PUBLIC

Code Assessment

of the Swaap Earn Protocol Smart Contracts

16 Apr, 2024

Produced for

by

Contents

1 Executive Summary

Dear Swaap Labs,

Thank you for trusting us and giving us the opportunity to audit your project. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Swaap Earn Protocol according to Scope to support you in forming an opinion on their security risks.

Swaap Labs implements asset management system, based on Sommelier Protocol, where Fund contracts can be created to manage the different assets. Funds have limited functionality and rely on adaptors to interact with external protocols. The system is designed to be modular and flexible, allowing for the addition of new adaptors and supported assets.

The most critical subjects covered in our audit are asset solvency, functional correctness, and front-running resistance. Security regarding all the aforementioned subjects is high.

The general subjects covered are system customisation, documentation, and gas efficiency. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	2
• Code Corrected	1
• Acknowledged	1
Low-Severity Findings	4
• Code Corrected	2
• Risk Accepted	1
• Acknowledged	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Swaap Earn Protocol repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	04 Mar 2024	a1510131f5bb1c9fa6c7c3e2b304cc57d2709571	Initial Version
2	15 Apr 2025	98caa6d6be5357479a418fcb23f8c80545f9f8af	Version 2
3	15 Apr 2025	171a871390b59c413bfc3541b23bca9a446cf45c	Version 3

For the solidity smart contracts, the compiler version `0.8.21` was chosen.

The contracts in scope are:

Core

- src/base/ERC4626.sol
- src/base/Fund.sol
- src/base/permutations/FundWithShareLockFlashLoansWhitelisting.sol
- src/base/permutations/FundWithShareLockPeriod.sol
- src/Registry.sol
- src/Deployer.sol

Adaptors

- src/modules/adaptors/Aave/V3/AaveV3AccountExtension.sol
- src/modules/adaptors/Aave/V3/AaveV3AccountHelper.sol
- src/modules/adaptors/Aave/V3/AaveV3ATokenManagerAdaptor.sol
- src/modules/adaptors/Aave/V3/AaveV3DebtManagerAdaptor.sol
- src/modules/adaptors/AggregatorBaseAdaptor.sol
- src/modules/adaptors/BaseAdaptor.sol
- src/modules/adaptors/ERC20Adaptor.sol
- src/modules/adaptors/PositionlessAdaptor.sol
- src/modules/adaptors/Paraswap/ParaswapAdaptor.sol
- src/modules/adaptors/Swaap/SwaapV2Adaptor.sol
- src/modules/adaptors/Swaap/SwaapFundAdaptor.sol
- src/modules/adaptors/Balancer/BalancerFlashLoanAdaptor.sol
- src/modules/adaptors/Balancer/BalancerFlashLoanHelper.sol

Utils

- src/utils/Math.sol

- src/utils/SigUtils.sol
- src/utils/Uint32Array.sol

Price router

- src/modules/price-router/PriceRouter.sol
- src/modules/price-router/Extensions/Extension.sol
- src/modules/price-router/Extensions/Swaap/SwaapSafeguardPoolExtension.sol
- src/modules/price-router/Extensions/Balancer/BalancerPoolExtension.sol
- src/modules/price-router/Extensions/ERC4626Extension.sol

Fund Fees

- src/modules/fees/FeesManager.sol
- src/modules/fees/ManagementFeesLib.sol
- src/modules/fees/PerformanceFeesLib.sol

2.1.1 Excluded from scope

All the contracts not mentioned in scope are considered out-of-scope. External dependencies and libraries are out-of-scope as well. Tests and deployment scripts are out-of-scope.

2.1.2 Assumptions

The following assumptions were made during the assessment:

- The system will be correctly initialized and configured. The configuration is not part of this assessment.
- All privileged parties are trusted, non-malicious, and controlled by Swaap Labs.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Swaap Labs offers on-chain system, based on Sommelier Protocol, that allows management of the funds and assets in the DeFi ecosystem.

Overall, the system consists of the following smart contracts:

- `Adaptors` - together with configuration data each `Adaptor` contract represents a position in the system. Each position is bind to certain assets. Position can be of a credit or debt type.
- `Fund` - represents the fund in the system, responsible for the management of the positions by the owner.
- `Registry` - central storage for the permissions and addresses of the other smart contracts and positions.
- `PriceRouter` - responsible for pricing and evaluation the assets in the system.
- `Extensions` - responsible for the custom pricing logic of the assets in the system.

- `FeeManager` - responsible for the calculation of the fees in the system.

More detailed description of the contracts is provided below.

2.2.1 Registry

`Registry` smart contract can be seen as a central storage for the permissions and addresses of the other smart contracts in the system.

To sum up, the functionality of the contract are:

- Associate the addresses of the other smart contracts with the IDs. System-critical contracts have predefined IDs, while the owner adds new contracts with custom IDs. Protocol DAO is stored at ID 0, and only the protocol DAO itself can change the address stored at ID 0
- Trust and distrust the adaptors used by the `Fund` contract.
- Trust and distrust positions. While fully set up position has `adaptorData` and `configurationData`, the `Registry` trust process sets only the `adaptorData`. `configurationData` is set later in the `Fund` contract.
- Assign depositors that can deposit on behalf of other users in the `approvedForDepositOnBehalf` mapping.
- Pause and unpause specific addresses in the system. Fund contracts can be paused this way.
- Set and track the maximum trade volume that `AggregatorBasedAdaptor` (extension of `Adaptor`) contracts perform. Reverts if the volume exceeds the `maxVolumeInUSD` for the current period.

2.2.2 Fund

The `Fund` is the core contract of the protocol. It inherits the `ERC4626`, `Ownable` and `ERC20` contracts' functionalities. It manages its funds through a variety of positions. The fund allows its owner to rebalance positions, and the end users to deposit and withdraw from the fund.

Positions

Positions are used to track the assets in the fund. A position consists of the following:

- An adaptor
- The adaptor data of the adaptor
- Whether the position is a debit or a credit position
- Some configuration data

The following functions are used by the fund owner to manage positions:

- `addPosition(..)` adds a new position to the fund. All the data necessary to create the position (except for the `configurationData`) is retrieved from the registry.
- `removePosition(..)` removes a position with zero balance from the fund.
- `forcePositionOut(..)` removes a position distrusted position from the fund (balance can be positive).
- `swapPositions(..)` allows the owner of the fund to swap two positions in the position array.

Depositing and Withdrawing

Each fund has a base asset and a `holdingPosition` associated with it. The base asset is used to calculate the value of the fund. The `holdingPosition` is the default position, where the newly deposited assets are stored.

The following entry points allow users to interact with the fund:

- `deposit(assets, receiver)` can called by a user to deposit a specified amount of base asset to the fund. The user will receive shares in exchange for the deposited assets. If sender is not the receiver, `Registry.approvedForDepositOnBehalf` must be set.
- `mint(shares, receiver)` same as `deposit(..)`, but the user specifies the amount of shares to mint.
- `withdraw(assets, receiver, owner)` is called by a user to withdraw specified amount of base asset from the fund. The shares of the owner will be burned in exchange for the withdrawn assets. Approval from owner to sender will be lowered, if necessary.
- `redeem(shares, receiver, owner)` same as `withdraw(..)`, but the user specifies the amount of shares to redeem.

Fund management

Dedicated `callOnAdaptor(..)` function in `Fund` contract allows calls to the adaptor contracts. The function can be called either by the owner or by dedicated `AutomationActions` contract, whose address can be set by the owner. `callOnAdaptor()` performs delegate calls to the adaptors from `Fund.adaptorCatalogue`. Owner can add adaptors trusted by the `Registry` to the `adaptorCatalogue`. The value of Fund's assets after the call should not deviate more than `allowedRebalanceDeviation` from the value before the call. The default value of `allowedRebalanceDeviation` is 0.03%. The owner can change it via `setRebalanceDeviation()`.

Pausability and Emergency

`Registry` contract can pause the fund. During the pause, the `deposit()`, `withdraw()` and `callOnAdaptor()` functions are disabled. However, after `endPauseTimestamp` the pausing will be automatically lifted and contract cannot be paused again. The `endPauseTimestamp` is set during contract deployment and is 9 months (9 * 30 days) after the deployment by default. `initiateShutdown()` and `liftShutdown()` can be used by the owner of the fund to initiate or lift a shutdown. `withdraw()` and `redeem()` functions are still available.

`initiateShutdown()` and `liftShutdown()` can be used by the owner of the fund to initiate or lift a shutdown. During the shutdown, the `deposit()` and `callOnAdaptor()` functions are disabled. `withdraw()` and `redeem()` functions are still available.

Permutations

The fund is extended by the following permutations:

- `FundWithShareLockPeriod`: adds a lock that starts at deposit time and prevents the owner of the shares from withdrawing before a certain lock period is over.
- `FundWithShareLockFlashLoansWhitelisting`: implements the same functionalities of `FundWithShareLockPeriod`, in addition, only allows whitelisted users to deposit into the fund.

2.2.3 FeesManager

`FeesManager` smart contract is responsible for the calculation of the fees in the system, and their distribution to the payout addresses set by the fund owner. Every time the `deposit()` or `withdraw()` function is called on a Fund, the `FeeManager`:

- Provides exit/enter fees as % of the amount.
- Calculates the shares as fees that need to be minted to the `FeeManager`.

The following fees exits in the system:

- enter/exit fees - fees that are charged when a user deposits or withdraws assets from the fund. They are set as a percentage of the deposited/withdrawn amount.

- `managementFee` - fee that is periodically minted as shares. It only depends on totalSupply and the time since the management fee was last claimed. This fee is computed using the following formula: SwaapV2:

```
SF = TS * (e^(a*dT) - 1)
a = - (ln(1 - f) / 1 year)
```

  `SF` are the shares to be minted as fees, `TS` is the totalSupply of the fund, `dT` is the time elapsed since the last claim, and `f` is the yearly management fees percentage. The yearly management fees percentage `f` is the percentage of the pool ownership that LPs will lose in a year due to the application of the fee (e.g. 5%).

- `performanceFee` (high-watermark fee) - fee that is periodically minted as shares. Based on a "high-water mark": the highest share price that the fund has reached so far. Performance fees only gets paid if the fund price goes above the high-water mark. If the price of the fund doesn't reach the high-water mark, then no performance fee is paid. The performance fee is based on the difference between the current price of the fund and the high-water mark. Watermark price can be reset by the Registry owner if It has been more than 30 days since last reset OR `totalAssets` have increased by at least 50% since last mark.

2.2.4 PriceRouter

`PriceRouter` smart contract is responsible for pricing and evaluation the assets in the system. The owner of the contract can add new assets or edit the settings of existing assets.

The edit process is two-step: first, the owner need to call the `startEditAsset()` function to start the edit process, and after 7 days `completeEditAsset()` function to update the asset settings. Hash of the new settings is calculated and stored in the contract, thus the owner can't complete the edit process with different settings. The pending edit can be canceled by the owner via `cancelEditAsset()`

Main functionality of the contract is represented by following functions:

- `getPriceInUSD(..)` - returns the price of the asset in USD. Has a multi-asset version `getPricesInUSD(..)`.
- `getValue(..)` - returns the value of base amount of asset in quote asset. Has a multi-asset version `getValues(..)`.
- `getValuesDelta(..)` - helper function, that basically computes difference between two `getValues(..)` calls.
- `getExchangeRate(..)` - returns the exchange rate between two assets, in decimals of quote asset. Has a multi-asset version `getExchangeRates(..)`.

The functionality of all these functions relies on `_getPriceInUSD(..)` function. The `_getPriceInUSD(asset, settings)` based on `settings.derivative` field can call different functions to calculate the price of the asset. The `settings.derivative` field can be one of the following values:

- 1 - for the assets that are priced using Chainlink oracles.
- 2 - for the assets that are priced using TWAP (Uniswap V3-like) oracles.
- 3 - for the assets that are priced using Extension contracts.

2.2.5 Extensions

`Extensions` are used to extend the functionality of the `PriceRouter` contract. Any extension must implement two functions:

- `setupSource()` - function that is called by the `PriceRouter` to set up the source of the asset.
- `getPriceInUSD()` - function that returns the price of the asset in USD.

Particular non-abstract extensions that were part of the scope are:

- `ERC4626Extension` - extension that allows to price ERC4626 tokens.
- `SwaapSafeguardPoolExtension` - extension that allows to price Swaap Safeguard Pool tokens.

2.2.6 Adaptors

Each position in the Fund has an associated adaptor. Adaptors are purpose-specific contracts that handle specific use cases in an external protocol.

Adaptors are stateless, and their behavior depends on the arguments that are passed as input when calling their functions. To rebalance a positions, strategists make delegate calls from the fund into the position's adaptor.

Adaptors implement the following functions:

- `identifier` returns the unique identifier of the adaptor.
- `deposit` deposits into a position.
- `withdraw` withdraws from a position.
- `assetOf` returns the underlying asset of a position.
- `balanceOf` return the balance of a position.
- `isDebt` returns whether or not the position is a debit position.

System has a special `PositionlessAdaptor` abstract adaptor, that cannot be associated with a position in the fund, but can be used by `callOnAdaptor()`. It reverts or return zero in all default functions except for `identifier` and `isDebt`, but can define custom functions and is meant to be extended. `AggregatorBaseAdaptor` is an example of such an extension. `AggregatorBaseAdaptor` allows swapping an amount of `tokenIn` for a target amount of `tokenOut` in an arbitrary external swap contract It checks the slippage and reports the exchanged volume to `Registry`. `Registry` can limit this volume.

Following non-abstract adaptors were part of the scope:

ERC20Adaptor

Adaptor that handles the deposit and withdrawal of ERC20 tokens. Does not have any custom functions accessible via `callOnAdaptor()`.

SwaapV2Adaptor

Adaptor that allows to interact with SwaapV2 pools (similar to Balancer V2). It has custom functions accessible via `callOnAdaptor()`:

- `joinPool` joins a target SwaapV2 pool with an array of tokens and a maximum amount for each of them.
- `joinPoolWithAllowlistOn` allows to join a pool if allowlist is enabled; a deadline, and signature data must be provided.
- `exitPool` exits a target SwaapV2 pool by burning SPT and receiving the tokens specified. The tokens received are tracked in the Fund via a position.
- `makeFlashLoan` allows executing a flashloan from Balancer V2. The balancer vault will call `receiveFlashLoan` on the fund, therefore, the fund `FundWithBalancerFlashLoans` or `fundwithsharelockflashloanswhitelisting` must be used.

SwaapFundAdaptor

Adaptor that allows the fund manager to deposit funds into another `Fund` contract. The adaptor has the following functions accessible via `callOnAdaptor()`:

- `depositToFund()` - deposits assets into another fund.

- `withdrawFromFund()` - withdraws assets from another fund.

BalancerFlashLoanAdaptor

Adaptor that allows the fund to initiate flash loans from the Balancer vault, via `makeFlashLoan` function.

ParaswapAdaptor

Positionless adaptor based on `AggregatorBaseAdaptor`, that can assets in the Paraswap contract pool. The `swapWithParaswap` function is accessible via `callOnAdaptor()`.

AaveV3ATokenManagerAdaptor

Adaptor that can supply and exit from Aave pools. Compared to most adaptors, the Funds does not directly interact with the Aave pool. Instead, a special `AaveV3AccountExtension` contract is deployed. This way, the fund can hold multiple positions in the Aave pool with different eModes. The account id is the eMode of the position.

Every time the `deposit()` is called, the adaptor checks if `AaveV3AccountExtension` contract is deployed. If not, it deploys it. Each account is identified in the Fund by its account id.

In Aave, every pair (pool, accountAddress) has a corresponding minimum health factor. Fund manager can configure the minimum health factor for all its accounts by setting the configuration data. However, it cannot go below the minimum health factor that is set by the deployer of the `AaveV3ATokenManagerAdaptor`

The following functions are accessible via `callOnAdaptor()`:

- `depositToAave()` - deposits assets into the Aave pool.
- `withdrawFromAave()` - withdraws assets from the Aave pool.
- `adjustIsolationModeAssetAsCollateral()` - adjusts the isolation mode of the asset as collateral.
- `createAccountExtension()` - creates a new account extension in the Aave pool.

AaveV3DebtManagerAdaptor

Adaptor that allows the fund to lend and repay in an AaveV3 pool.

The following functions are accessible via `callOnAdaptor()`:

- `borrowFromAave` allows the fund owner to borrow assets from the pool.
- `borrowFromAave` allows the fund owner to repay loans to the pool.
- `repayWithATokens` allows the fund owner to repay loans (of `tokens`) to the pool using `aTokens`.
- `flashLoan` allows the fund owner to take a flash loan.

2.2.7 Ownership model

`PriceRouter` and `Registry` have similar ownership model. Each of them inherits `Ownable` contract from OpenZeppelin library. However, in addition to default `transferOwnership()` function, the contract has its owner change process. Protocol DAO address, stored in `Registry`, can change the owner in a two-step process: first, the DAO needs to call the `transitionOwner()` function to start the change process and after 7 days new owner needs to call `completeTransition()` function to update the owner. Old owner immediately loses access to privileged functions after the `transitionOwner()` function is called. Pending owner can be canceled by the DAO via `cancelTransition()` function.

`Fund` has default OpenZeppelin `Ownable` contract functionality.

2.3 Changes in Version 3

`ParaswapAdaptor` was removed from the system. Instead, the `AggregatorBaseAdaptor` was renamed to `AggregatorAdaptor` and extended with the `swapWithAggregator` function. Owner of `Registry` contract for any given adaptor can set spender address in `aggregatorSpender` mapping. `AggregatorAdaptor.swapWithAggregator` gets adaptor address as argument, checks that `Registry` has set spender address for the adaptor, gives spender permission to spend the amount of `tokenIn` and calls the adaptor with arbitrary data.

`Registry` owner can use `approvedFlashLoanSource` to whitelist the flash loan sources. `FundWithShareLockFlashLoansWhitelisting` checks that msg.sender that calls `receiveFlashLoan()` is in the whitelist.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1

- Arbitrage Opportunities Acknowledged

Low-Severity Findings	2

- Call to applyFeesBeforeJoinExit Can Burn All Gas Risk Accepted
- Donation Can Break Performance Fees Acknowledged

5.1 Arbitrage Opportunities

Security Medium Version 1 Acknowledged

CS-SWAAP-EARN-013

Arbitrage opportunities due to price update could arise if the default implementation of Fund (`Fund.sol`) was deployed with deposit/withdraw fees that are low. In this case, an attacker could deposit funds, observe a price change transaction, and withdraw the funds to profit from the price change. This can happen if an arbitrageur observes a price change transaction and sandwich it with a deposit and a withdraw transaction.

Note that such opportunities might be discouraged by the application of enter and exit fees on deposit and withdraw.

Finally, it is important to highlight that the `Fund` variants `FundWithShareLockPeriod.sol` and `FundWithShareLockFlashLoansWhitelisting.sol` offer have an additional layer of protection against this attack vector. The shares of the fund are locked for a certain period of time after a deposit. This prevents the attacker from withdrawing the funds immediately after a deposit.

Acknowledged:

The issue is acknowledged. Swaap will only use the `FundWithShareLockFlashLoansWhitelisting.sol` variation.

5.2 Call to `applyFeesBeforeJoinExit` Can Burn All Gas

Design Low Version 1 Risk Accepted

The call to `FEES_MANAGER.applyFeesBeforeJoinExit` is done using try catch, so if the call fails, the flow with continue without fees. However, in some case the call can cause EVM-level panic and burn all gas, so there will be no gas left to continue the flow.

Risk accepted:

Swaap Labs has acknowledged the issue and has decided to keep the code unchanged in Version 2. In case the FeeManager panics, the Fund can be shut down, allowing LPs to withdraw without calling the FeesManager.

5.3 Donation Can Break Performance Fees

Security | Low | Version 1 | Acknowledged

Just after the deployment, by default Fund is required to have at least `10^4` of assets and `10^4 * 10^ (_FUND_DECIMALS - _ASSET_DECIMALS)` of shares. The `FeeManager` stores the `highWaterMarkPrice` as type(uint72) with 18 decimals precision ~ 4.72e21. When the first deposit is made, the `highWaterMarkPrice` is set to the new value. However, an attacker can donate a small amount of assets to the fund via token transfer and make the `highWaterMarkPrice` more than `type(uint72).max`. This way the performance fee will be broken and the Fund will keep operating without fees. Inflating initial `10^4` of shares for 18 decimal asset won't require a lot of value for some tokens.

Swaap Labs has acknowledged the issues and has decided to keep the code unchanged in Version 2. However, to tackle any possible risk, Swaap Labs will initiate the Fund with at least 1$ worth of assets such that the attack would be expensive to perform.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1

- Fees Calculation Discrepance Code Corrected

Low-Severity Findings	2

- Deployer Cannot Accept Ether Code Corrected
- resetHighWaterMark() Ignores Pending Fees Code Corrected

Informational Findings	4

- Specifications Problems Code Corrected
- Events Code Corrected
- A Fund Cannot Take Flashloans From Both Balancer and Swaap Code Corrected
- Gas Efficiency Code Corrected

6.1 Fees Calculation Discrepance

Design Medium Version 1 Code Corrected

CS-SWAAP-EARN-001

The enter and exit fees are computed in a way that introduces discrepancies in the system.

1. `maxDeposit` returns an incorrect amount compared to the maximum amount that will be possible to actually withdraw using `withdraw`.
2. When depositing, `mint` will be a cheaper option compared to `deposit`.

Example of 1.

Assume that the Fund that has 1000 shares and 1000 assets. Also, it has 1100 shares cap and a 20% fee. No fee shares pending to be minted.

When user calls `maxDeposit()`, the function will return:

100 shares * 1000 assets / 1000 shares * (100% + 20%) / 100% = 120 assets

When user calls `deposit()` with 120 assets, the function will mint:

120 assets * 1000 shares / 1000 assets * (100% - 20%) / 100% = 96 shares

The difference is 4 shares. Calling deposit with 125 assets will mint 100 shares, which is what `maxDeposit()` value. The discrepancy is due to how the fee is calculated in `_applyEnterOrExitFees()` function. In one case, `(100% + fee)/100%` is used, and in the other, `(100% - fee)/100%` is used. But the one is not the inverse of the other.

Example of 2.

A 50% entrance fee will increase the number of assets needed by `mint` by 1.5 times, while decrease the number of minted shares by 2 in `deposit`. Thus `mint()` will effectively be cheaper.

Code corrected: Swaap Labs has corrected the code in Version 2 by changing the way the fees are applied. Now, when entering the fund, the number of assets is virtually increased by the fee, and when exiting, the number of assets is decreased by the fee. This way the discrepancy is removed.

6.2 `Deployer` Cannot Accept Ether

Design | Low | Version 1 | Code Corrected

CS-SWAAP-EARN-004

Deployer has `deployContract()` function that deploys a contract using CREATE3 library. This function accepts a `value` parameter that is supposed to be transferred to the deployed contract. The library function `CREATE3.deploy(salt, createCode, value)` will try to transfer `value` of ether together with the deployment transaction. However, `deployContract()` function does not accept Ether and `Deployer` contract does not have any other way to receive Ether. This limits the functionality of the `deployContract()` function and complicates the deployment process if ether transfer is required.

Code corrected: Swaap Labs has corrected the code by making the function `deployContract` `payable` and by passing `msg.value` to the CREATE3 library.

6.3 `resetHighWaterMark()` Ignores Pending Fees

Design | Low | Version 1 | Code Corrected

CS-SWAAP-EARN-005

The `resetHighWaterMark()` function can be called by Registry owner to reset the high-water mark. However, the function does not take into account the fees that would have been minted if `collectFees()` was called before computing the new mark. When mark is reset and fees are minted afterward, the high water mark will be higher than the new price of shares.

Code corrected:

Swaap Labs has corrected the code in Version 2 by collecting the management and performance fees via `collectFees()`, before resetting the high-watermark price.

6.4 A Fund Cannot Take Flashloans From Both Balancer and Swaap

Informational | Version 1 | Code Corrected

CS-SWAAP-EARN-012

A fund can request a flash loan from Balancer or Swaap thanks to the `SwaapV2Adaptor` and `BalancerFlashLoanAdaptor` adaptors.

However, it is worth noting that the `receiveFlashLoan()` function in `FundWithBalancerFlashLoans` will only be callable by the immutable address `balancerVault`. Therefore, the contract `FundWithBalancerFlashLoans` will have to be configured at deployment to either receive flash loans from Balancer OR from Swaap.

Code corrected:

The `Registry` contract now holds a mapping `approvedFlashLoanSource`. `FundWithBalancerFlashLoans` verifies that `msg.sender` is the approved flash loan source before executing `receiveFlashLoan()`.

6.5 Events

Informational | Version 1 | Code Corrected

CS-SWAAP-EARN-007

The following events could be improved:

- `Deployer.ContractDeployed(name, contractAddress, creationCodeHash)` name could be indexed.
- `Registry.DepositorOnBehalfChanged(depositor, state)` depositor could be indexed.
- `Registry.TargetPaused(target)` and `Registry.TargetUnpaused(target)` target could be indexed.
- `Fund.AdaptorCalled(adaptor, data)` could have adaptor indexed.

Code corrected:

Swaap Labs has corrected the code in Version 2.

6.6 Gas Efficiency

Informational | Version 1 | Code Corrected

CS-SWAAP-EARN-008

The following code could be optimized to consume less gas:

Fund

1. `cachePriceRouter`: computing `assetsAfter` does not serve any purpose unless `checkTotalAssets` is enabled.
2. `maxdeposit`: `sharesupplycap` is read twice from storage and assigned to `_cap`.

PriceRouter

1. `completeTransition`: the requirement that `pendingOwner != 0` is redundant.
2. `getValue`, `_getValues`, `getExchangeRate` and `getExchangeRates`: all redundantly check that the derivative is non-zero; the check is already performed in `_getPriceInUSD`.

Registry

1. `completeTransition`: the requirement that `pendingOwner != 0` is redundant, since `msg.sender == pendingOwner`.

2. `setMaxAllowedAdaptorVolumeParams`: some of the variables in the event could be replaced with equivalent stack variables.

AaveV3ATokenManagerAdaptor

1. `depositToAave`: when revoking approval, the conversion of `_revokeExternalApproval` to ERC20 is redundant.

Code corrected:

Swaap Labs has corrected the code in Version 2.

6.7 Specifications Problems

Informational Version 1 Code Corrected

CS-SWAAP-EARN-010

The specifications of the codebase present some issues:

Fund

1. `deposit` and `redeem` have outdated comment "Check for rounding error since we round down in previewDeposit.".
2. `_MAX_POSITIONS` has a typo in the natspec.
3. `_DELAY_UNTIL_END_PAUSE` has a typo in the natspec.
4. `cachePriceRouter` has a typo in the natspec.
5. Swaap Governance and Swaap Strategist used interchangeably when `onlyOwner` modifier is used.

PriceRouter

1. `_updateAsset` has a missing parameter in the natspec.

Registry

1. `setMaxAllowedAdaptorVolumeParams` has an incorrect natspec for the parameter `resetVolume`.
2. `checkAndUpdateFundTradeVolume` has incorrect natspec.

BaseAdaptor #. `_verifyConstructorMinimumHealthFactor` has incorrect natspec.

AaveV3DebtManagerAdaptor #. Contract natspec "Adaptor Data Specification" is incorrect.

AaveV3ATokenManagerAdaptor #. In the contract natspec, the statement "Funds with multiple aToken positions MUST only specify minimum health factor on ONE of the positions" is outdated.

Code corrected:

Swaap Labs has corrected all the issues mentioned above in Version 2.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Circular Dependencies

Informational Version 1 Acknowledged

CS-SWAAP-EARN-006

It is possible for a Swaap fund to have a position in another Swaap fund. This means that funds might be configured such that a circular dependency is created. This might break the price evaluation, due to a loop.

Circular dependencies should be avoided, however, when adding a new position into a fund, there are no mechanisms in place to prevent this issue.

Acknowledged:

Swaap Labs has acknowledged the potential risks and has decided to leave the code unchanged in Version 2.

7.2 Output Token From Swaap Fund Positions

Informational Version 1 Acknowledged

CS-SWAAP-EARN-009

A fund can have a position in another Swaap Fund. Let's call the fund with the position `fund1` and the fund as asset - `fund2`.

However, when withdrawing, the function `SwaapFundAdaptor.withdraw()` does not check that the assets the `fund2` can transfer to `fund1` are tracked. This might might cause big slippage of share prices of `fund1`.

Acknowledged:

Swaap Labs has acknowledged this limitation and will pay additional attention when adding such positions.

7.3 Withdrawal Priority

Informational Version 1 Acknowledged

CS-SWAAP-EARN-011

The `Fund` stores withdrawable assets in `creditAssets` array. During withdrawal, the assets are withdrawn from the `creditAssets` array in the order of their list index, until the needed amount is withdrawn. This makes that assets with higher indices less likely to be withdrawn. Fund owners must be aware of this behavior and adjust the asset list when needed.

Acknowledged:

Swaap Labs is aware of this behavior and intends to support Funds with properly arranged assets.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Manipulating the Price of SwaapSafeguardPools

Note Version 1

While highly unlikely, it is possible that the asset composition `SwaapSafeguardPool` can be manipulated by 3rd parties, due to AMM nature of the pool Thus, the price returned by the price extension `SwaapSafeguardPools` could be subject to manipulation

8.2 Staking Reward Tokens

Note Version 1

Certain protocols like `AAVE` and `COMP` have a staking reward token that is not the same as the underlying token. Fund might need a special adapter to handle these rewards.