



AirPuff

SMART CONTRACT AUDIT



April 10th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passed a security audit.



ZOKYO AUDIT SCORING AIRPUFF

1. Severity of Issues:

- Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
- High: Important issues that can compromise the contract in certain scenarios.
- Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
- Low: Smaller issues that might not pose security risks but are still noteworthy.
- Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.

2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.

3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.

4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.

5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.

6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:

- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:

- 0 Critical issues: 0 points deducted
- 3 High issues: 3 resolved: 2 points deducted for the concern regarding the inconsistent implementation of additional anti-frontrun measures (in a form of modifier for restrictions for the user-only calls)
- 0 Medium issues: 0 points deducted
- 1 Low issues: 1 resolved = 0 points deducted
- 4 Informational issues: 1 resolved, 1 verified, 1 unresolved, 1 acknowledged = 2 points deducted for the best practices and documentation violation

Thus, $100 - 4 = 96$ points out of 100.

TECHNICAL SUMMARY

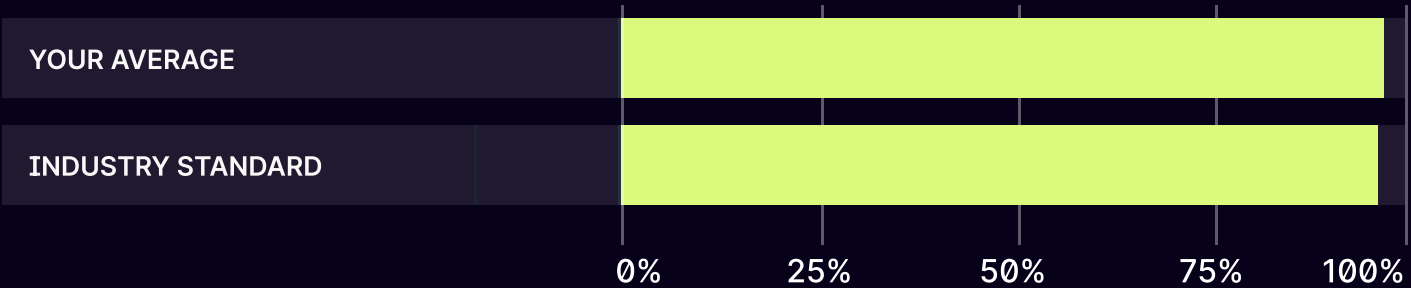
This document outlines the overall security of the AirPuff smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the AirPuff smart contract codebase for quality, security, and correctness.

Contract Status



Testable Code



Corresponds to the industry standards.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network’s fast-paced and rapidly changing environment, we recommend that the AirPuff team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

Auditing Strategy and Techniques Applied	6
Executive Summary	7
Protocol overview	8
Protocol Description	20
Roles and responsibilities	21
Settings	23
Deployment	24
Structure and Organization of Document	25
Complete Analysis	26
Code Coverage and Test Results for all files written by Zokyo Security	32

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the AirPuff repository:

<https://github.com/Airpuff/AirPuff-Contracts/tree/main>

Branch: main

Initial commit: 966424e7b95f5cdf99dd9115b588bf4be8b95ad2

Final commit: 24e0d8dc888e1e7d5b7775a75bb201c9dbc60694

Within the scope of this audit, the auditors reviewed the following contract(s):

- AirPuff1X
- AirPuffHandlerM
- AirPuffVaultM

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of AirPuff smart contract. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contract logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

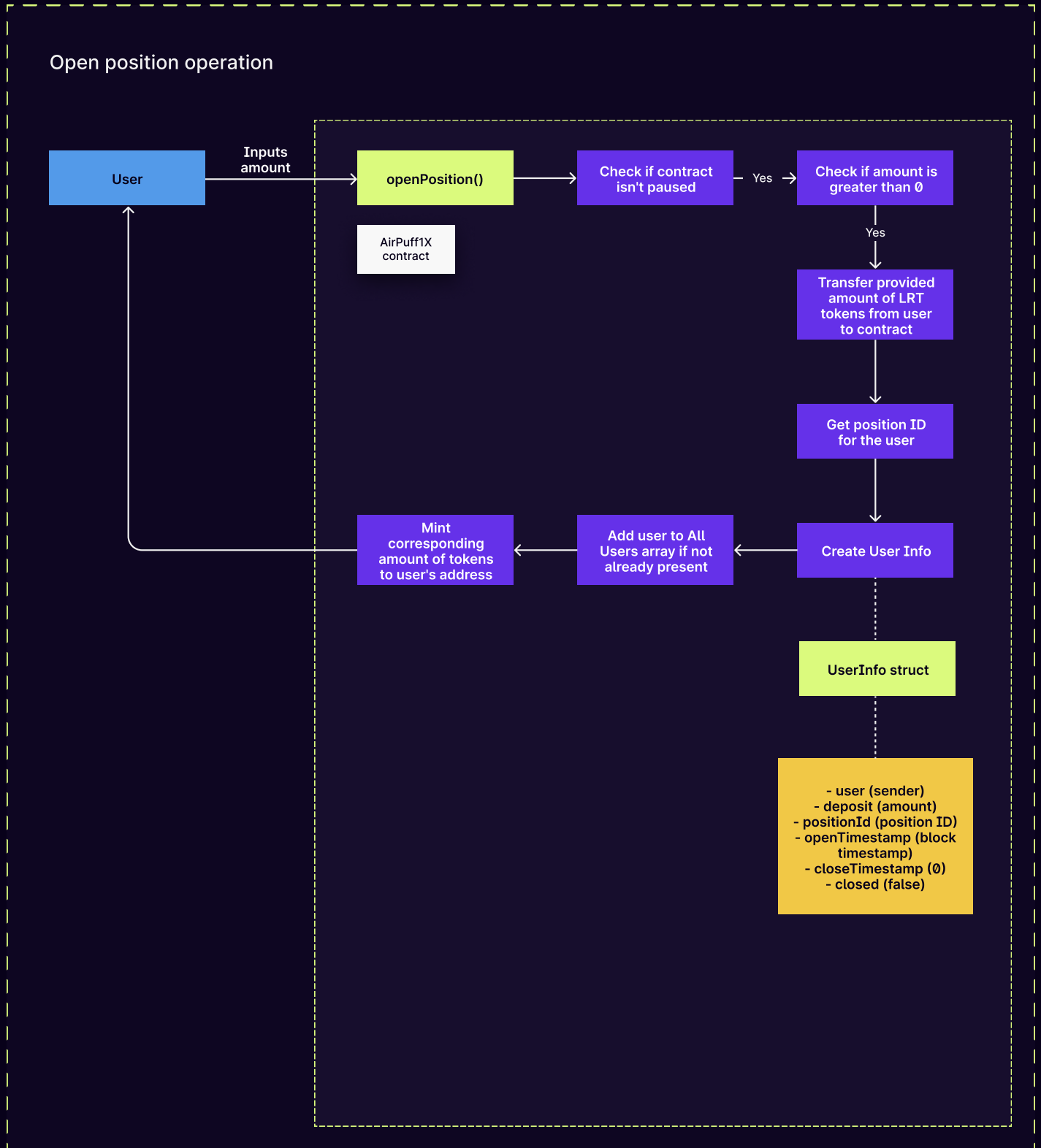
As part of the second audit iteration, the Zokyo Security team conducted a comprehensive AirPuff protocol audit covering all aspects, including lending, handling, and vault contracts.

The audit's goal was to verify the correctness of vault implementation, the connection between contracts, and all main functionality (lend, repay, open, liquidate, and close position) and to perform the check that the tokens on the contract were safe. The auditors reviewed the code line by line, compared it against several vulnerable areas and checklists of vulnerabilities (including the standard one), validated the business logic of the contracts, and ensured that best practices in terms of gas spending were applied. The setup and deployment scripts of the contracts were also carefully audited.

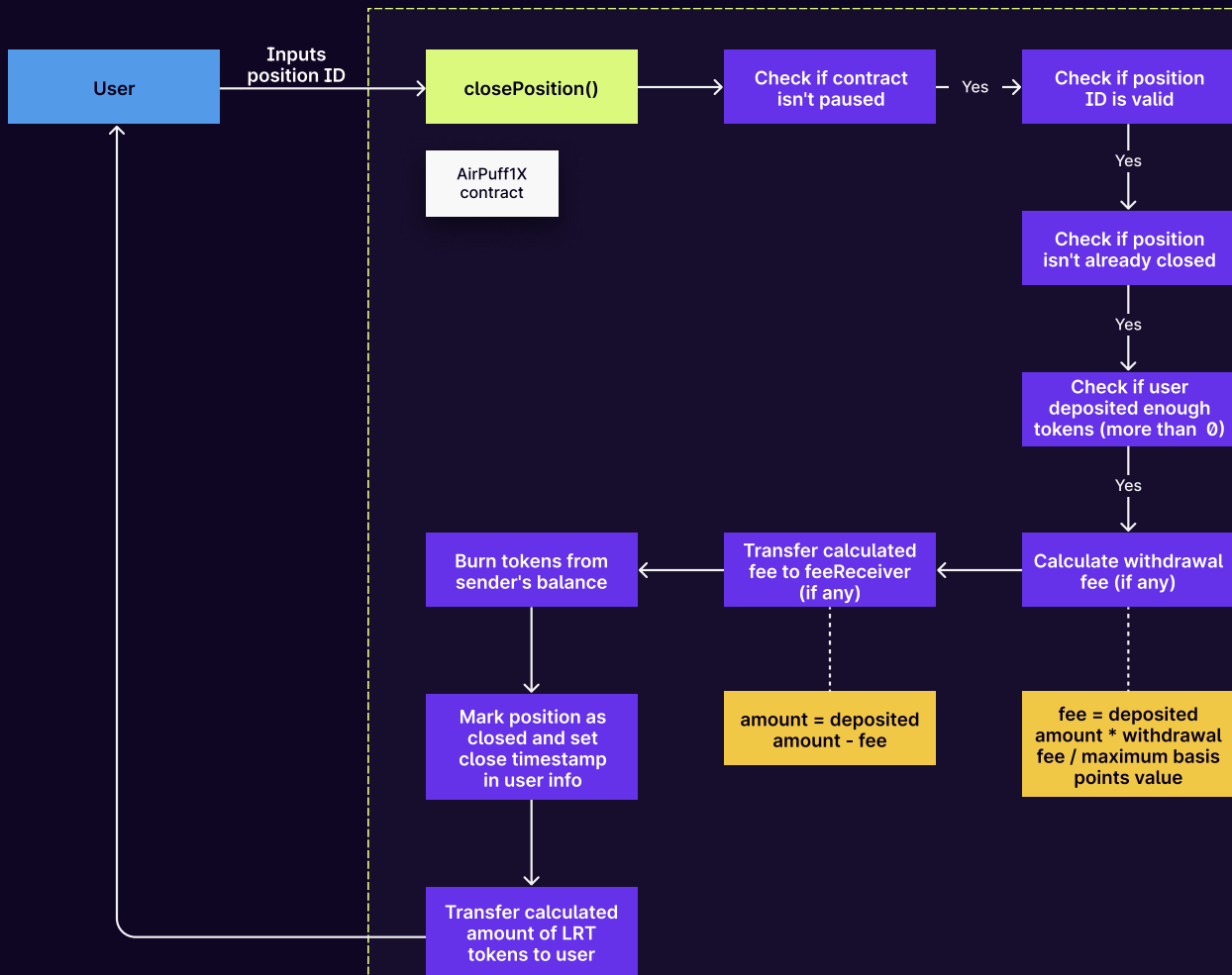
During the manual portion of the audit, three high-risk issues were discovered regarding quoter uni swap integration, possible front-run attack, and explicit tokens burn. The AirPuff team resolved all the issues by removing the quoter, calculating the price and minimum swap amountOut based on the oracles data, and disabling explicit tokens burn. Although calculating the swap value of the output amount from oracles data should prevent frontrun attacks, the security team highly recommends an additional measure: adding a modifier to restrict function calls to EOAs only. Other low-level and informational severity issues were related to the checks logic, parameters off-chain determination, and lack of events emitting. The AirPuff team resolved and verified most of these issues, although others have been marked as unresolved and acknowledged.

Another part of the audit process involved tests suite preparation. Zokyo Security prepared its own set of unit tests and additional scenarios to cover all main user flows and functionalities. The complete set of unit tests can be found in the Code Coverage and Test Result sections.

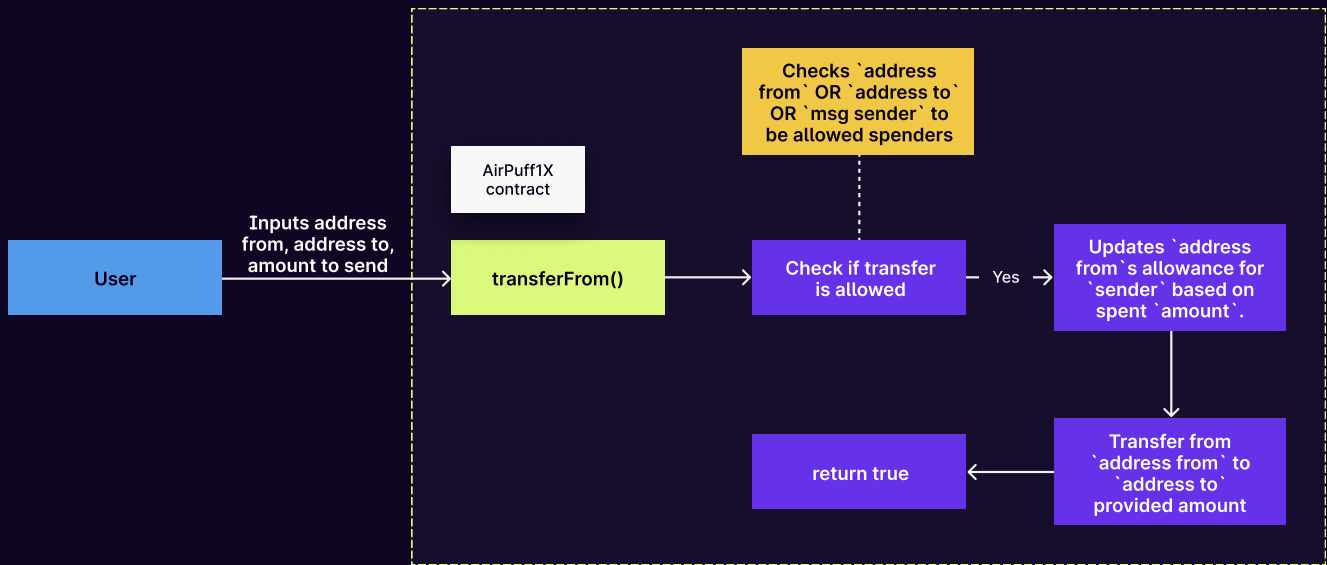
Overall, the security level of the contracts is high. The contract is well-written, with informative code comments. Nevertheless, the contract fails to check against the backdoors, as it is upgradeable. Despite its common approach, it still creates a controllable backdoor, which should be noted in the funds-bearing contract. From other points of view, the protocol functions as expected.



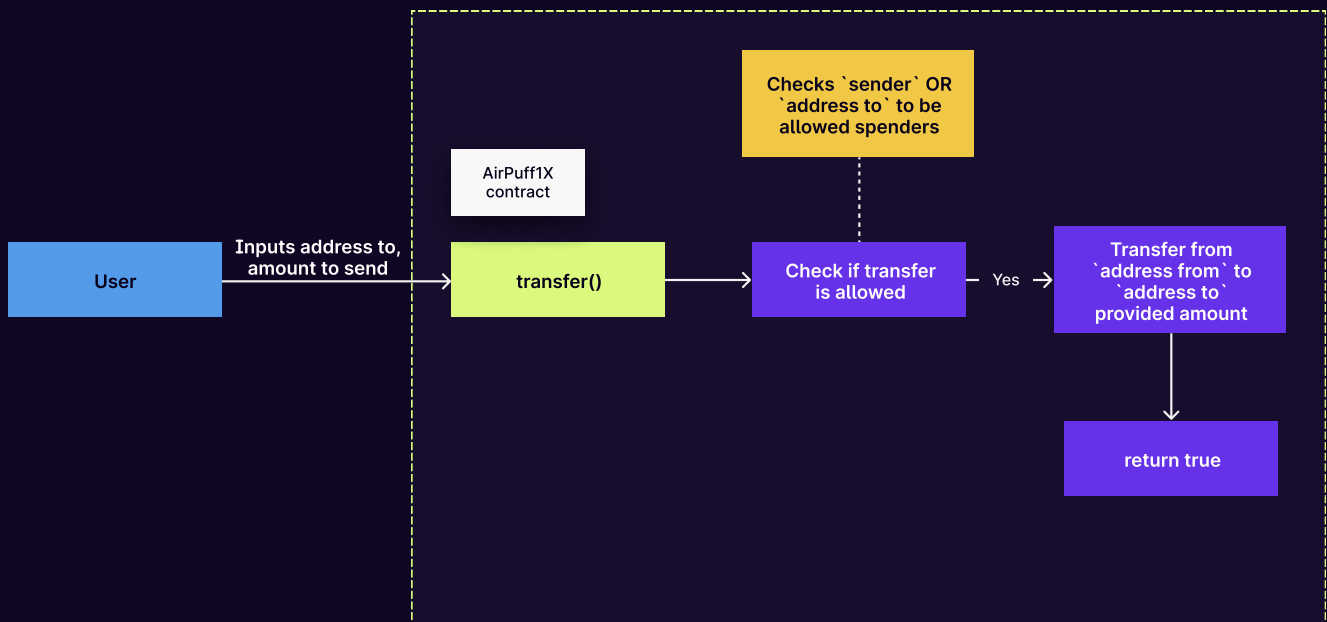
Close position operation



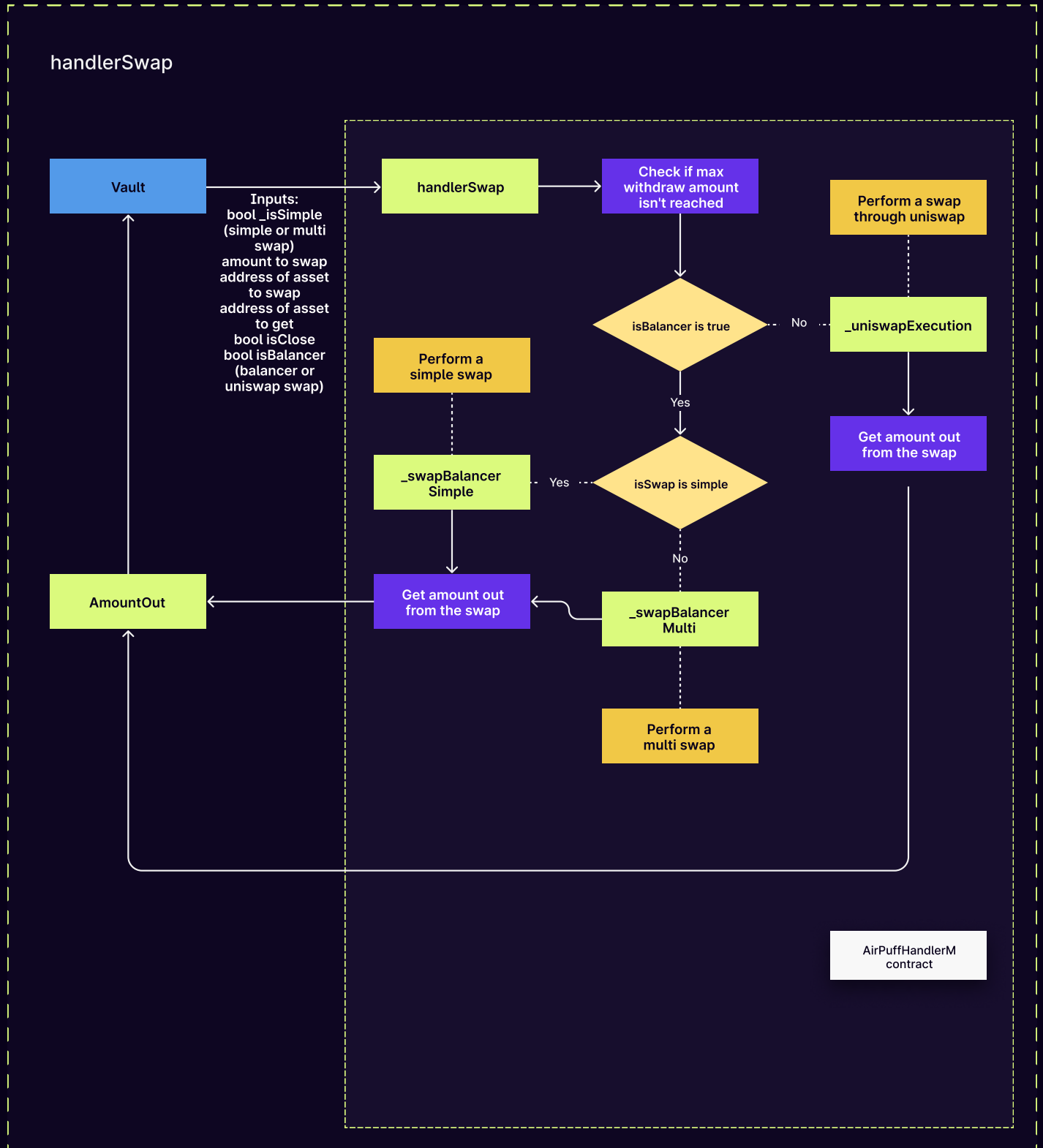
Transfer from operation



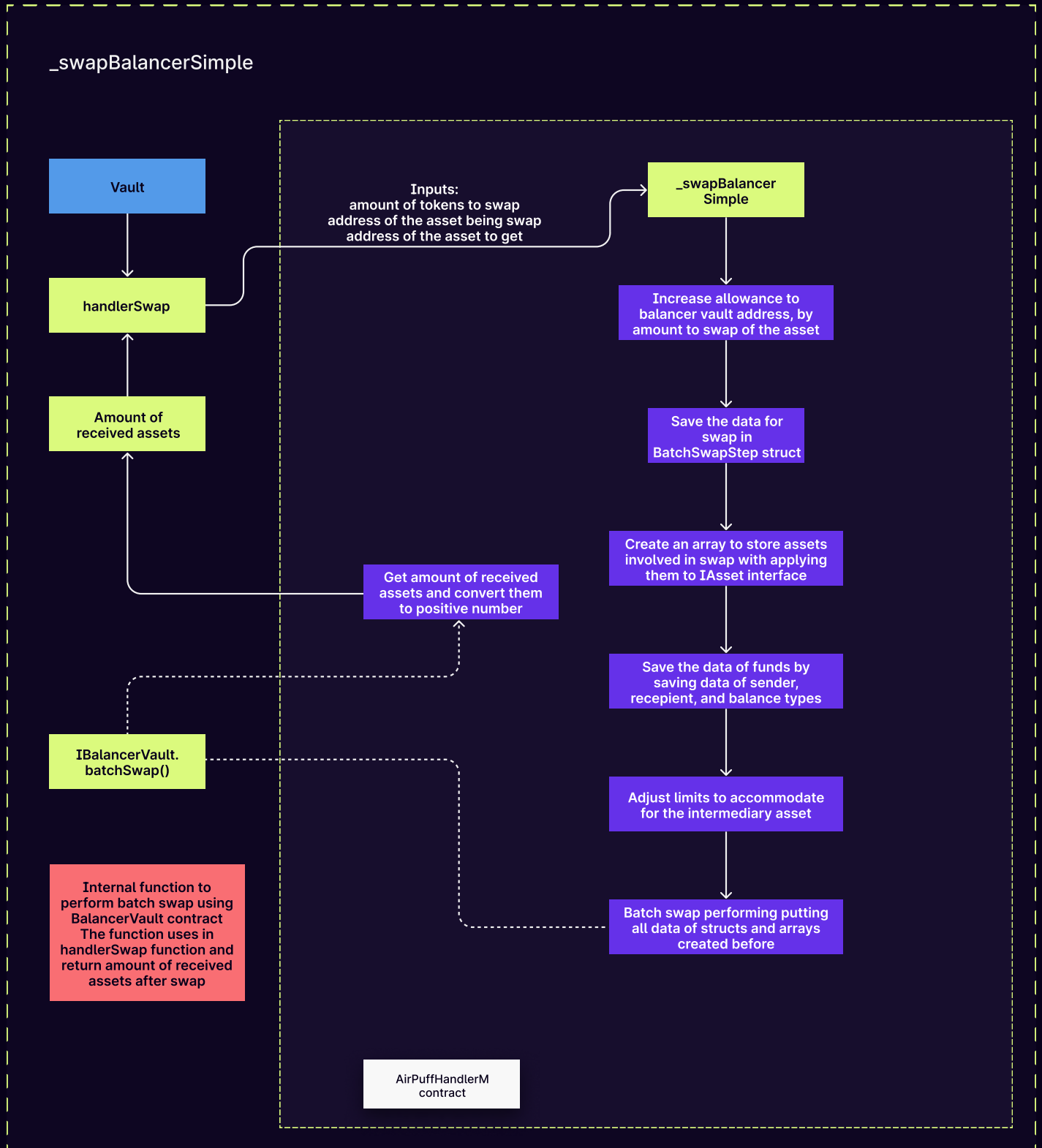
Transfer operation



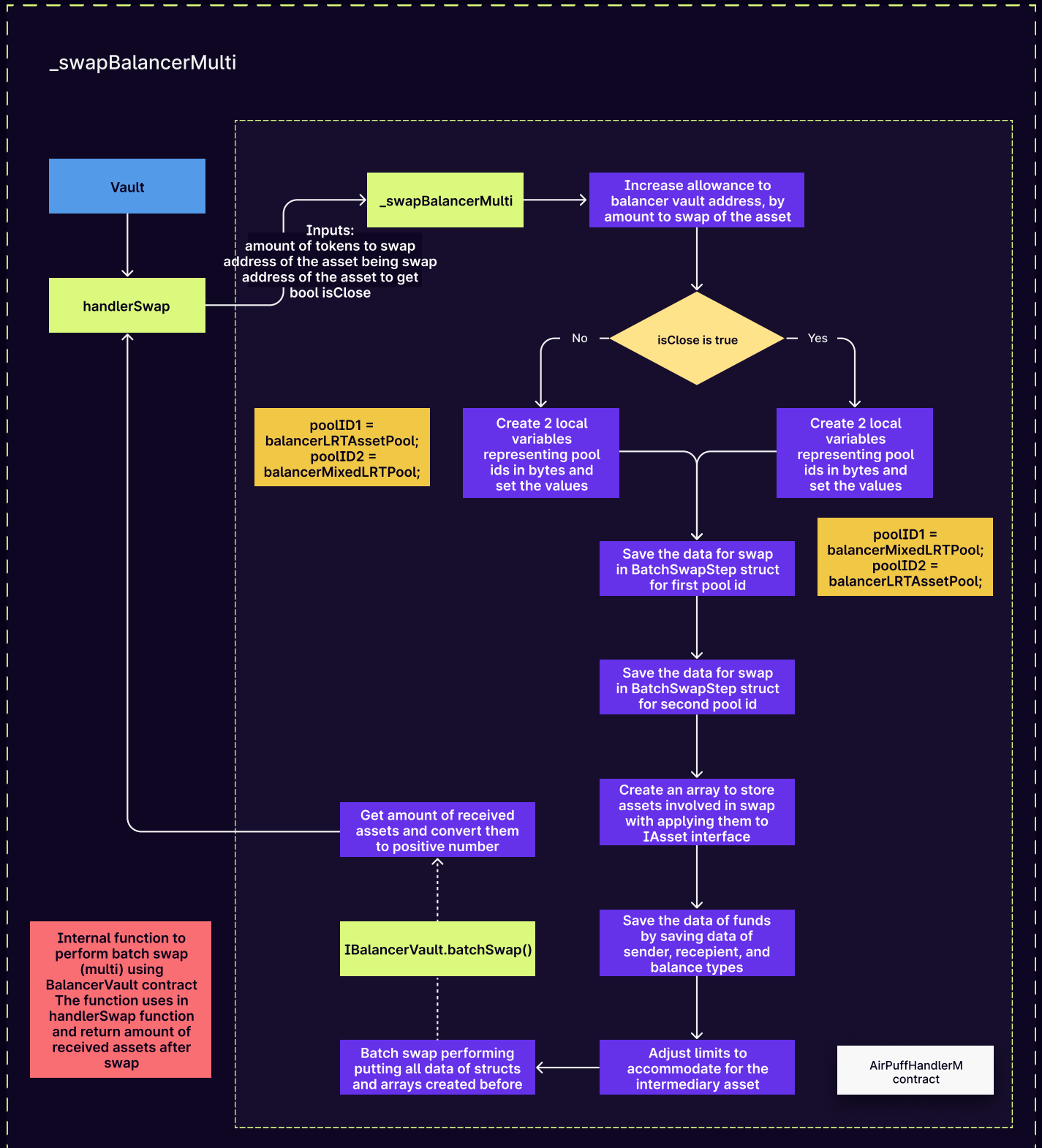
AirPuffHandlerM



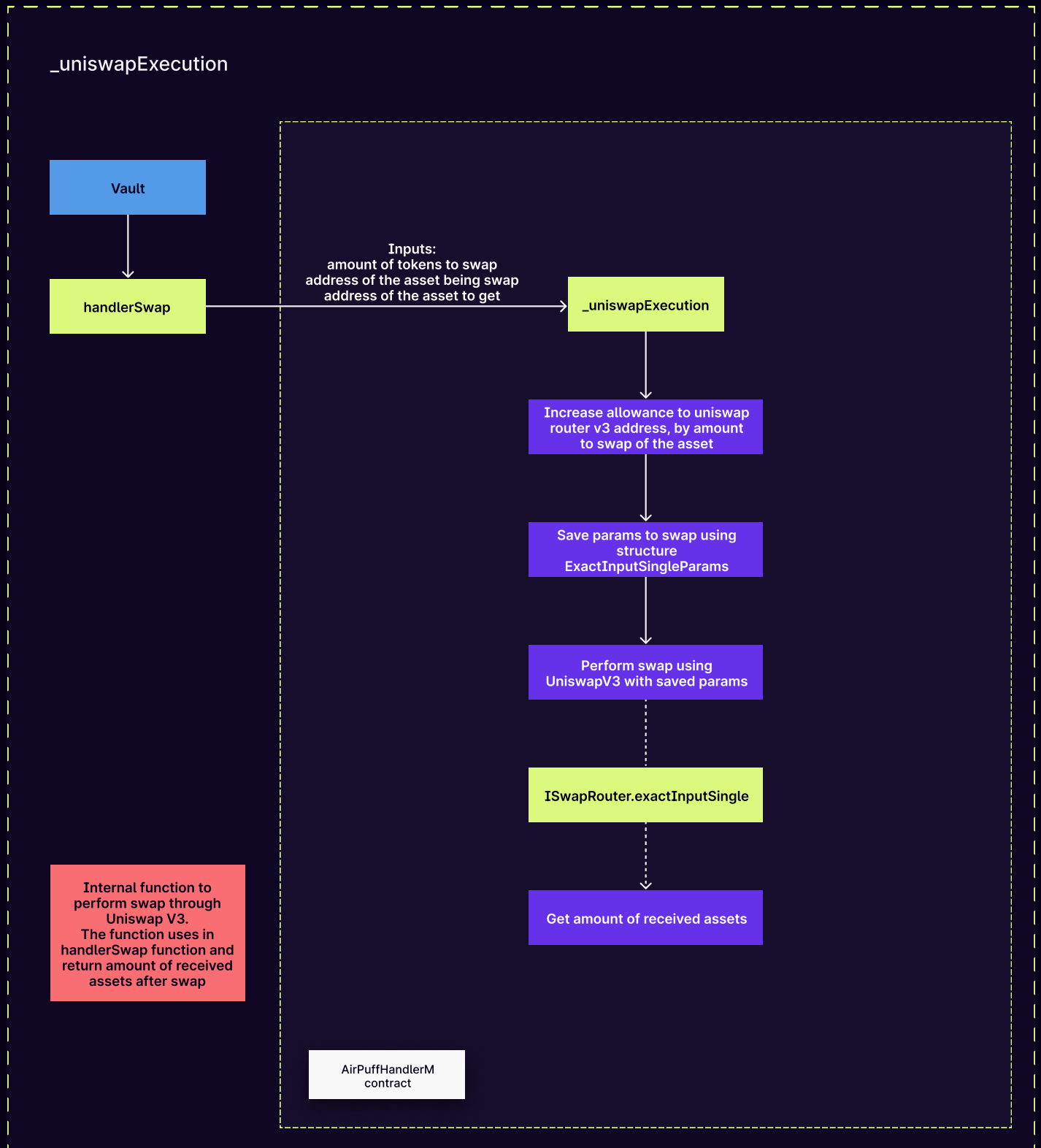
AirPuffHandlerM



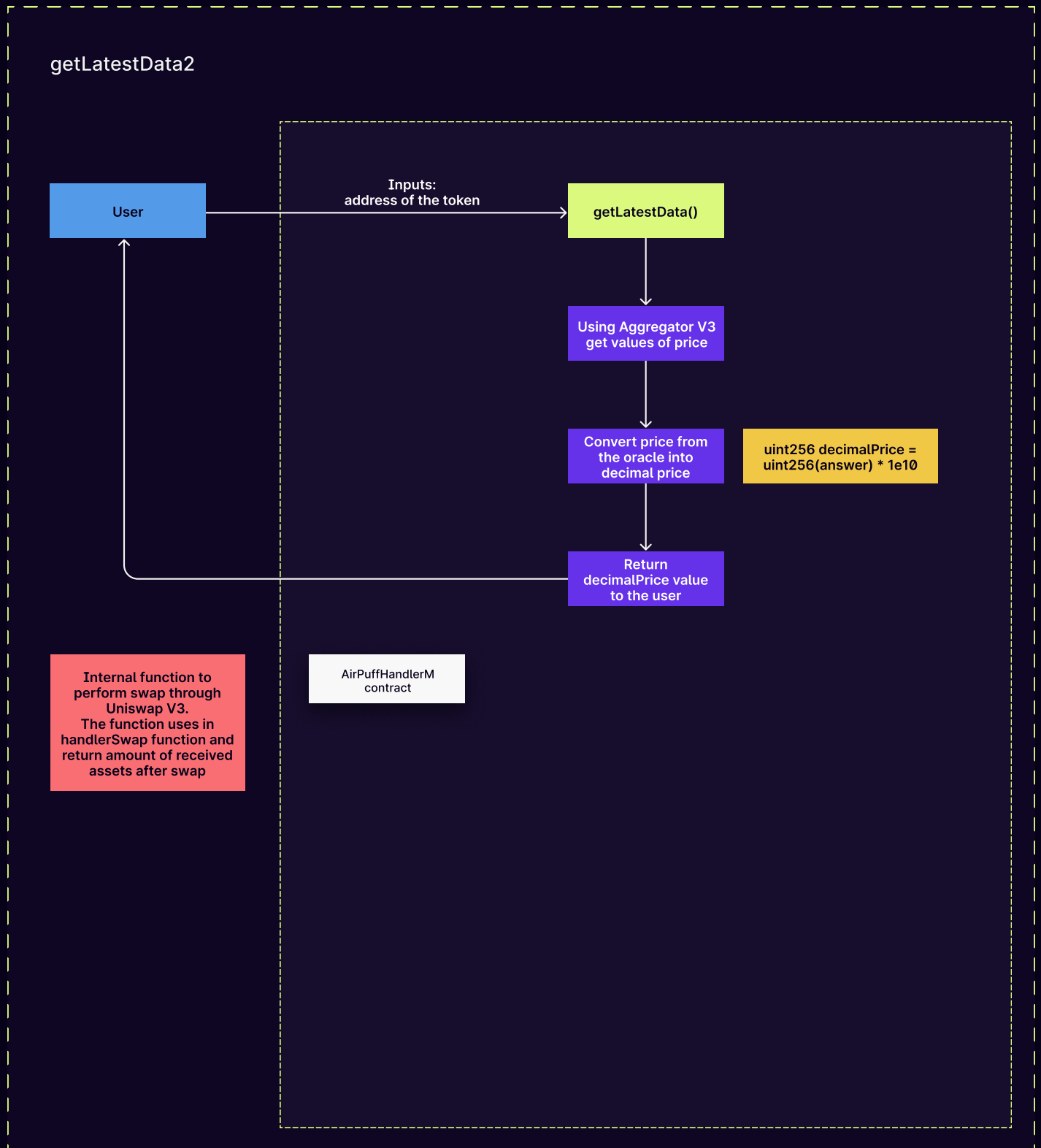
AirPuffHandlerM



AirPuffHandlerM

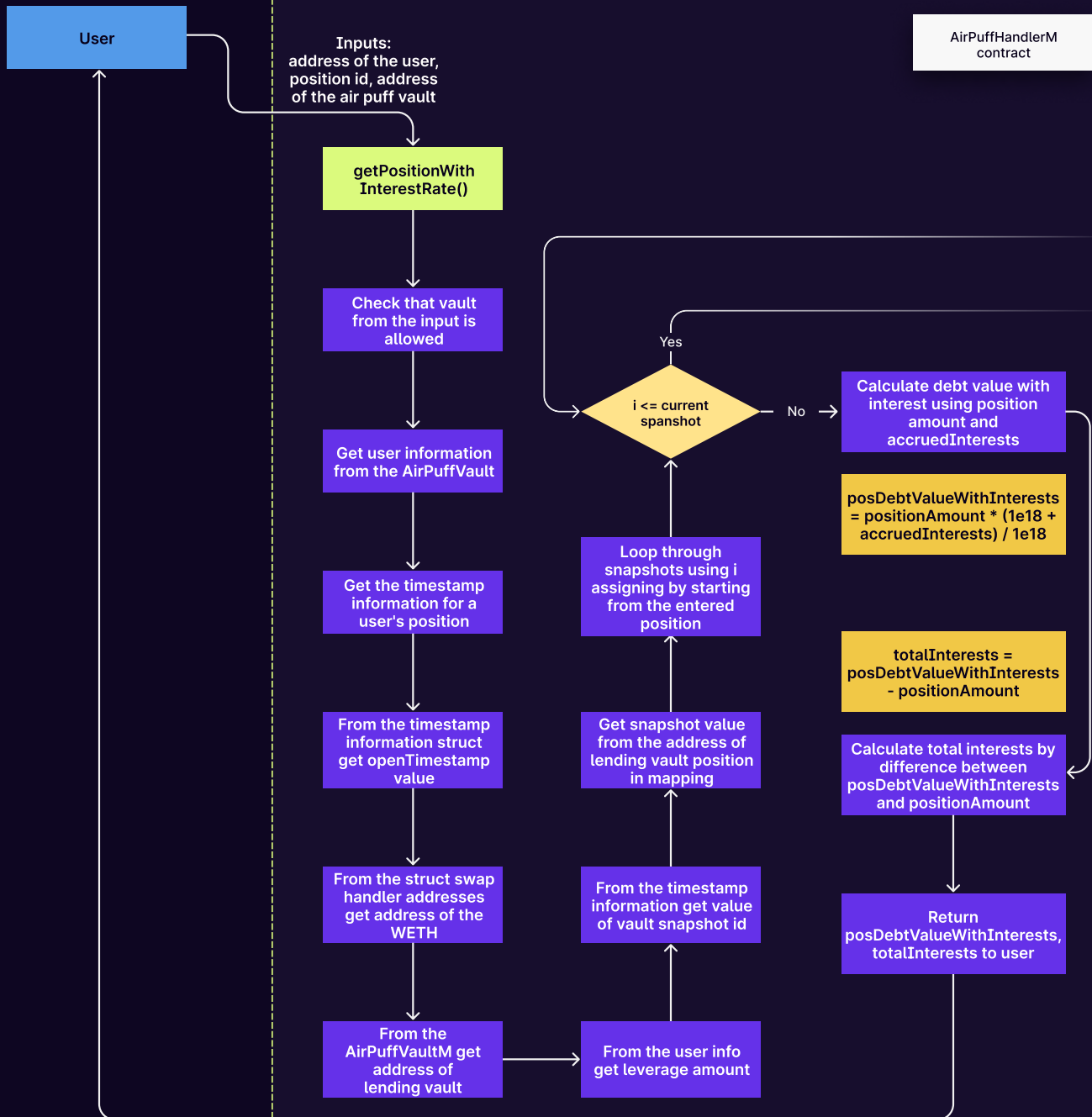


AirPuffHandlerM

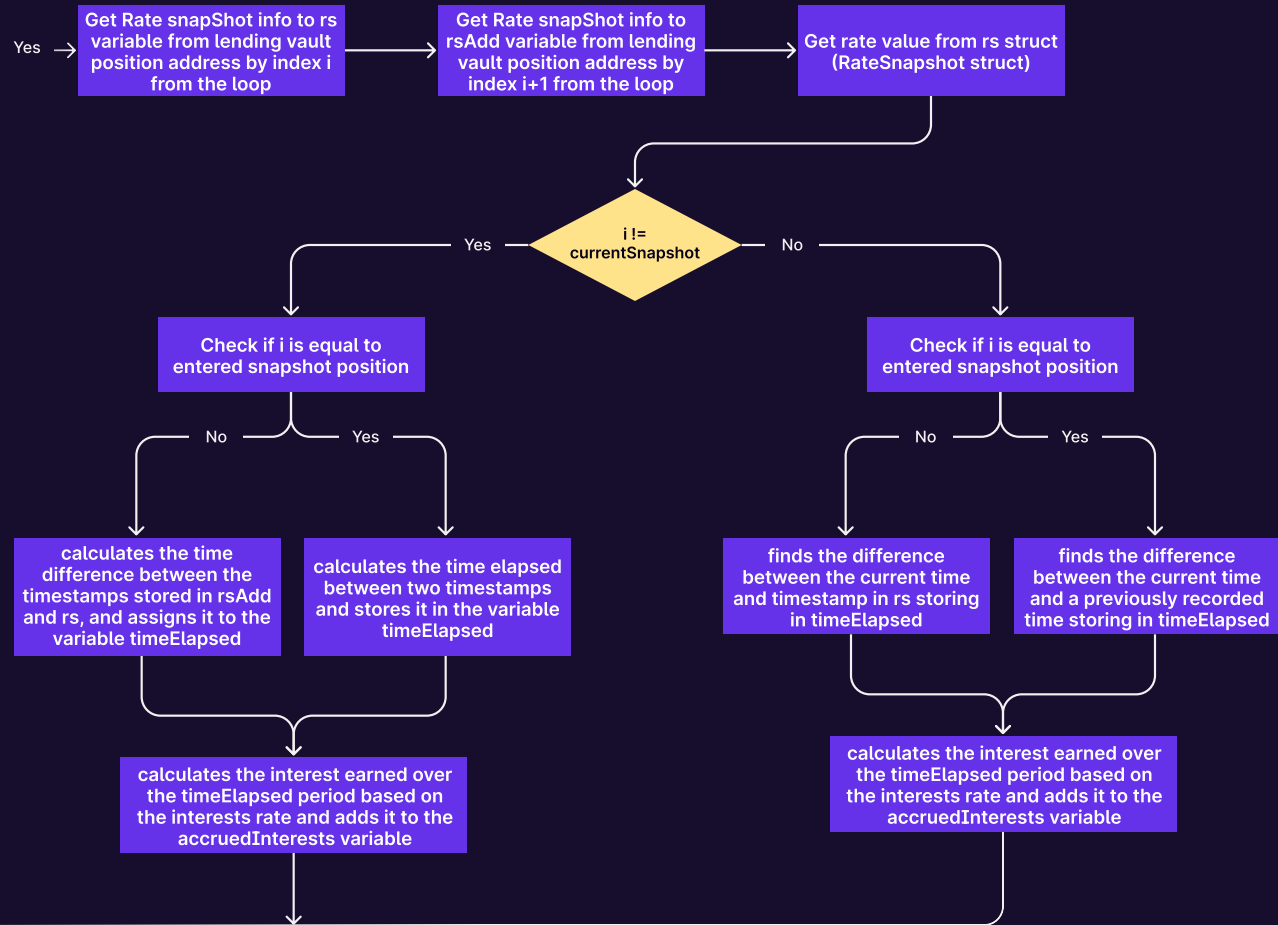


AirPuffHandlerM

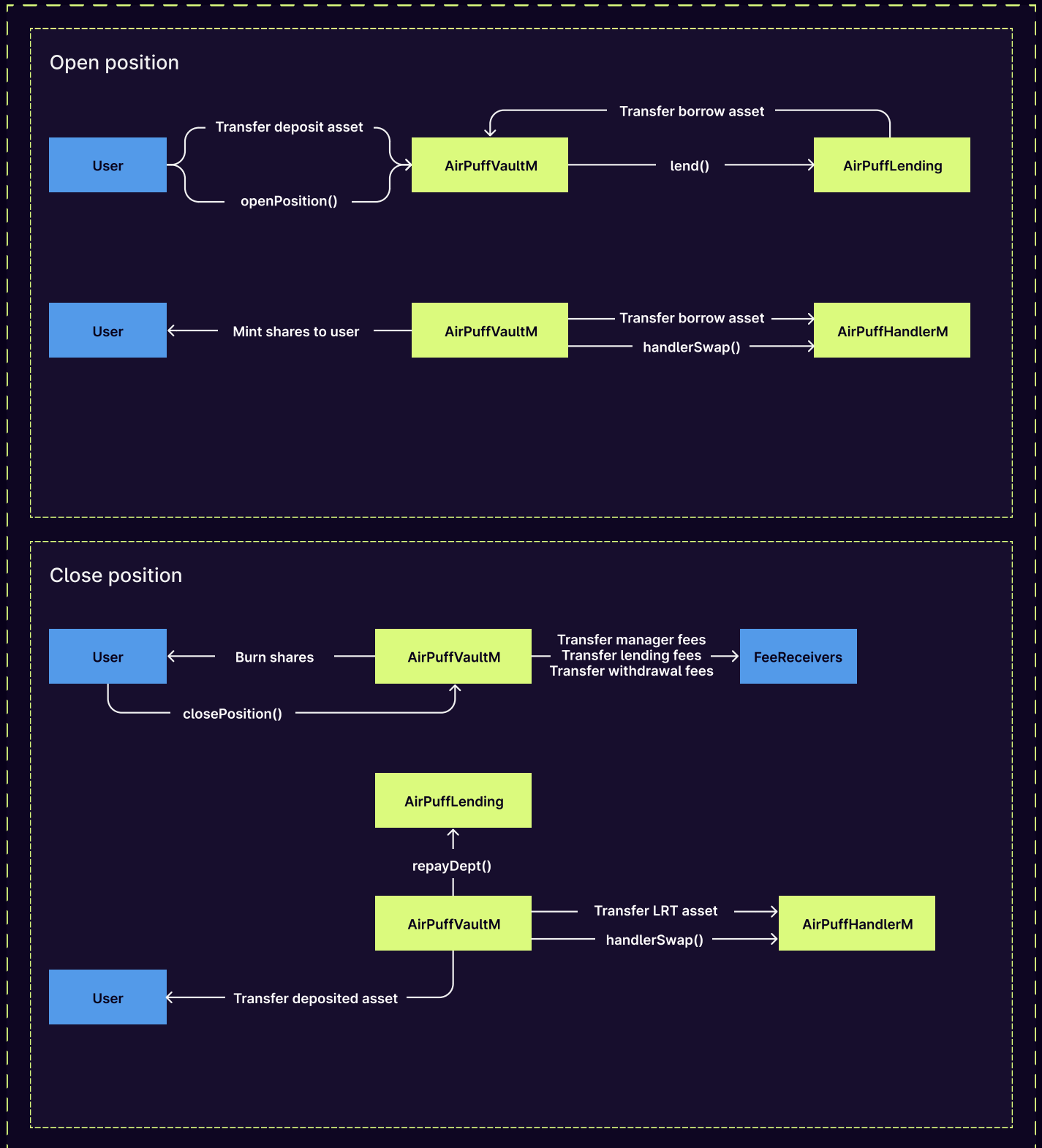
getPositionWithInterestRate2



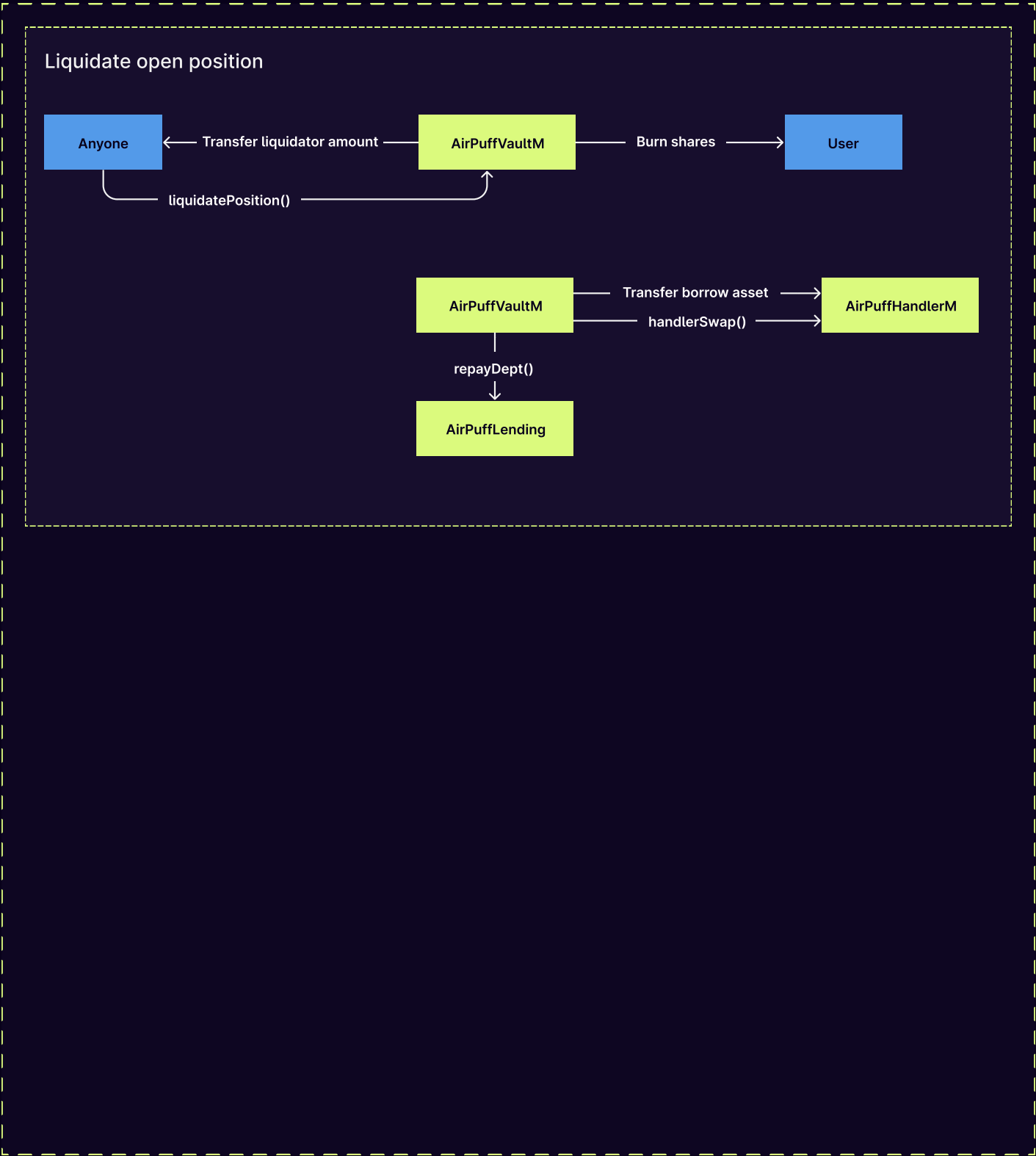
getPositionWithInterestRate2



User flow



User flow



PROTOCOL OVERVIEW

Description

AirPuffVaultM:

AirPuffVaultM is an abstract smart contract similar to the AirPuffHandler contract. It allows users to open leveraged positions. The contract interacts with AirPuffLending to get lending tokens and with AirPuffHandlerM to perform swaps for the main vault token.

Key functionalities include:

- Positions: Gives users the ability to open, close or liquidate leveraged positions.
- LRTAsset storage: Swaps deposited user tokens to assets that are stored on the contract.

AirPuffHandlerM:

A contract similar to the AirPuffHandler contract plays a crucial role in the system. It is used to execute asset swaps, control and manage interest rates, receive asset prices, and manage allowed vaults, taking interest snapshots.

Key functionalities include:

- Swap sessions: Gives you the ability to do conversions of one asset to another using Uniswap V3 or batch swaps through Balancer Vault contract.
- Interest rate control: Control, manage, and change the interest rate to adapt to market situations. As well as creating snapshots of the current interest rate of all allowed vaults.
- Vault Management: Manage vaults, namely adding vaults to the list of allowed vaults.
- Oracles: Integration of oracles from chainlink to get the latest information on assets. Control which assets and oracles are being used.
- Retrieve user information: retrieve user information from AirPuffVaultM, by specific ID position.

AirPuff1X:

AirPuff1X is an ERC-20 token contract with functionalities designed for managing user deposits, withdrawals and transfers within the AirPuff ecosystem, since the contract is abstract.

Key Functionalities:

- Deposits: Users can deposit LRT tokens (represented by the LRTAsset address) into the contract. Upon deposit, a user position is created with details like deposit amount, position ID, timestamp, and closed state. Users get the corresponding amount of shares.
- Withdrawals: Users can close their positions by withdrawing their deposited LRT tokens. A withdrawal fee (configurable by the owner) is applied if set, and the fee is sent to the designated fee receiver address. Users burn their ERC-20 tokens to withdraw their LRT tokens.
- Burn restrictions: By default, direct burning tokens are restricted.
- Transfer restrictions: By default, token transfers are restricted. Only addresses explicitly allowed by the owner through the allowedSenders mapping can transfer tokens.

Roles and Responsibilities

AirPuffVaultM.sol:

1. Owner:

- Set leverage limits for a specific borrowed asset
- Pause/unpause
- Set if swap is simple and if swap is balancer
- Set the addresses for API3Oracle and AirPuffHandlerM
- Set/delete the lending vault address
- Set DTV limit
- Set the protocol fee configuration

AirPuffHandlerM.sol:

1. Owner:

- Set the keeper's address.
- Allows to set and change the uniswap fee amount
- Allows setting and updating the swap handler addresses responsible for swap executions.
- Can allow or disallow vaults to interact with the handler.
- Set Chainlink Oracles for specific assets.
- Set the interest rate parameters using ceiling and maximum interest slopes and base interest rate.
- Has allowance to pause control functions.
- Responsible for setting pool IDs
- Allow to add vaults to LRT assets

1. Keeper:

- Has permission to take a snapshot of the interest rates for all allowed lending vaults by giving the address of the AirPuffVault.

AirPuff1X.sol:

1. Owner:

- Can set the address of the LRT token (deposable asset).
- Can configure who is allowed to send/transfer tokens (through the allowedSenders mapping).
- Can set the protocol fee configuration, including the fee receiver address and the withdrawal fee percentage.
- Can pause and unpause the contract, potentially restricting deposits and withdrawals during a pause.

List of values assets

1. ERC20 ARB Token

The main asset used for lending, borrowing, depositing, and withdrawing within the protocol (AirPuffLendingARB).

2. ERC20 LRT Token

The main asset used for depositing and withdrawing within the protocol (AirPuff1X).

3. WETH, LRT ERC20 tokens

Assets are set in the swap handler addresses and are involved in the swap execution process and vault interactions.

AIRPUFF SETTINGS

Settings

1. AirPuffVaultM.sol:

- StrategyAddresses: The addresses of API3Oracle and AirPuffHandlerM.
- SimpleSwap: Handler for simple swaps (with short swap path).
- Balancer: Handler for swaps processed via the Balancer strategy.
- LendingVault: The address of the lending vault.
- FeeConfiguration: The addresses that receive fees.
- DTVLimit: The debt-to-value limit.
- LeverageLimit: Leverage limits for a specific borrowed asset.

2. AirPuffHandlerM.sol:

- Interest rate configuration (interestRateConfig): This structure consists of base interest rate and slope parameters used to calculate rates under varying utilization conditions. It allows the platform to adapt to changing market dynamics.
- Uniswap fee (uniFee): This is a value that represent fee commission of uniswap. The value represents percentages.
- LRT assets to vaults: Represents mapping of the vault to ERC-20 token, which represents asset LRT.
- LRT asset pool balancer (balancerLRTAssetPool): represents the LRT asset pool of the Balancer in bytes32. It is required for simple swap and multi-swap.
- Mixed LRT pool balancer (balancerMixedLRTPool): Represents the mixed LRT pool of Balancer in bytes32. It is required for multi-swap.
- Keeper (keeper): The address of the keeper. The keeper has permission to call the function to take interest snapshots.
- Swap handling addresses (swapHandlerAddresses): This variable has a type structure that defines critical components for swaps, such as the address of the BalancerVault contract, the address of the Uniswap v3 router, and the WETH token.
- Vault Allowlisting (allowedVaults): Mapping is responsible for managing permitted vaults that can interact with the handler functions, such as getting position value with interest rate, taking snapshots of interest rates, and swap execution.
- Chainlink Oracles (chainlinkOracle): This mapping is responsible for setting and updating the addresses of Chainlink oracles for tokens. It is also used to get the latest price data of the token.

3. AirPuff1X.sol:

- LRTAsset: The address of the ERC-20 token representing the depositable asset (LRT).
- MAX_BPS: A constant representing the maximum basis points value (used for fee calculations).
- feeConfiguration: Struct containing the fee receiver address and withdrawal fee percentage.
- allowedSenders: Mapping to specify addresses allowed to transfer tokens (by default, all transfers are restricted).

Deployment script

AirPuffHandlerM:

Deployment script was not provided for the AirPuffHandlerM contract and settings to him. However, initialization function include some first settings for interest rate config structure. Base interest rate initializing with hardcoded value which is equal to value 5e16, CEIL_SLOPE_1 is equal to 8e17, while CEIL_SLOPE_2 is equal to 1e18. MAX_INTEREST_SLOPE_1 is 1e17, and MAX_INTEREST_SLOPE_2 is 3e17. There is also a value to set uniswap fee, amount of is equal to 500 or 0.5%.

AirPuffVaultM:

Deployment script was not provided since AirPuffVaultM is an abstract contract.

AirPuff1X:

Deployment script was not provided since AirPuff1X is an abstract contract.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

 **Critical**
The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

 **High**
The issue affects the ability of the contract to compile or operate in a significant way.

 **Medium**
The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.

 **Low**
The issue has minimal impact on the contract's ability to operate.

 **Informational**
The issue has no impact on the contract's ability to operate.

COMPLETE ANALYSIS

HIGH-1 | RESOLVED

Quoter uniswap integration.

`AirPuffHandlerM._uniswapExecution()`.

The function uses a quoter for getting the expected amount out without executing the swap and uses this value as `amountOutMinimum` as a parameter for the swap. However, this method of getting the expected amount is not recommended for use for several reasons. The first and main reason is that using a quoter on a chain is not stable enough, which often causes reverts. Quoter is recommended for use in the backend regarding uniswap documentation. Also, quoter is not gas efficient method.

Recommendation:

Remove quoter integration in the `_uniswapExecution` function and add a more stable and gas-efficient variant for checking the `amountOutMinimum` value by adding the `amountOutMinimum` argument to the swap function and swap structure.

Post audit.

Quoter was removed. Price is calculated based on the price returned from oracle.

HIGH-2 | RESOLVED

Tokens can be burnt explicitly.

`AirPuffVaultM.burnFrom()`, `AirPuffVault.burnFrom()`, `AirPuff1X.burnFrom()`

Native function ``burnFrom()`` still can be executed, so tokens can be burnt. Since the contracts try to disallow any burn and transfer transactions, it will be necessary to disallow function ``burnFrom()`` to avoid related risks. For now the user can approve burning tokens for himself and burn any amount.

Recommendation:

Add overridden ``burnFrom()`` function for the contracts, that disable burn.

Post audit.

``burnFrom()`` functions were disabled.

Frontrun attack possible.

1. AirPuffHandlerM._curveSimpleSwap(), _curveRouterSwapWithPaths()

_curveSimpleSwap function uses the `exchange` function to make the swap. One of the arguments that is required for swap is responsible for the minimum amount out, but min_dy is always set to 0.

_curveRouterSwapWithPaths() function uses the `exchange` function through router has an argument responsible for minimum, amount out, but _expected argument also always be 0. These can cause a front-running attack as the amount is not validated for a certain desired minimum.

2. AirPuffHandlerM._swapBalancerSimple(), _swapBalancerMulti()

Both of these functions have a limits array that is responsible for setting the minimum amount of assets to receive after the swap. In both functions, the minimum amount is set to -1, which means the amount is not specified. This is also a crucial part of the swap to prevent the attack by setting the desired minimum.

Recommendation:

Add checks for the amount of swapped tokens to prevent possible attacks. Pass minAmountOut to exchange function as for _curveSimpleSwap and _curveRouterSwapWithPaths. Add minimum amount to receive in limits array in function _swapBalancerSimple and _swapBalancerMulti.

Add a modifier which will revert if `msg.sender != tx.origin`

Use modifier in the following functions:

- AirPuffVault: openPosition, closePosition, liquidatePosition
- AirPuff1X: openPosition, closePosition
- AirPuffVaultM: openPosition, closePosition, liquidatePosition
- AirPuffLendingARB: deposit, withdraw

Post audit.

Now, minAmountOut is calculated based on Chainlink or Api3Oracle oracles. The AirPuff team has also added an interface for calculating price based on several on-chain oracles (IVectorOracle). Though this approach is not implemented right now, the AirPuff team has prepared minor developments for it. Modifier `msg.sender != tx.origin` was added to AirPuffVaultM contract. Auditors highly recommend to add this modifier to other contracts as well, as an additional measure over implemented steps.

Broken logic in check.

`AirPuffHandlerM._curveRouterSwapWithPaths()`.

The function has two checks if any Curve pools are set. If no addresses are set, this check will never be reached because `curvePoolsIn` and `curvePoolsOut` are structs with 5 zero addresses by default. So, the check will always show that the Curve pools' length is greater than zero. This can lead to problems during the swap. The issue is marked as Low-risk because the tx will revert without losing funds, thus marking an unreachable pool.

Recommendation:

Change the "require" check for the correct pools in the `_curveRouterSwapWithPaths` function. Arrays in this struct have a limited number of addresses (11), and by default, all these addresses are equal to zero. We recommend adjusting the logic to check that at least the first 2 addresses in both arrays do not equal zero.

Post audit.

The logic of the ``require`` check was changed as recommended; the first 2 addresses in the swap path of the Curve router are checked for non-zero values.

Duplicated check.

`AirPuffHandlerM.handlerSwap()`, `_curveSimpleSwap()`.

Before processing the swap via the Curve, the contract checks if any Curve pool is set for the vault. If the strategy has no specific router path for chosen tokens, the contract calls the `_curveSimpleSwap` function, which contains one more check for the pool specified for the strategy. Since both checks are the same, the one in the `_curveSimpleSwap` function can be removed to save some gas.

Recommendation:

Remove require check in `_curveSimpleSwap` function for set curve pool for the strategy.

Routing and swap parameters must be determined off-chain..

`AirPuffHandlerM._curveRouterSwapWithPaths()`

The function uses already-set parameters to make swaps through the Curve protocol with the router. All parameters are set directly in the contract through multiple functions. According to the Curve documentation, all these parameters should be determined off-chain and set as arguments to function. Also, the router was designed for gas efficiency and ease of use.

Recommendation:

Verify that the parameters for swapping through the curve router set on-chain correspond to your business logic.

Post-audit.

The client verified, that current swap paths have enough liquidity and correspond to current business logic.

Lack of event emitting.

`AirPuffVaultM.setProtocolFee()`

This function is responsible for setting crucial addresses and fee amounts. Five parameters participate in the setting, but only three emit in the event, which may cause some issues.

Recommendation:

Consider adding all values for the emitting event in `setProtocolFee()` OR verify that the management fee percentage and receiver address should not be revealed in the event.

NatSpec required.

NatSpec is a documentation format that helps in the readability of contract functions and a better understanding of what they are doing. It is recommended that you have NatSpec in your contracts. More details about it here: <https://docs.soliditylang.org/en/v0.8.21/natspec-format.html>.

Recommendation:

Add NatSpec and comments to contract functions and variables. To emphasize the contract's role, also add NatSpec to the contract description.

Airpuff protocol (scoped contracts)	
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting AirPuff in verifying the correctness of their contracts code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the AirPuff contracts requirements for details about issuance amounts and how the system handles these.

AirPuffHandlerM

Initialization

- ✓ Should initialize properly
- ✓ Should revert if trying to initialize twice

Setters

Uniswap fee

- ✓ Should allow to update uni fee
- ✓ Should revert if caller is not owner

Uniswap slippage

- ✓ Should allow to set uniswap slippage
- ✓ Should revert if caller is not owner
- ✓ Should revert if slippage is too big

Balancer deadline

- ✓ Should allow to set balancer deadline
- ✓ Should revert if caller is not owner

Keeper

- ✓ Should allow to set keeper
- ✓ Should revert if keeper is zero address
- ✓ Should revert if caller is not owner

Swap Handler Addresses

- ✓ Should allow to set swap handler addresses
- ✓ Should revert if caller is not owner

Vaults management

- ✓ Should allow to set vaults
- ✓ Should revert if caller is not owner
- ✓ Should revert if vault is zero address

Chainlink oracles management

- ✓ Should allow to set chainlink oracles
- ✓ Should revert if caller is not owner
- ✓ Should revert if asset is zero address

Interest rate config

- ✓ Should allow to set interest rate config
- ✓ Should revert if caller is not owner
- ✓ Should revert if ceil slopes invalid
- ✓ Should revert if max interest slopes invalid

Curve router controller

- ✓ Should allow to set curve router
- ✓ Should revert if caller is not owner
- ✓ Should revert if curve router is zero address

Strategy to curve pool

- ✓ Should allow to set strategy to curve pool
- ✓ Should revert if caller is not owner

Vault to LRT token

- ✓ Should allow to set vault to LRT token
- ✓ Should revert if caller is not owner

Strategy router swap path

- ✓ Should allow to set strategy router swap path (52ms)
- ✓ Should revert if caller is not owner

Strategy to curve pools

- ✓ Should allow to set strategy to curve pools
- ✓ Should revert if caller is not owner

Curve pools to swap paths

- ✓ Should allow to set curve pools swap paths (84ms)
- ✓ Should revert if caller is not owner

Strategy to pool ids

- ✓ Should allow to set strategy to pool ids
- ✓ Should revert if caller is not owner

Interest snapshot taker

- ✓ Should allow to take interest snapshot
- ✓ Should revert if caller is not keeper
- ✓ Should revert if vault is not allowed

User info getter

- ✓ Should allow to get user info (3021ms)

User timestamp getter

- ✓ Should allow to get user timestamp (715ms)

Position with interest rate getter

- ✓ Should allow to get position with interest rate in current snapshot (798ms)
- ✓ Should revert if vault is not allowed (737ms)

Price getter

- ✓ Should allow to get price of asset (1061ms)

Interest rate getter

- ✓ Should allow to get interest rate when utilRate greater than CEIL_SLOPE_1 (915ms)
- ✓ Should allow to get interest rate when utilRate less than CEIL_SLOPE_1 (382ms)

Swap

- ✓ Should revert if caller is not allowed vault

Uniswap

- ✓ Should allow to open position and make swap through Uniswap (508ms)

Balancer

- ✓ Should allow to make swap thorough simple balancer swap (4004ms)
- ✓ Should allow to make swap thorough multiple balancer swap (3115ms)

Curve

- ✓ Should allow to make swap through Curve (crvUsd->usdc.e) (5895ms)
- ✓ Should allow to make swap through Curve (usdc.e->crvUsd) (983ms)
- ✓ Should allow to swap through Curve using router (6627ms)
- ✓ Should allow to swap through Curve using router with isClose true (865ms)

AirPuff1X

Initialization

- ✓ Initialize correctly (81ms)
- ✓ Initialize twice fails
- ✓ Initialize with zero address fails (186ms)

Set LRT asset

- ✓ Set by non-owner fails
- ✓ Set zero address fails
- ✓ Set LRT asset (75ms)

Set allowed senders

- ✓ Set by non-owner fails
- ✓ Set allowed sender (104ms)

Set protocol fee

- ✓ Set by non-owner fails
- ✓ Set fee more than max fails
- ✓ Set zero address as fee receiver fails
- ✓ Set withdrawal fee and fee receiver (75ms)

Pause

- ✓ Pause by non-owner fails
- ✓ Pause

Unpause

- ✓ Unpause by non-owner fails
- ✓ Unpause (72ms)

Get all users

- ✓ Get users (60ms)

Get total numbers of open position by user

- ✓ Get users (51ms)

Open position

- ✓ Open while paused fails
- ✓ Open for zero amount fails
- ✓ Open position (216ms)
- ✓ Open position twice by one user (171ms)

Close position

- ✓ Close while paused fails (41ms)
- ✓ Close invalid position fails
- ✓ Close while paused fails (53ms)
- ✓ Close position with withdrawal fee set to zero (136ms)
- ✓ Close position with non-zero withdrawal fee (144ms)

Transfer from

- ✓ Transfer from non-allowed address fails (40ms)
- ✓ Transfer (127ms)

Transfer

- ✓ Transfer with non-allowed address fails
- ✓ Transfer (85ms)

Burn

- ✓ Direct burn fails

Burn from

- ✓ Direct burn from fails

AirPuffVaultM

- # Set is swap simple
 - ✓ Set by non-owner fails (45ms)
 - ✓ Set is swap simple (144ms)
- # Set is balancer
 - ✓ Set by non-owner fails
 - ✓ Set is balancer (139ms)
- # Set strategy addresses
 - ✓ Set by non-owner fails
 - ✓ Set strategy addresses (122ms)
- # Set lending vault
 - ✓ Set by non-owner fails
 - ✓ Set lending vault (67ms)
- # Set DTV limit
 - ✓ Set by non-owner fails
 - ✓ Set DTV limit
- # Set min max leverage bounds
 - ✓ Set by non-owner fails
 - ✓ Set incorrect min leverage bound fails
 - ✓ Set incorrect max leverage bound fails
 - ✓ Set swapped bounds fails
 - ✓ Set leverage bounds
- # Set protocol fee
 - ✓ Set by non-owner fails
 - ✓ Set invalid withdrawal fees fails
 - ✓ Set invalid m fee percent fails
 - ✓ Set fee (149ms)
- # Pause
 - ✓ Pause by non-owner fails
 - ✓ Pause
- # Unpause
 - ✓ Unpause by non-owner fails
 - ✓ Unpause (68ms)
- # Get all users
 - ✓ Get users (330ms)
- # Get total numbers of open positions by user

- ✓ Get number of positions (192ms)
- # Get updated debt
 - ✓ Get debt if position liquidated (337ms)
 - ✓ Get debt if position closed (393ms)
 - ✓ Get debt (211ms)
- # Get current position value
 - ✓ Get current position value if position liquidated (262ms)
 - ✓ Get current position value if position closed (336ms)
 - ✓ Get current position value (175ms)
- # Open position
 - ✓ Open while paused fails
 - ✓ Invalid leverage fails
 - ✓ Leverage lesser than min fails
 - ✓ Leveraged amount is zero fails (57ms)
 - ✓ Open position twice (231ms)
 - ✓ Open position (197ms)
- # Close position
 - ✓ Close position with invalid ID fails
 - ✓ Close liquidated position fails (298ms)
 - ✓ Close position twice fails (348ms)
 - ✓ Close position if DTVLimit reached fails (214ms)
 - ✓ Close position (413ms)
 - ✓ Close position with zero fees (387ms)
 - ✓ Close position with zero total interest (400ms)
 - ✓ Close position with leverage amount greater than amount out (326ms)
- # Liquidate position
 - ✓ Liquidate position with invalid ID fails
 - ✓ Liquidate position twice fails (263ms)
 - ✓ Liquidate position if DTV limit unreached fails (185ms)
 - ✓ Liquidate position (262ms)
- # Transfer from
 - ✓ Direct transfer from has no effect
- # Transfer
 - ✓ Direct transfer has no effect
- # Burn
 - ✓ Direct burn has no effect
- # Burn from
 - ✓ Direct burn from has no effect

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% FUNCS	% BRANCH	% LINES
AirPuffHandlerM.sol	100	90.48	100
AirPuff1X.sol	100	93.75	100
AirPuffVaultM.sol	100	90.7	100
Total %	100	91.64	100

We are grateful for the opportunity to work with the AirPuff team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the AirPuff team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

