Intended branches

- The function correctly determines the `initCodeHash` and factory based on the `dexId`.
  ☑ Test coverage
- The swap ratios and amounts are calculated correctly.
  ☑ Test coverage

Negative behavior

- The function reverts if the final `amountOut` is less than `toAmountMin`.
  ☐ Negative test

Function call analysis

- `_getPair(fromToken, toToken, initCodeHash, factory)`
  - **What is controllable?** `fromToken` and `toToken`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is used as an address for the pair; if manipulated, it can lead to swapping on the wrong pair.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `IUniswapPair(pairs[i]).getReserves()`
  - **What is controllable?** `pairs[i]`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value (`reserveA` and `reserveB`) impacts the calculation of swap amounts; this cannot be controlled.

  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `SafeTransferLib.safeTransfer(ERC20(path[0]), pairs[0], fromAmount)`
  - **What is controllable?** `path[0]`, `pairs[0]`, and `fromAmount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `_executeSwap(path, pairs, amounts)`
  - **What is controllable?** `path`, `pairs`, and `amounts`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

5.11 Module: uniswapV3.sol

Function: `uniswapV3SwapCallback(int256 amount0Delta, int256 amount1Delta, byte[] _data)`

This is the callback implementation for UniswapV3 pools.

Inputs

- `amount0Delta`
  - **Constraints**: Either one of `amount0Delta` or `amount1Delta` should be greater than zero.
  - **Impact**: The amount of `token0` received.
- `amount1Delta`
  - **Constraints**: Either one of `amount0Delta` or `amount1Delta` should be greater than zero.
  - **Impact**: The amount of `token1` received.
- `_data`
  - **Constraints**: Should correctly decode to `tokenIn`, `tokenOut`, and `fee`.
  - **Impact**: The encoded pool address, fee, and `tokenOut` address.

Branches and code coverage (including function calls)

Intended branches

- The function requires either `amount0Delta` or `amount1Delta` to be greater than zero.
  - ☑ Test coverage

Negative behaviour

- The function reverts if neither `amount0Delta` nor `amount1Delta` are greater than zero.
  - ☐ Negative test
- The function reverts if the caller is the incorrect pool.
  - ☑ Negative test

Function call analysis

- `decodePool(_data)`
  - **What is controllable?** `_data`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is used to extract `tokenIn`, `tokenOut`, and `fee`; if manipulated, it could lead to incorrect token transfers.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `getPool(tokenIn, tokenOut, fee)`
  - **What is controllable?** `tokenIn`, `tokenOut`, and `fee`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is used as the caller of the function; if manipulated, an incorrect pool could be considered as the caller.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `SafeTransferLib.safeTransfer(ERC20(tokenIn), msg.sender, amount0Delta > 0 ? uint256(amount0Delta) : uint256(amount1Delta))`
  - **What is controllable?** `tokenIn`, `msg.sender`, `amount0Delta`, and `amount1Delta`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `_executeSwap(address tokenIn, address tokenOut, uint256 fromAmount, uint24 fee)`

This executes the swap with the simulated V3 pool from `tokenIn`, `tokenOut`, and `fee`.

Inputs

- `tokenIn`
  - **Constraints**: Should be a valid input token, such that the pool address generated is correct.

  - **Impact**: The address of the `fromToken`.
- `tokenOut`
  - **Constraints**: Should be a valid output token, such that the pool address generated is correct
  - **Impact**: The address of the `toToken`.
- `fromAmount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of `fromToken` to swap.
- `fee`
  - **Constraints**: Should be the valid fee, such that the pool address generated is correct.
  - **Impact**: The fee for the pool.

Branches and code coverage (including function calls)

Intended branches

- The function handles swaps from `tokenIn` to `tokenOut` correctly.
  ☑ Test coverage

Function call analysis

- `getPool(tokenIn, tokenOut, fee)`
  - **What is controllable?** `tokenIn`, `tokenOut`, and `fee`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is used to select the pool for the swap.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `IUniswapV3Pool(getPool(tokenIn, tokenOut, fee)).swap( ... )`
  - **What is controllable?** `tokenIn`, `tokenOut`, `fee`, `address(this)`, `zeroForOne`, `int256(fromAmount)`, `sqrtPriceLimitX96`, and data.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is used to extract the `amountOut`.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

Function: `_swapUniswapV3(uint256 fromAmount, byte[] payload)`

This decodes the payload and conducts the swaps.

Inputs

- `fromAmount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of the `fromToken` being swapped.
- `payload`
  - **Constraints**: No constraints.
  - **Impact**: The encoded payload for the swap.

Branches and code coverage (including function calls)

Intended branches

- The function handles swaps for multitoken paths correctly.
  ☑ Test coverage

Negative behavior

- The function reverts if the resulting `amountOut` is less than `toAmountMin`.
  ☐ Negative test

Function call analysis

- `_executeSwap(path[0], path[1], fromAmount, fee[0])`
  - **What is controllable?** `path[0]`, `path[1]`, `fromAmount`, `fee[0]`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is used as the input for the next swap.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `_executeSwap(path[i], path[i + 1], amountOut, fee[i])`
  - **What is controllable?** `path[i]`, `path[i + 1]`, `amountOut` and `fee[i]`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is used as the input for the next swap.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.
- `_executeSwap(path[path.length - 2], path[path.length - 1], amountOut, fee[path.length - 2])`
  - **What is controllable?** `path[path.length - 2]`, `path[path.length - 1]`, `amountOut`, `fee[path.length - 2]`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value is the final `amountOut`.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If this reverts, the entire call fails — no reentrancy issues.

5.12 Module: vaultController.sol

Function: `_claimRefund(address sender, address token)`

This claims a refund for the original sender and token.

Inputs

- `sender`
  - **Constraints**: No constraints.
  - **Impact**: The address of the original sender.
- `token`
  - **Constraints**: No constraints.
  - **Impact**: The address of the token to refund.

Branches and code coverage (including function calls)

Intended branches

- The function checks if the sender is eligible for a refund and the amount to be refunded.
  ☑ Test coverage
- The function transfers the refunded amount to the sender either in ETH or the specified token.
  ☑ Test coverage
- The function emits a `RefundClaimed` event.
  ☑ Test coverage

Negative behavior

- The function reverts if the sender is not eligible for a refund (i.e., the mapping is zero).
  ☑ Negative test
- The function reverts if the ETH transfer fails.
  ☐ Negative test

Function call analysis

- `payable(sender).call{value: amount}("")`
  - **What is controllable?** `sender`.
  - **If return value controllable, how is it used and how can it go wrong?** The return value indicates the success of the call, and the data provides more details in case of failure.

- **What happens if it reverts, reenters, or does other unusual control flow?** If the call fails, it will revert and the transaction will be rolled back. The checks-effects-interactions pattern is followed.

Function: `_depositToVault(uint256 id, uint256 amount, address inputToken, address vaultAddress)`

Deposits ERC-20 or ETH to the vault.

Inputs

- `id`
  - **Constraints**: Should be the correct epoch ID for the Y2K vault.
  - **Impact**: The epoch ID for the Y2K vault.
- `amount`
  - **Constraints**: No constraints.
  - **Impact**: The amount of the token to deposit.
- `inputToken`
  - **Constraints**: No constraints.
  - **Impact**: The address of the token to deposit.
- `vaultAddress`
  - **Constraints**: Should be a valid vault address.
  - **Impact**: The address of the vault to deposit to.

Branches and code coverage (including function calls)

Intended branches

- The function checks if `inputToken` is equal to `sgEth` to determine if an ETH or ERC-20 deposit is needed.
  ☑ Test coverage
- The function returns `true` if the deposit is successful and `false` if it fails.
  ☑ Test coverage

Negative behavior

- The function returns `false` if any of the deposit attempts (ETH or ERC-20) fail.
  ☑ Negative test

Function call analysis

- `IEarthquake(vaultAddress).depositETH{value: amount}(id, address(this))`
  - **What is controllable?** `id`, `amount`, `vaultAddress`.

  - **If return value controllable, how is it used and how can it go wrong?** This function call does not return a value — only success/failure.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the deposit of ETH fails, the function would return false.
- `IEarthquake(vaultAddress).deposit(id, amount, address(this))`
  - **What is controllable?** `id`, `amount`, and `vaultAddress`.
  - **If return value controllable, how is it used and how can it go wrong?** This function call does not return a value — only success/failure.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the deposit of ERC-20 tokens fails, the function would return false.

Function: `_withdrawFromVault(uint256 id, uint256 assets, address receiver, address vaultAddress)`

This withdraws from the vault.

Inputs

- `id`
  - **Constraints**: Should be the correct epoch ID for the Y2K vault.
  - **Impact**: The epoch ID for the Y2K vault.
- `assets`
  - **Constraints**: No constraints.
  - **Impact**: The amount of the token to withdraw.
- `receiver`
  - **Constraints**: No constraints.
  - **Impact**: The address to receive the withdrawn tokens.
- `vaultAddress`
  - **Constraints**: Should be a valid vault address.
  - **Impact**: The address of the vault to withdraw from.

Branches and code coverage (including function calls)

Intended branches

- The function calls the `withdraw` function of the IEarthquake contract to initiate the withdrawal.
  ☑ Test coverage

Negative behavior

- The function reverts if the `withdraw` function call fails.

☑ Negative test

Function call analysis

- `IEarthquake(vaultAddress).withdraw(id, assets, receiver, address(this))`
  - **What is controllable?** `id`, `assets`, `receiver`, and `vaultAddress`.
  - **If return value controllable, how is it used and how can it go wrong?** This function call returns the actual amount of assets withdrawn.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If the withdrawal fails, it will revert and the transaction will be rolled back — no reentrancy scenarios.

5.13 Module: zapDest.sol

Function: `claimRefund(address token, address sender)`

This allows to claim a refund for the original sender and token. The `eligibleRefund[sender][token]` should not be zero. The `eligibleRefund` is set if the `sgReceive` function has failed.

Inputs

- `token`
  - **Constraints**: `eligibleRefund[sender][token] ≠ 0`.
  - **Impact**: The address of tokens that will be refunded.
- `sender`
  - **Constraints**: `eligibleRefund[sender][token] ≠ 0`.
  - **Impact**: The receiver of token refund.

Branches and code coverage (including function calls)

Intended branches

- Refund is performed properly for token `== sgEth`.
  ☑ Test coverage
- Refund is performed properly for token `!= sgEth`.
  ☑ Test coverage

Negative behavior

- `eligibleRefund[sender][token] == 0` for `msg.sender`.
  ☑ Negative test

- Repeated function call after successful refund.
  - ☐ Negative test

Function call analysis

- `_claimRefund → payable(sender).call{value: amount}("");`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Can reenter but without negative impact.
- `_claimRefund → ERC20(token).safeTransfer(sender, amount)`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Can reenter but without negative impact.

Function: `lzReceive(uint16 _srcChainId, byte[] _srcAddress, uint64 _nonce, byte[] _payload)`

This allows to receive the message from other chains using LayerZero protocol. Only LZ Endpoint can call this function. The function allows to perform one of the actions that is determined by the value of the `funcSelector`. If `funcSelector == 0x01` then `_withdrawFromVault` function will be invoke to direct withdraw funds from vaultAddress to the `receiver` address in this chain. If `funcSelector == 0x03` then the withdrawn from vaultAddress tokens will be swaped for the token requested by the user and after that bridged to the other chain. Otherwise the withdrawn from vaultAddress tokens will be just bridged to the other chain.

Inputs

- `_srcChainId`
  - **Constraints**: There is a check that `trustedRemoteLookup` for `_srcChainId` is equal to `_srcAddress`.
  - **Impact**: The messages can be received only from trusted chains.
- `_srcAddress`
  - **Constraints**: There is a check that `trustedRemoteLookup` for `_srcChainId` is equal to `_srcAddress`.
  - **Impact**: The messages can be received only from trusted user application from other chains.
- `_nonce`
  - **Constraints**: N/A.

  - **Impact**: Not used.
- `_payload`
  - **Constraints**: N/A.
  - **Impact**: Contains the data: the action, `bridgeId`, `receiver`, `id`, and `vaultAddress`.

Branches and code coverage (including function calls)

Intended branches

- The swap over UniswapV2 performed properly.
  ☐ Test coverage
- The swap over UniswapV3 performed properly.
  ☐ Test coverage

Negative behavior

- `msg.sender` is not `layerZeroRelayer`.
  ☑ Negative test
- `trustedRemoteLookup` is not set.
  ☐ Negative test
- `_srcChainId` is untrusted.
  ☑ Negative test
- `swapId > 0x02`.
  ☐ Negative test

Function call analysis

- `_withdraw(funcSelector, bridgeId, receiver, id, _srcChainId, vaultAddress, _payload);` → `_withdrawFromVault(id, assets, receiver, vaultAddress)`
  - **What is controllable?** `id` and `vaultAddress`.
  - **If return value controllable, how is it used and how can it go wrong?** n/a
  - **What happens if it reverts, reenters, or does other unusual control flow?** the whitelisted trusted `vaultAddress` contract is called which transfer the `assets` amount to the receiver. If `receiver` address is wrong then funds can be lost, but in case of calling from other chain over lz the address of receiver is msg.sender address, so it cannot be wrong.
- `_withdraw(funcSelector, bridgeId, receiver, id, _srcChainId, vaultAddress, _payload);` → `_swapToBridgeToken(amountReceived, asset, _payload);`
  - **What is controllable?** _payload
  - **If return value controllable, how is it used and how can it go wrong?** return the resulted token address, the rest of payload and the final swap

amount. The token address is controlled by user, but for this address the valid uniswap pair contract should exists, otherwise function will revert.
    - **What happens if it reverts, reenters, or does other unusual control flow?** can revert if uniswap pair is not exist for `token` and `toToken` addresses.
- `_withdraw(funcSelector, bridgeId, receiver, id, _srcChainId, vaultAddress, _payload);` → `_bridgeToSource(bridgeId, receiver, asset, _srcChainId, _payload);`
    - **What is controllable?** bridgeId, _payload, _srcChainId
    - **If return value controllable, how is it used and how can it go wrong?** n/a
    - **What happens if it reverts, reenters, or does other unusual control flow?** can revert due to issues during bridging

Function: `sgReceive(uint16 _chainId, byte[] _srcAddress, uint256 _nonce, address _token, uint256 amountLD, byte[] _payload)`

Allows to receive tokens from another chain. Only `stargateRelayer` (Router) can call this function. The native token will be transferred to this contract before triggering this function.

Inputs

- `_chainId`
    - **Constraints**: N/A.
    - **Impact**: Not used.
- `_srcAddress`
    - **Constraints**: N/A.
    - **Impact**: Not used.
- `_nonce`
    - **Constraints**: N/A.
    - **Impact**: Not used.
- `_token`
    - **Constraints**: No checks.
    - **Impact**: The address of token will be deposited to the vault. This address is received from `pool`.
- `amountLD`
    - **Constraints**: This value is not controlled by the user who initiated the token transfer between chains. This value is a result of swap.
    - **Impact**: The amount of tokens received. This amount of `_token` will be deposited to the vault and the `receiverToVaultToIdToAmount` for the receiver will be increased by `amountLD` value.
- `_payload`

- **Constraints**: `vaultAddress` should be whitelisted.
- **Impact**: Contains this data — `receiver`, `id`, and `vaultAddress`.

Branches and code coverage (including function calls)

Intended branches

- Deposit in case `token == sgEth`.
  ☑ Test coverage
- Deposit in case `token ≠ sgEth`.
  ☑ Test coverage

Negative behavior

- `msg.sender` is not `stargateRelayer` or `stargateRelayerEth`.
  ☑ Negative test
- `vaultAddress` is not whitelisted.
  ☐ Negative test

Function call analysis

- `_stageRefund(receiver, _token, amountLD)`
  - **What is controllable?** `receiver`, `_token`, and `amountLD`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** The function increments the global `eligibleRefund[receiver][_token]` by a `amountLD` value. This `eligibleRefund` value is used in the `claimRefund`→`_claimRefund` function.
- `_depositToVault(id, amountLD, _token, vaultAddress)`
  - **What is controllable?** `id`, `amountLD`, `_token`, and `vaultAddress`.
  - **If return value controllable, how is it used and how can it go wrong?** Returns a boolean value, successfully or unsuccessfully the external call of `deposit` function was executed.
  - **What happens if it reverts, reenters, or does other unusual control flow?** the function deposits funds to the vaultAddress contract.

Function: `withdraw(byte[1] funcSelector, byte[1] bridgeId, address receiver, uint256 id, uint16 _srcChainId, address vaultAddress, byte[] _withdrawPayload)`

The direct call of the `_withdraw` function is only available for `receiver` for which the value `receiverToVaultToIdToAmount[receiver][vaultAddress][id]` is not zero.

5.14 Module: zapFrom.sol

Function: `bridge(uint256 amountIn, address fromToken, uint16 srcPoolId, uint16 dstPoolId, byte[] payload)`

There is no check that `msg.value` is not less than `amountIn` in case of `fromToken == address(0)`. The function allows to bridge and deposit to vaults using Stargate.

Inputs

- `amountIn`
  - **Constraints**: $\neq 0$.
  - **Impact**: The amount of token for the swap.
- `fromToken`
  - **Constraints**: There is no check, but it should be the same address as `pool.token()`.
  - **Impact**: The address of the token in `srcPoolId`.
- `srcPoolId`
  - **Constraints**: `router.swap` reverts if `factory.getPool(_poolId)` returns zero address.
  - **Impact**: The ID of the SRC pool.
- `dstPoolId`
  - **Constraints**: The owner of the router should create and activate the chain path for `_dstChainId` and `_dstPoolId`. The `_dstChainId` is constant `ARBITRUM_CHAIN_ID`.
  - **Impact**: The ID of the DST pool.
- `payload`
  - **Constraints**: This is not verified.
  - **Impact**: Contain the data for ZapDest, expected address receiver, `uint256 vaultId`, and address `vaultAddress`.

Branches and code coverage (including function calls)

Intended branches

- The Stargate `swap()` is performed properly.
  - ☐ Test coverage

Negative behavior

- `msg.value < amountIn` and `fromToken == address(0)`.
  - ☐ Negative test

- `msg.sender` does not have enough `fromToken` tokens.
  - ☐ Negative test

Function call analysis

- `ERC20(fromToken).safeTransferFrom(msg.sender,address(this),amountIn)`
  - **What is controllable?** `fromToken` and `amountIn`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Can reenter but without negative impact.
- `_bridge(amountIn, fromToken, srcPoolId, dstPoolId, payload)` → `IStargateRouter(stargateRouterEth).swapETHAndCall{value: msgValue}`
  - **What is controllable?** `amountIn`, `fromToken`, `srcPoolId`, `dstPoolId`, and `payload`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Deposit funds to the `stargateEthVault` and call `stargateRouter.swap`. Will revert if `amountIn > msgValue`.
- `_bridge(amountIn, fromToken, srcPoolId, dstPoolId, payload)` → `IStargateRouter(stargateRouter).swap{value: msg.value}`
  - **What is controllable?** `amountIn`, `fromToken`, `srcPoolId`, `dstPoolId`, and `payload`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** No problem.

Function: `permitSwapAndBridge(address receivedToken, uint16 srcPoolId, uint16 dstPoolId, byte[1] dexId, PermitTransferFrom permit, SignatureTransferDetails transferDetails, byte[] sig, byte[] swapPayload, byte[] bridgePayload)`

The function allows to swap with permit, bridge, and deposit to vaults using Stargate. Add check that `receivedToken` is equal to the last token in swap.

Inputs

- `receivedToken`
  - **Constraints**: N/A.
  - **Impact**: Expected that this token is last in swap.
- `srcPoolId`
  - **Constraints**: `router.swap` reverts if `factory.getPool(_poolId)` returns zero

address.
  - **Impact**: The ID of SRC pool.
- `dstPoolId`
  - **Constraints**: The owner of the router should create and activate the chain path for `_dstChainId` and `_dstPoolId`. The `_dstChainId` is constant `ARBITRUM_CHAIN_ID`.
  - **Impact**: The ID of DST pool.
- `dexId`
  - **Constraints**: Revert if `dexId` is not 0x01, 0x02, 0x03, 0x04, or 0x05.
  - **Impact**: The ID of the DEX that will be used.
- `permit`
  - **Constraints**: Checked inside the `permitTransferFrom` function.
  - **Impact**: The permit data signed over by the owner.
- `transferDetails`
  - **Constraints**: N/A.
  - **Impact**: The spender's requested transfer details for the permitted token.
- `sig`
  - **Constraints**: Checked inside the `permitTransferFrom` function.
  - **Impact**: The signature to verify.
- `swapPayload`
  - **Constraints**: N/A.
  - **Impact**: The data required for swap.
- `bridgePayload`
  - **Constraints**: Is not verified.
  - **Impact**: Contain the data for ZapDest, expected address receiver, `uint256 vaultId`, and address `vaultAddress`.

Branches and code coverage (including function calls)

Intended branches

- `_swapBalancer` is performed properly.
  - ☐ Test coverage
- `_swapUniswapV2` is performed properly.
  - ☑ Test coverage
- `_swapUniswapV3` is performed properly.
  - ☐ Test coverage
- SushiSwap is performed properly.
  - ☐ Test coverage
- `_swapWithCurve` is performed properly.

☐ Test coverage

Negative behavior

- The invalid permit.
  ☐ Negative test
- The invalid signature.
  ☐ Negative test
- The `receivedToken` is not the last token in swap.
  ☐ Negative test
- The `receivedToken` is zero address.
  ☐ Negative test

Function call analysis

- `_swap(dexId,transferDetails.requestedAmount,swapPayload);` → `otherFunction(args)`
  - **What is controllable?** `dexId`, `transferDetails.requestedAmount`, and `swapPayload`.
  - **If return value controllable, how is it used and how can it go wrong?** Returns the final number of tokens after the swap.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Can revert if `dexId` is invalid — also can revert if final amount will be less than minimum out amount.
- `_swapBalancer(swapPayload)` → `balancerVault.call(swapPayload)`
  - **What is controllable?** `swapPayload`.
  - **If return value controllable, how is it used and how can it go wrong?** Returns the final number of tokens after the swap.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Revert if swap failed.
- `_bridge(amountIn, fromToken, srcPoolId, dstPoolId, payload)` → `IStargateRouter(stargateRouterEth).swapETHAndCall{value: msgValue}`
  - **What is controllable?** `amountIn`, `fromToken`, `srcPoolId`, `dstPoolId`, and `payload`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Deposit funds to the `stargateEthVault` and call `stargateRouter.swap`. Will revert if `amountIn > msgValue`.
- `_bridge(amountIn, fromToken, srcPoolId, dstPoolId, payload)` → `IStargateRouter(stargateRouter).swap{value: msg.value}`
  - **What is controllable?** `amountIn`, `fromToken`, `srcPoolId`, `dstPoolId`, and `payl`

oad.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** No problem.

Function: `swapAndBridge(uint256 amountIn, address fromToken, address receivedToken, uint16 srcPoolId, uint16 dstPoolId, byte[1] dexId, byte[] swapPayload, byte[] bridgePayload)`

The same function as `permitSwapAndBridge` but transfers `fromToken` with approve from `msg.sender` instead of permit.

Function: `withdraw(byte[] payload)`

This allows to send a message to withdraw funds from the `ARBITRUM` chain. The message can trigger one of these functions: `withdraw`, `withdrawAndBridge`, or `withdrawSwapAndBridge`.

Inputs

- `payload`
  - **Constraints**: No verifications.
  - **Impact**: It contains all the necessary data for withdrawal: `funcSelector`, `bridgeId`, receiver, ID (the ID for the epoch being withdraw from), and `vaultAddress`. Also can contain data for swap — `swapId`, `toAmountMin`, `dexId`, `toToken`, and `fee` — and for bridging: `maxSlippage` in case of `_bridgeWithCeler`, `maxSlippage` and `bonderFee` in case of `_bridgeWithHop`. The receiver address will be changed to the `msg.sender` address.

Function call analysis

- `ILayerZeroRouter(layerZeroRouter).send`
  - **What is controllable?** `payload`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Can revert in case of unpaid fee.

6 Assessment Results

At the time of our assessment, the reviewed code was deployed to Arbitrum.

During our assessment on the scoped Y2K Finance contracts, we discovered 14 findings. One critical issue was found. Three were of high impact, four were of medium impact, five were of low impact, and the remaining finding was informational in nature. Y2K Finance acknowledged all findings and implemented fixes.

6.1 Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.