MAGIC YEARN

SMART CONTRACT AUDIT

zokyo

August 17th 2022 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

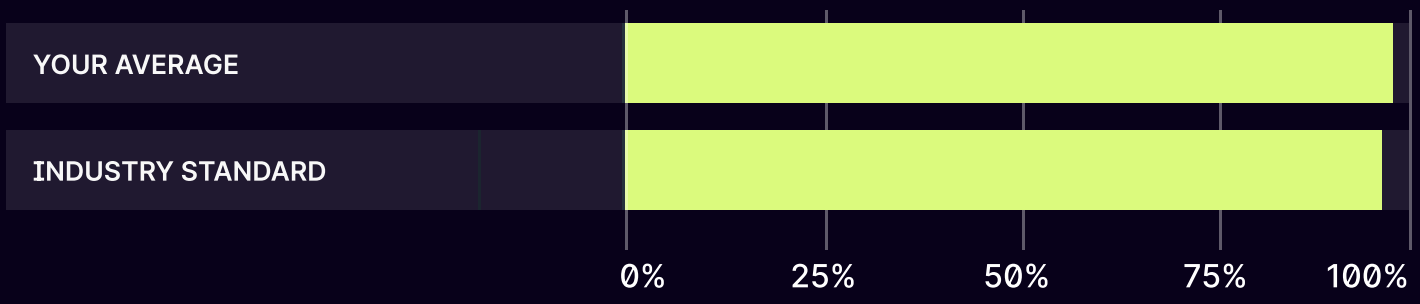
TECHNICAL SUMMARY

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

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

Contract Status

Testable Code



The 98% of the code is testable, which is above the standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Magic Yearn 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

The source code of the smart contract was taken from the Magic Yearn repository.
https://github.com/myearn-official/contracts

Initial commit: 47edf075a1ded8a902f08f9787e4ceb67904e95f

Last audited commit: 6c847a2e24ef93e47e165636cd7383b1622fd56c

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

- staking/MyStaking.sol
- token/MyExchange.sol
- utils/FeeReducer.sol
- utils/StakingFactory.sol
- token/MyShare.sol
- token/MyToken.sol
- wrapper/MyWrapper.sol

Throughout the review process, Zokyo Security ensures 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 in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Magic Yearn smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented any issues as they were discovered. A part of this work included writing a unit test suite using the Truffle 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.
02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.
04	Thorough manual review of the codebase, line by line.

Executive Summary

Zokyo Security has performed a security audit for the Magic Yearn protocol. During the audit, auditors checked the whole set of contracts manually, with the usage of automated tools and by writing a set of unit-tests with additional scenarios.

The protocol represents:

- MY tokens pegged to ERC20, the holders of which can earn on transfer fees;
- A staking protocol;
- MY Share ERC20 contract with a detailed emission model;
- An exchanger for migrating the older version of the token to a new one;
- A fee reducer that is in charge of users' fees reduction across the protocol;
- A wrapper contract, which allows users to buy MY tokens for pegged tokens and add liquidity to LP tokens in Uniswap V2 protocol.

All the contracts were carefully checked by auditors. Special attention was paid to the safety of users' funds across the protocol, formulas applied in emission and staking, and the correctness of the interaction with Uniswap V2 protocol.

There were some critical and high issues found within the contracts. Some of them were successfully verified by the team to ensure that the business logic of the contracts operates as intended. Other issues related to the corectness of the storing pending rewards and withdrawals of deposited tokens were successfully fixed by the Magic Yearn team. All other issues with medium and lower severity were also resolved by the team.

It should be noted that there is one informational issue that wasn't resolved by the team. The issue is related to the ability of an admin to blacklist any MY token holder and distribute their funds among the holders of MY token. According to the team, only inactive users will be blacklisted so that no funds can get locked on one account. The team has also promised to search for a solution to programmatically track which account is inactive and should be blacklisted so that a valid account can't be blocked. However, such mandatories are currently not present in the code.

Zokyo Security has also prepared a set of unit-tests with a mainnet fork to ensure that the protocol works correctly in the conditions close to the live mainnet. Auditors have carefully checked the functionality of smart contracts, such as transfers with fees, fee reduction, staking, reward distribution, minting of MY Share token and interaction with Uniswap V2 protocol. There was one critical and some informational issues found during unit-testing. The critical issue was connected to the possible blocking of funds in staking (See Critical-3 issue), and informational issues were mostly connected to the optimization of the code. All these issues were also successfully fixed by the Magic Yearn team.

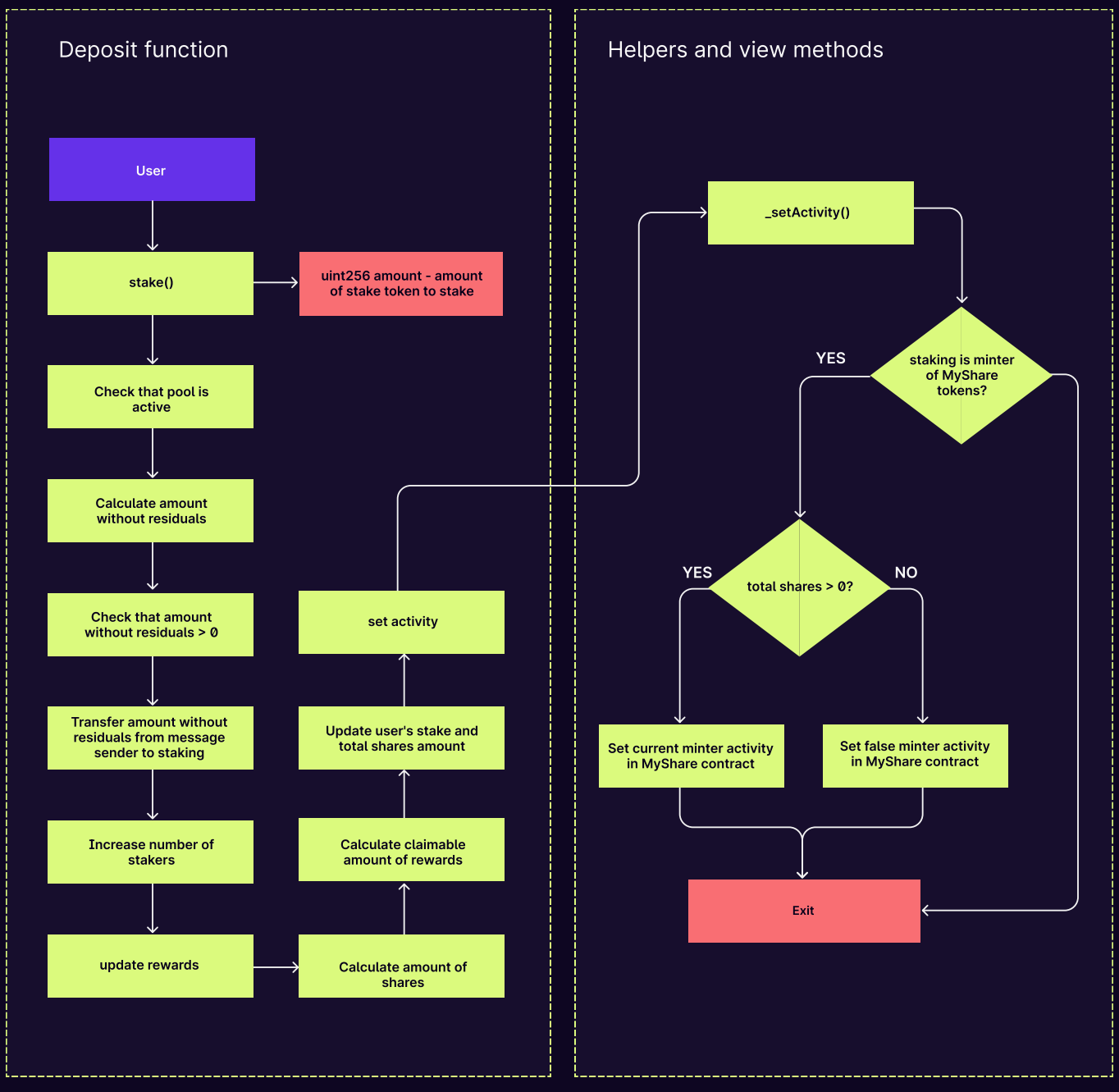
The overall security of the protocol is high enough. The code has passed all the security tests, and no critical vulnerabilities were found after the team had fixed the issues. The code has high readability and natspec documentation.

Also, it should be mentioned that the code has no original unit tests coverage (or it was not provided by the team), thus, the auditors have provided their own full unit-test coverage.

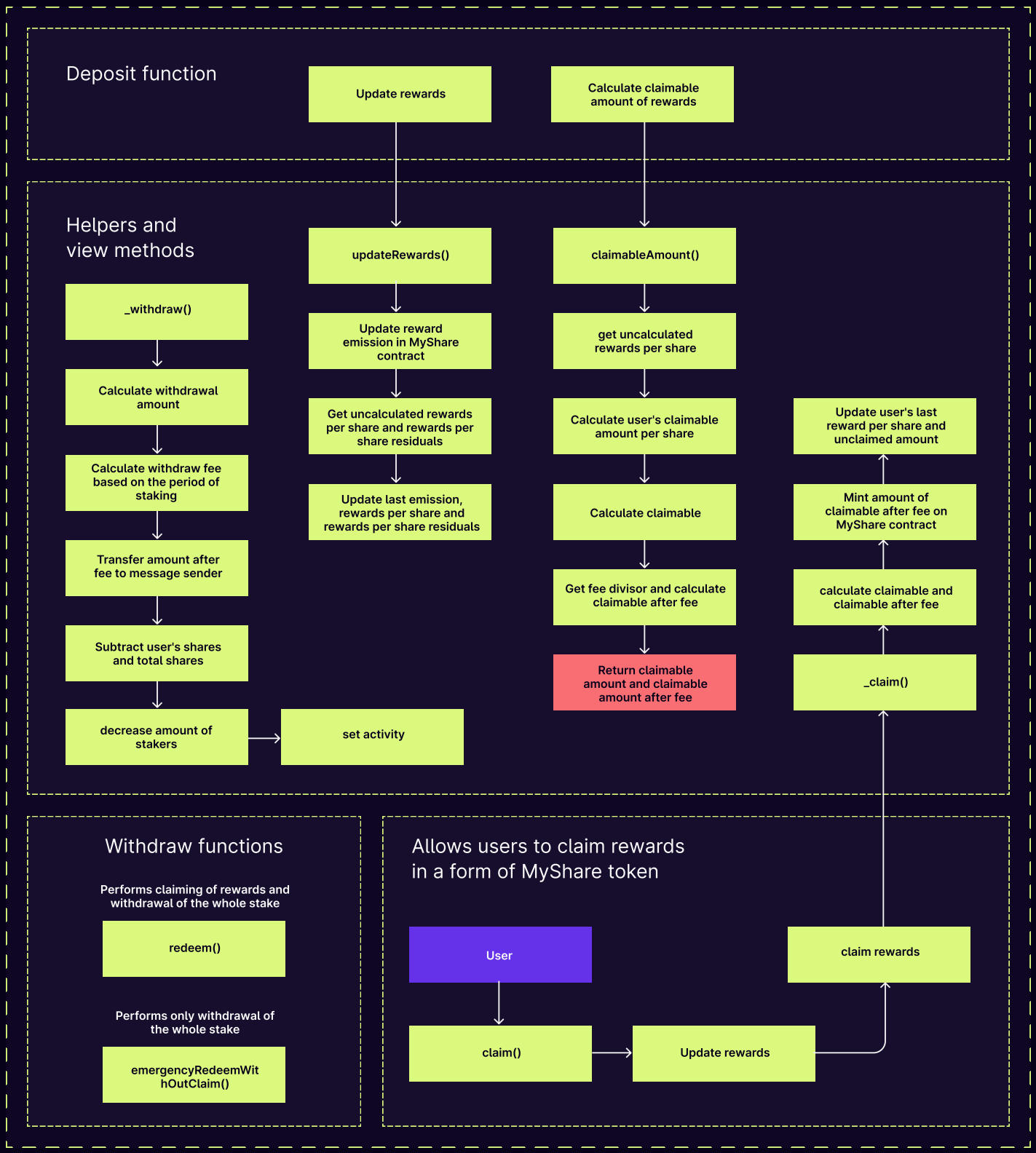
PROTOCOL OVERVIEW

MYSTAKING.SOL

MyStaking is a staking contract that allows users to stake a stake token and earn MyShare tokens. A withdrawal can be performed at any time, but a fee can be applied to the withdrawal amount depending on the duration of the stake. The longer a user stakes, the less the fee. After a certain period of stake, the fee is not applied. Also, users can't withdraw a part of the stake, only the full stake amount.



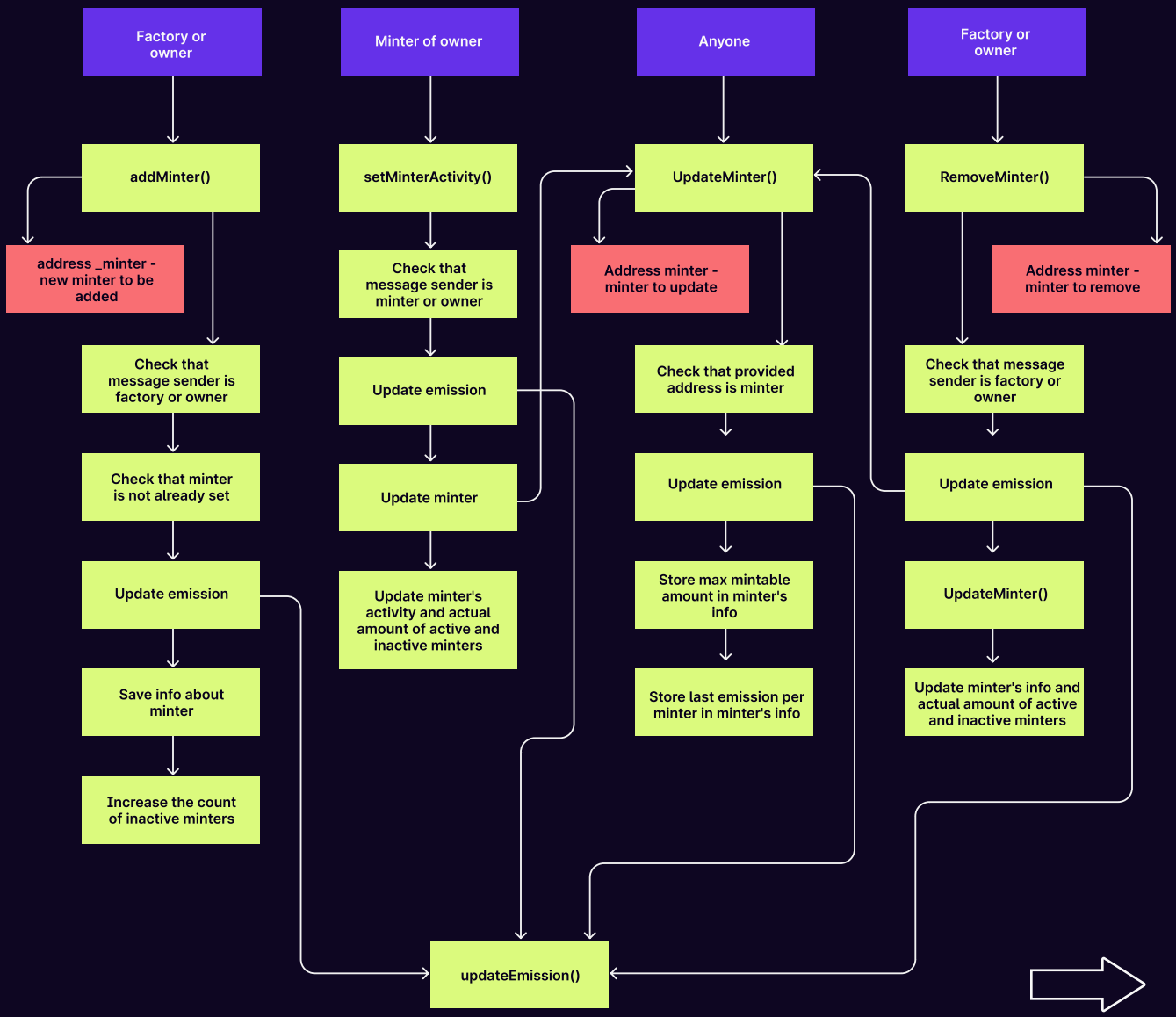
MYSTAKING.SOL



MYSHARE.SOL

MyShare is an ERC20 token with additional mint functionalities and fees on transferring tokens.
Tokens are issued during different epochs. Each epoch has its own emission amount per second.
Minters are to mint a part of the total emission.
Fees are applied to each transfer, where sender or receiver is not whitelisted.

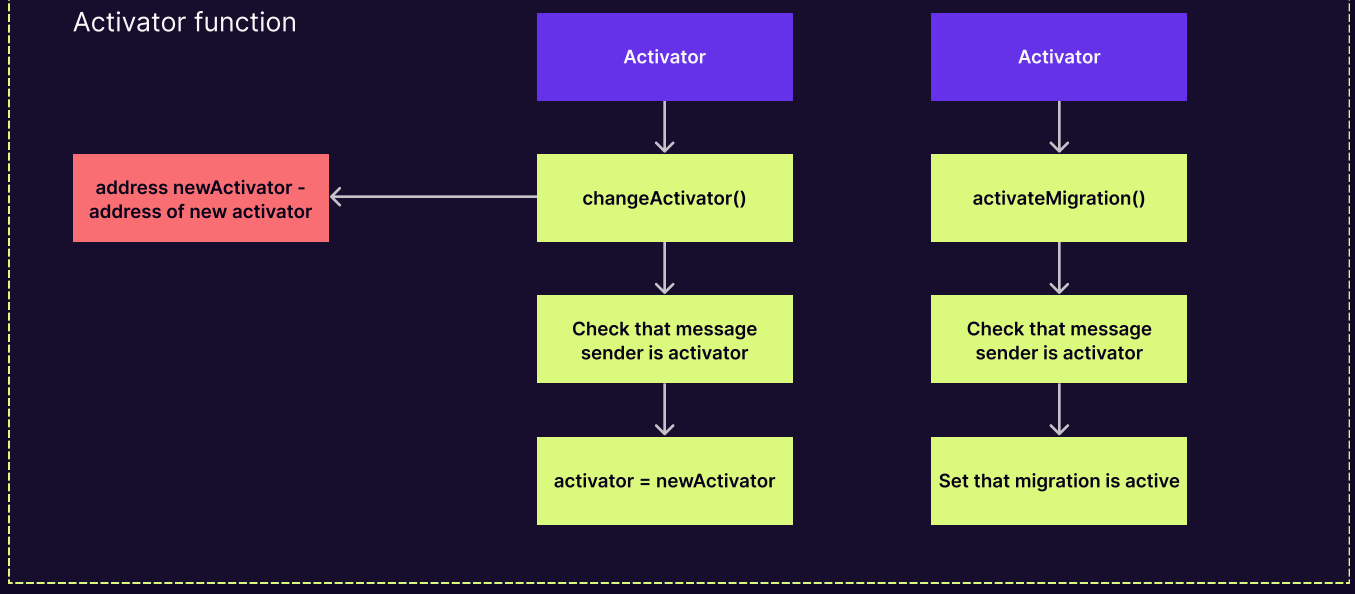
Managment of minters

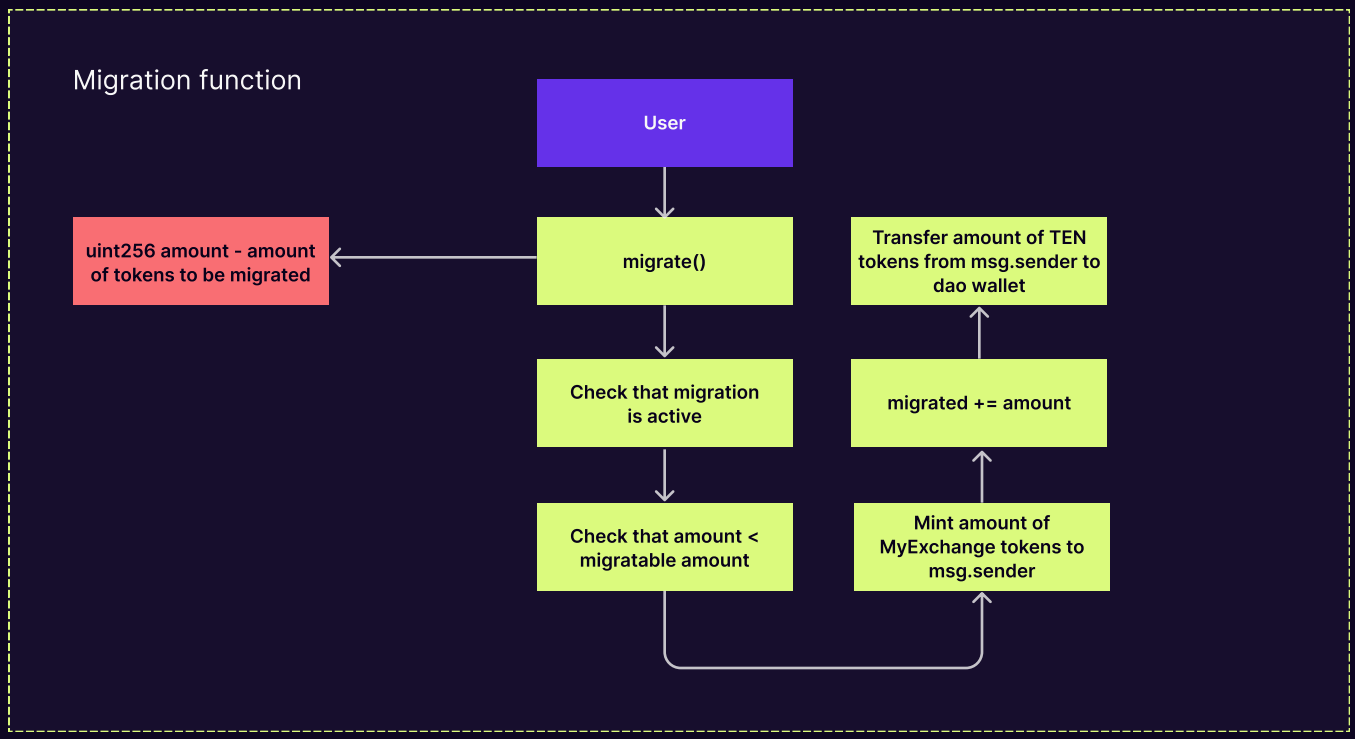


View functions

MYEXCHANGE.SOL

MyExchange is an ERC20 token with additional migration functionality. Once the activator has activated migrations, users can migrate their TEN tokens and receive MyExchange tokens in the ratio of 1:1. All TEN tokens are transferred to the DAO wallet. Up to 100 million TEN tokens can be migrated. 200 million are pre-minted at the creation of the contract to the activator's balance. Both TEN and MyExchange tokens have 18 decimals.

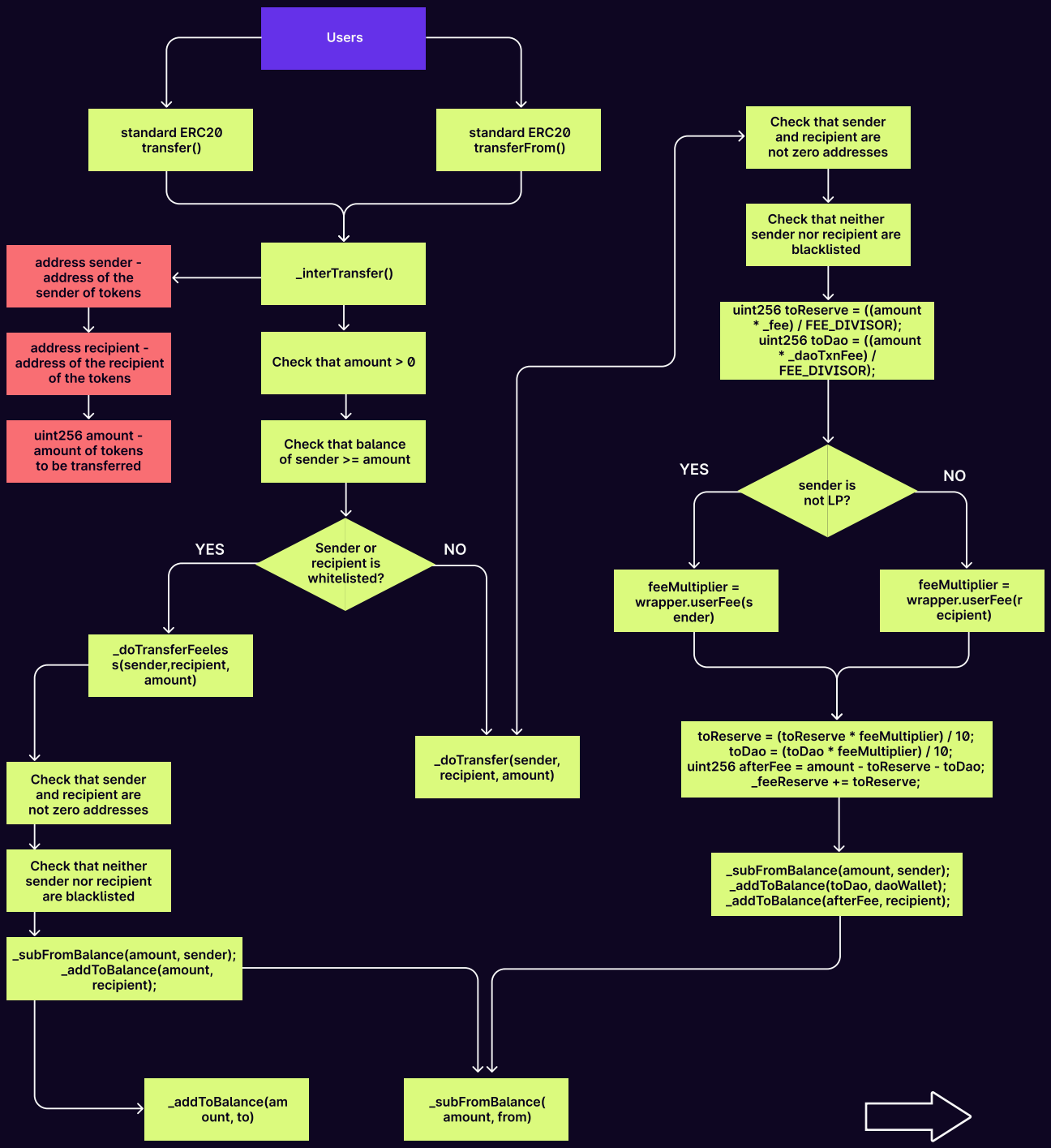




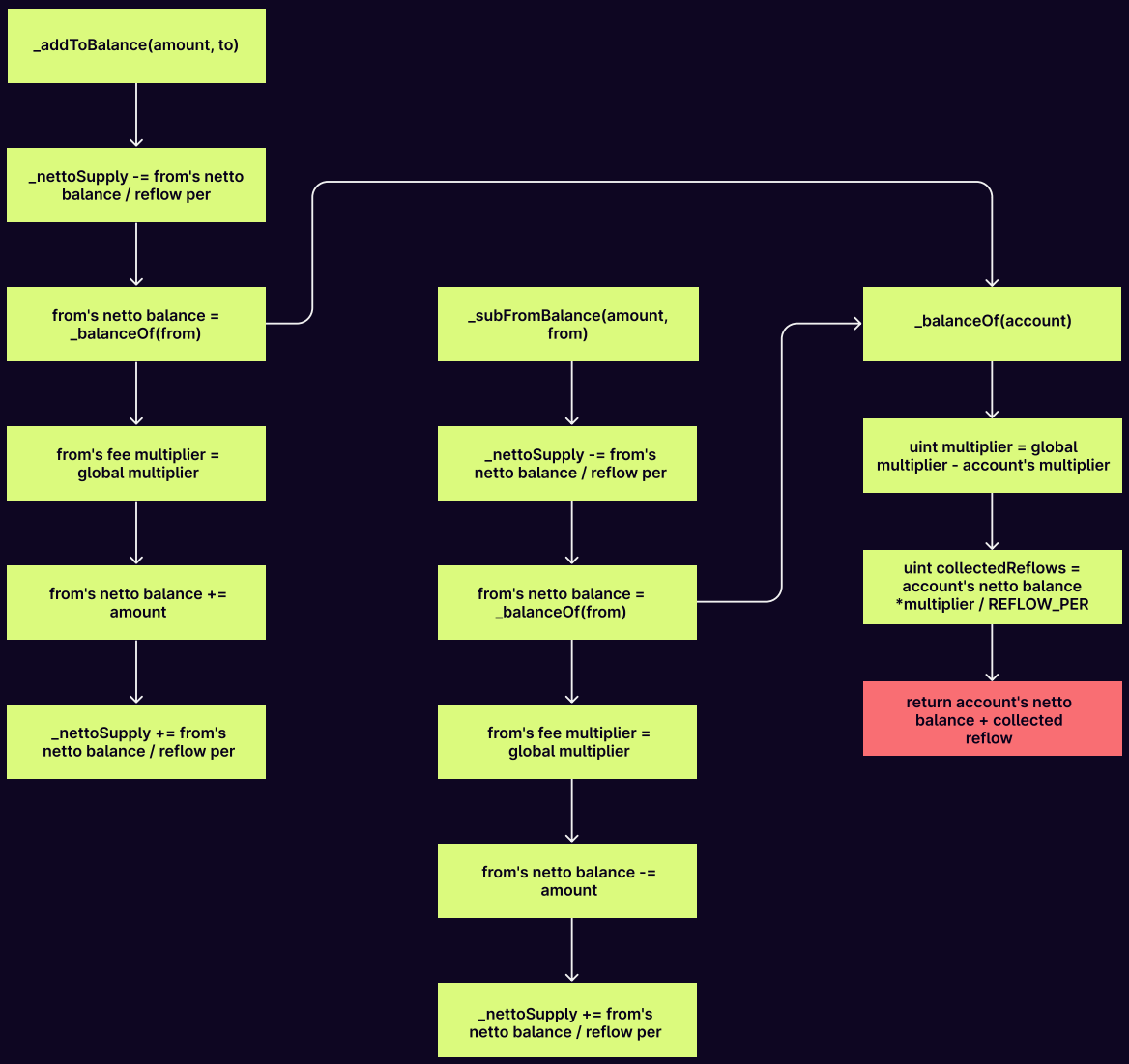
MYTOKEN.SOL

MyToken is an ERC20 token with an additional fee and reflow functionality. The fee is applied to each transfer, where none of the participants is whitelisted. The fee than can be reflowed and distributed among all the holders. The token can be minted and burnt by MyWrapper contract, the owner can set fees and wrapper contract.

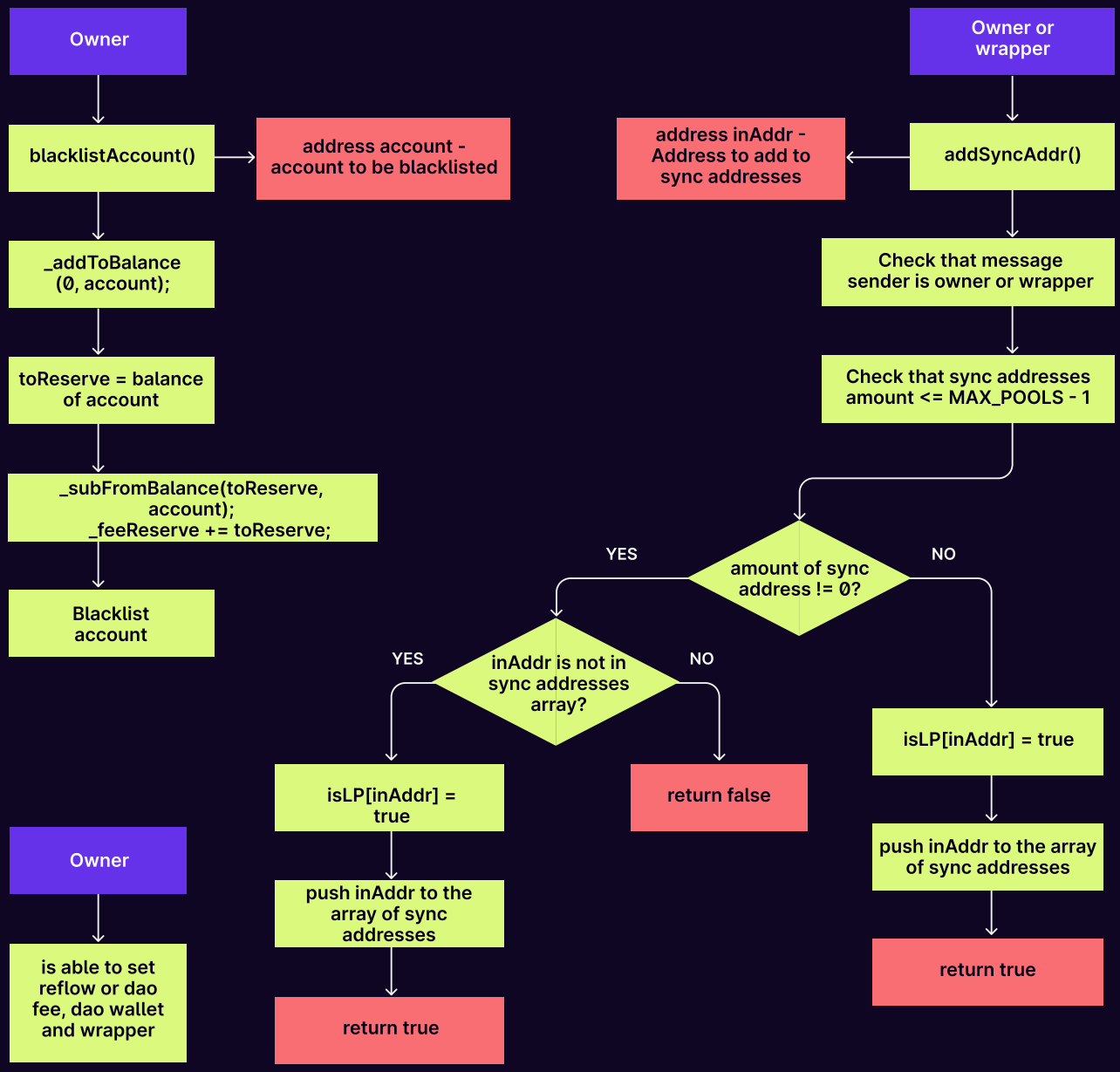
Transfer functions



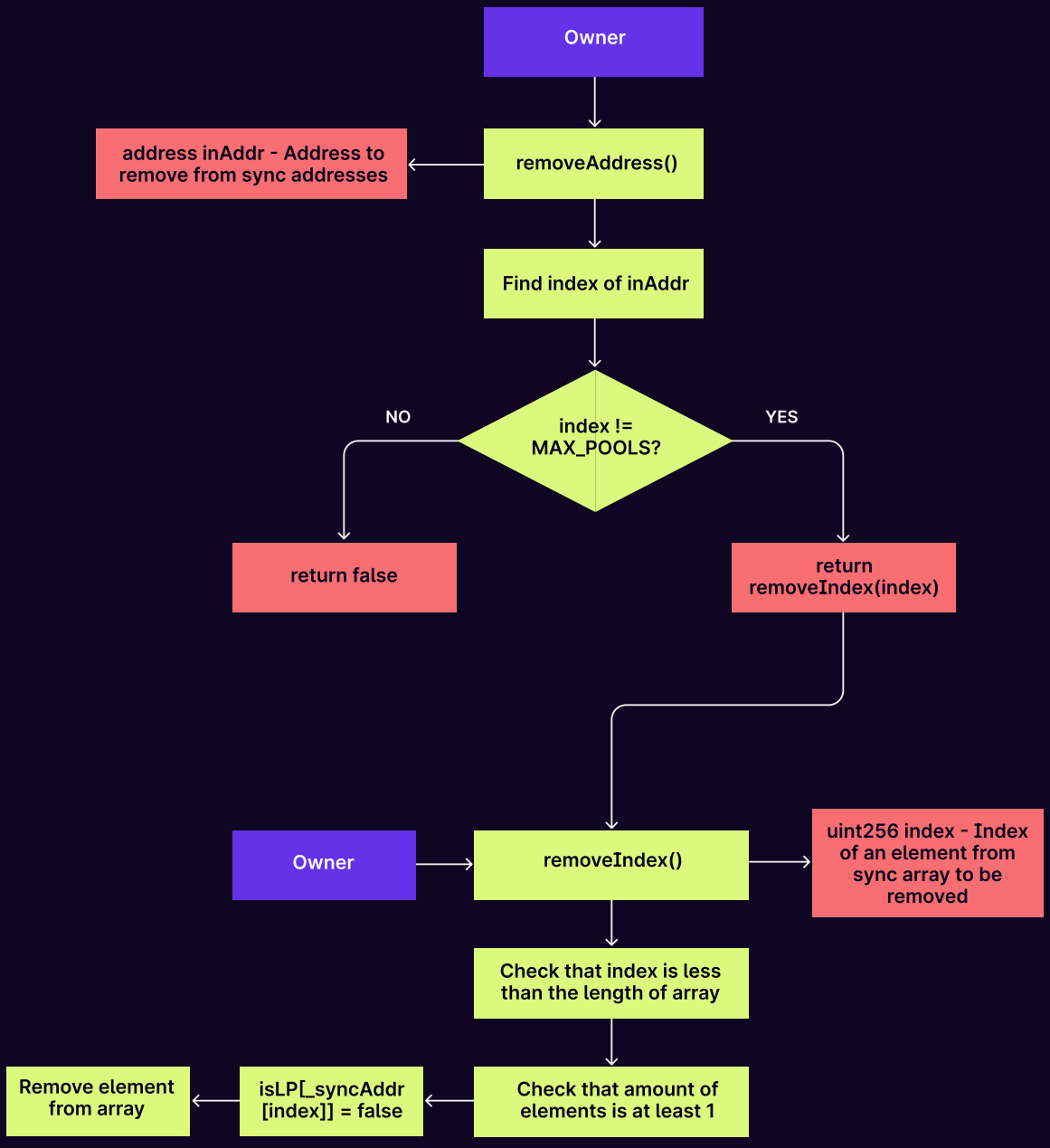
Balance internal functions



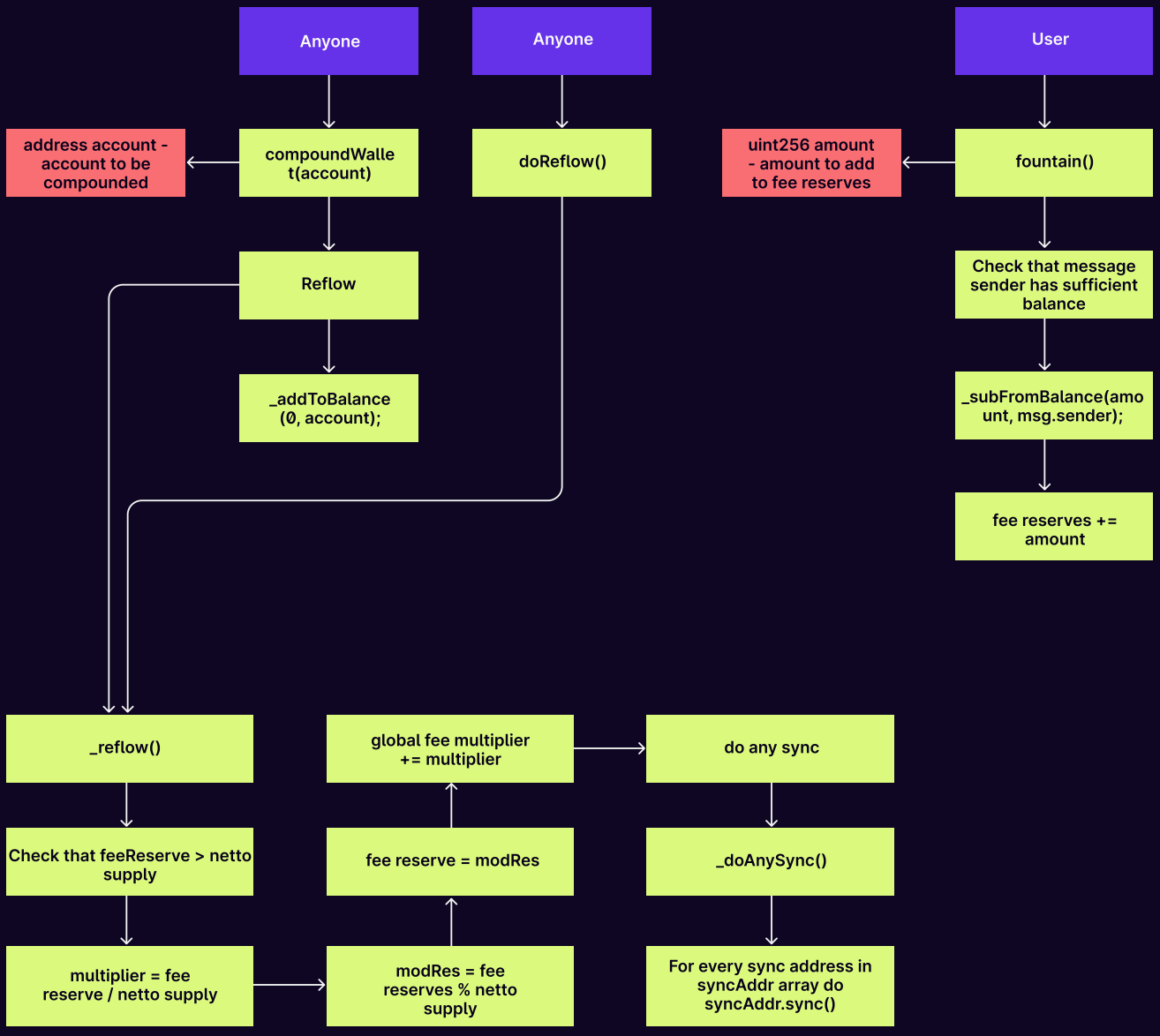
Restricted functions



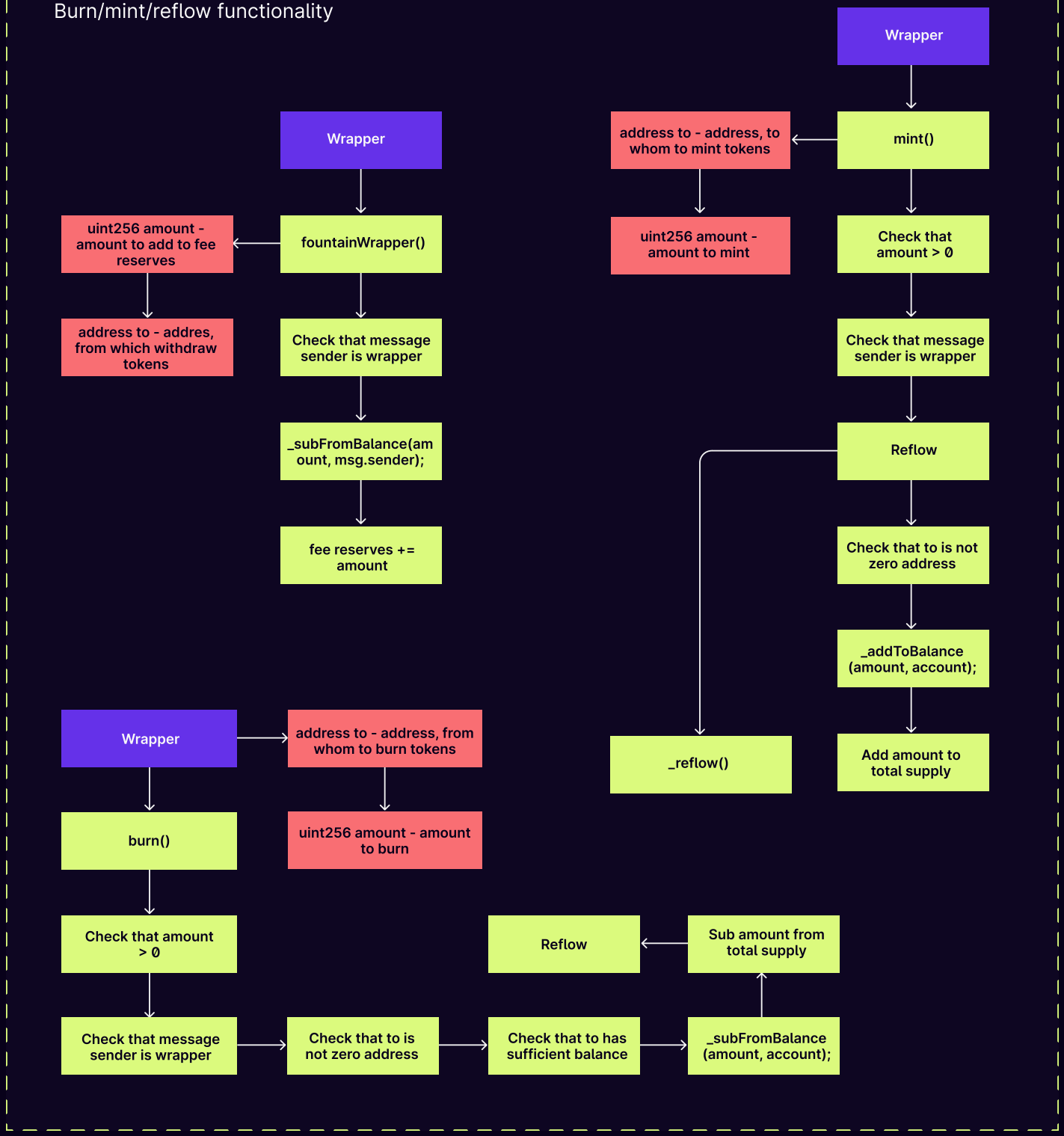
Restricted functions



Burn/mint/reflow functionality

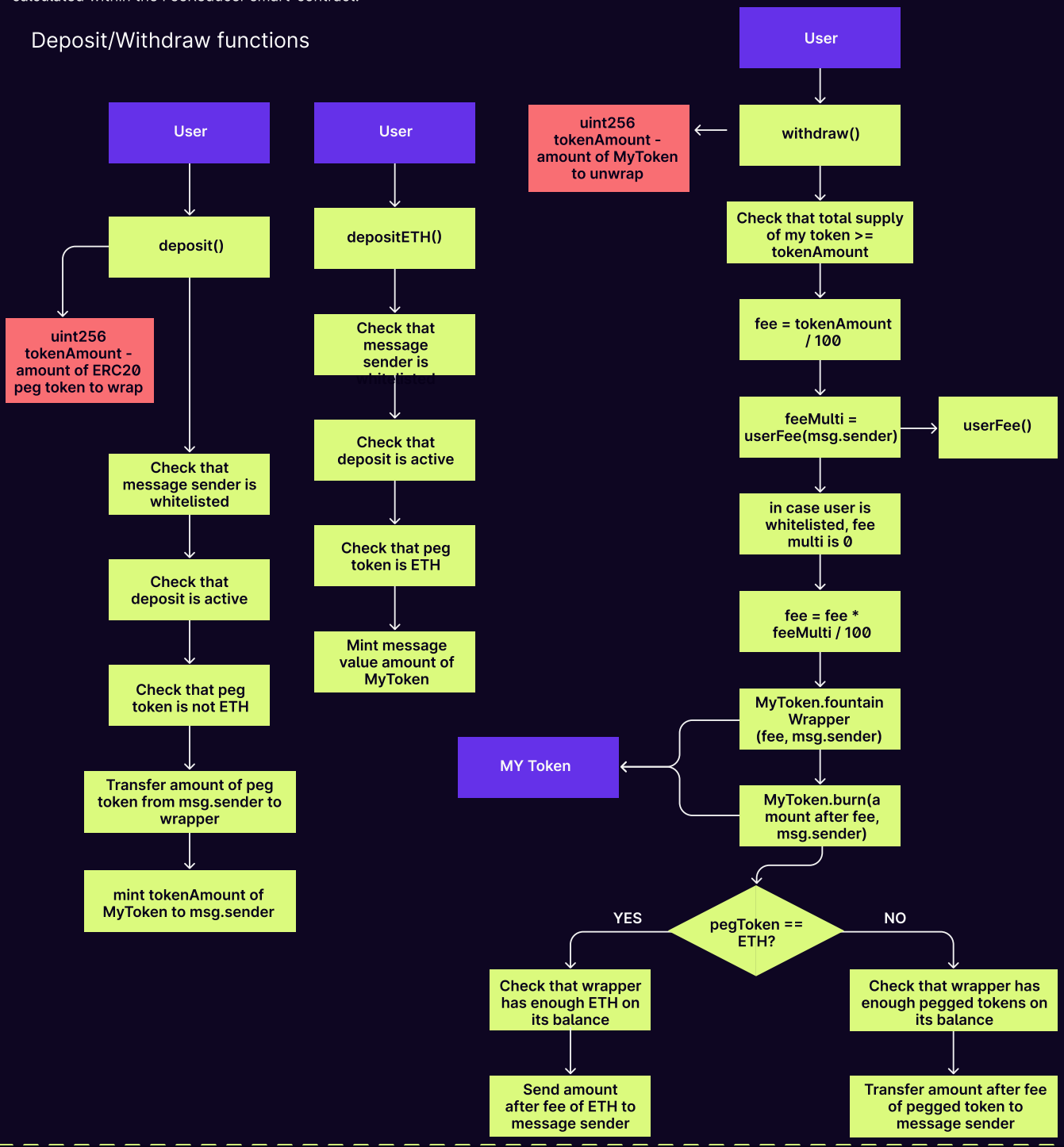


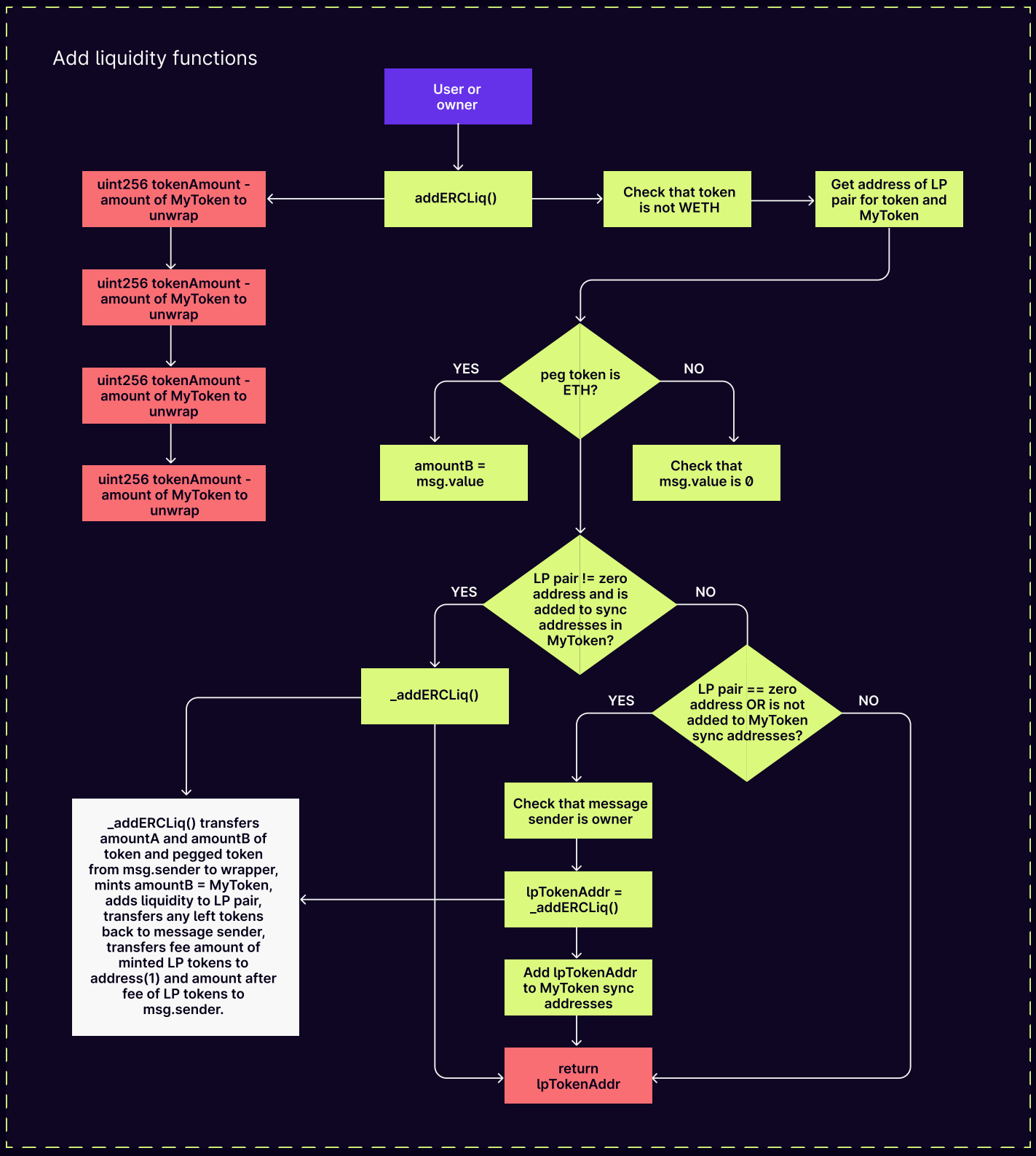
Burn/mint/reflow functionality



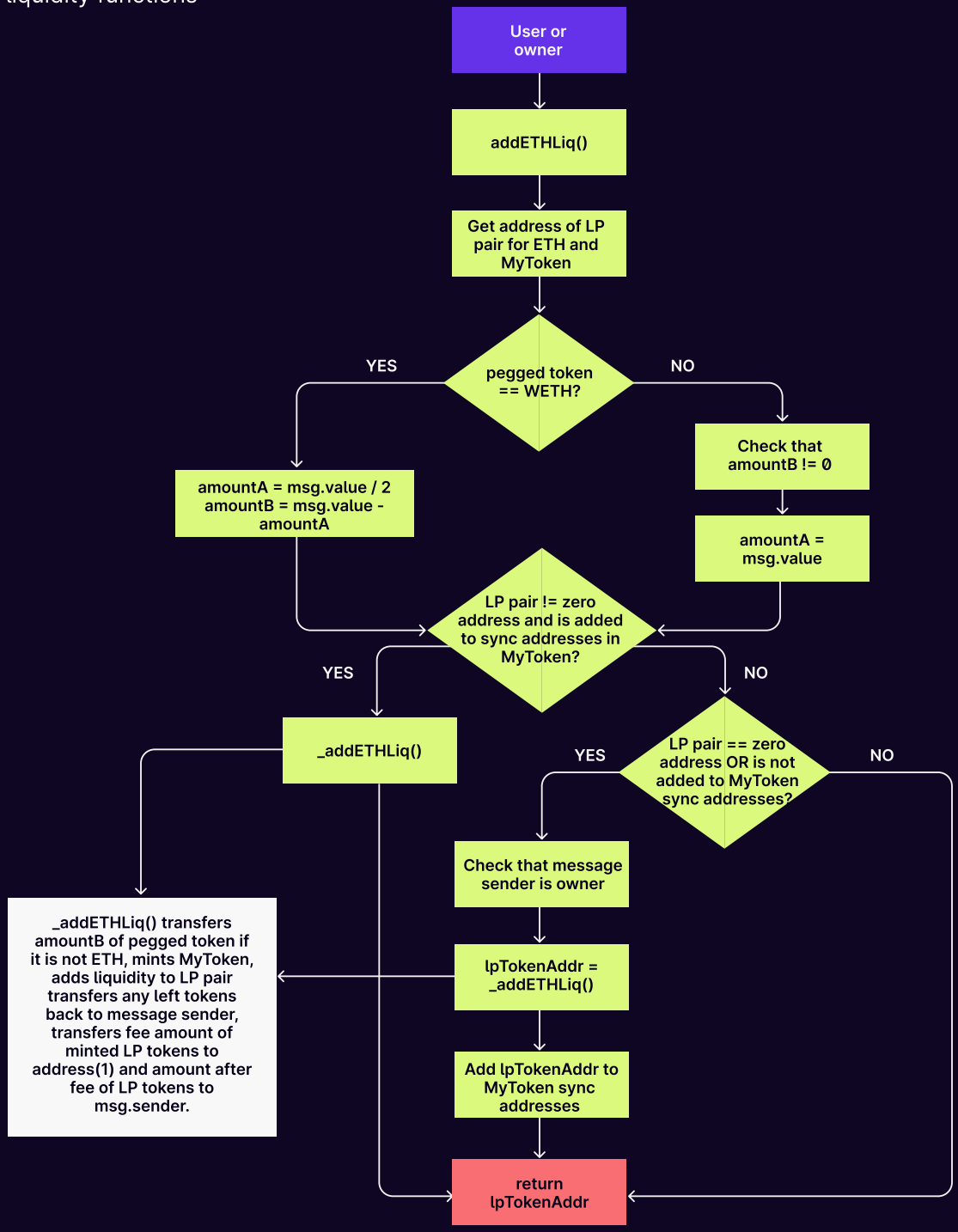
MYWRAPPER.SOL

MyWrapper is a contract for buying MyToken for pegged token in exchange of 1:1. Only whitelisted users can buy MyToken. Wrapped MyToken can be withdrawn in exchange for a pegged token. A fee is applied for every withdrawal in case the user is not whitelisted. Users can also add liquiidity to LP pairs with MyToken. This way, the users who were not whitelisted can receive MyToken. A fee is also applied when bying liquidity tokens. All fees are calculated within the FeeReducer smart-contract.



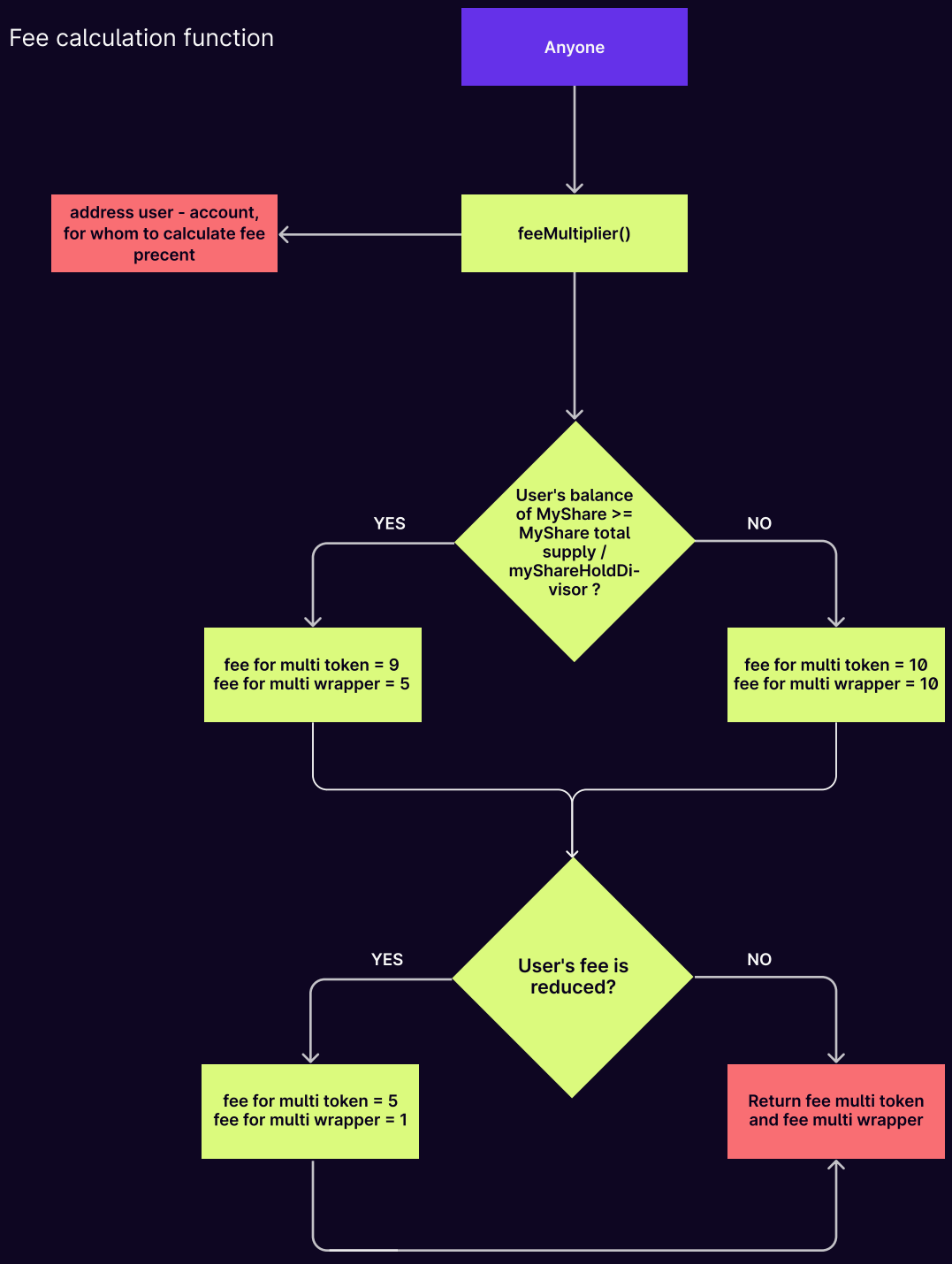

Add liquidity functions
User or owner
addERCLiq()
uint256 tokenAmount - amount of MyToken to unwrap
uint256 tokenAmount - amount of MyToken to unwrap
uint256 tokenAmount - amount of MyToken to unwrap
uint256 tokenAmount - amount of MyToken to unwrap
Check that token is not WETH
Get address of LP pair for token and MyToken
peg token is ETH?
YES
NO
amountB = msg.value
Check that msg.value is 0
LP pair != zero address and is added to sync addresses in MyToken?
YES
NO
_addERCLiq()
LP pair == zero address OR is not added to MyToken sync addresses?
YES
NO
Check that message sender is owner
lpTokenAddr = _addERCLiq()
Add lpTokenAddr to MyToken sync addresses
return lpTokenAddr
_addERCLiq() transfers amountA and amountB of token and pegged token from msg.sender to wrapper, mints amountB = MyToken, adds liquidity to LP pair, transfers any left tokens back to message sender, transfers fee amount of minted LP tokens to address(1) and amount after fee of LP tokens to msg.sender.

Add liquidity functions

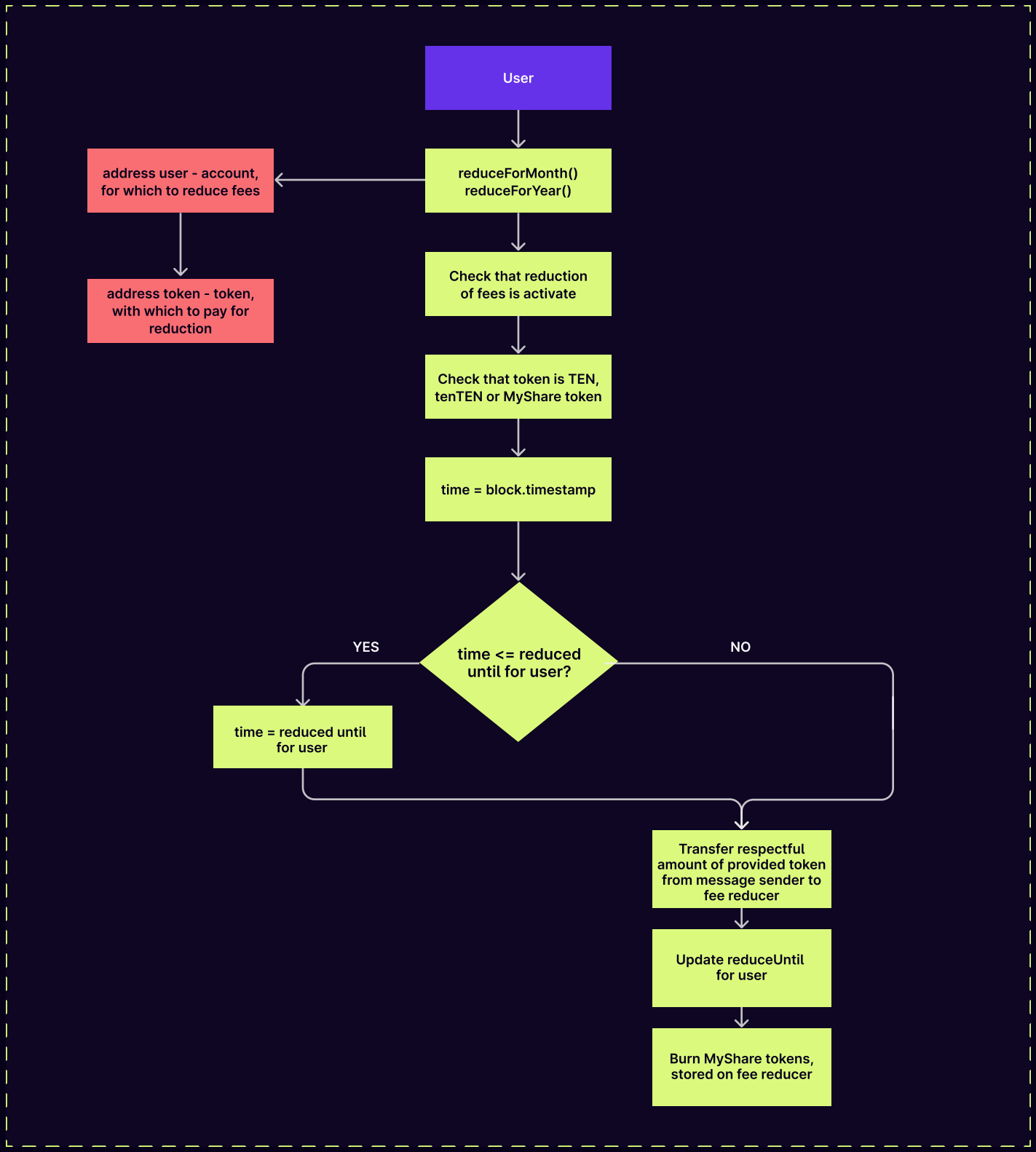


FEEREDUCER.SOL

FeeReducer is a contract, responsible for fee calculations across the protocol. Fees can be reduced for some period of time by paying ten, tenTEN or MyShare tokens.



FEE REDUCTION



STRUCTURE AND ORGANIZATION OF DOCUMENT

For easier navigation, sections are arranged from most critical to least critical. Issues are tagged “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 Customer’s side or is an issue that the Customer disregards as a problem. 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 it can lead to a significant loss, funds may be lost or allocated incorrectly.

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

CRITICAL | VERIFIED

The beginning of the stake is updated before rewards are calculated.
MyStaking.sol: function _stake().
“stakeBegin” is used within the function getDivisorByTime() during the calculation of the claimable amount. Since _setTimer() is called before rewards are calculated (Lines 243-244), “stakeBegin” will be equal `0` for msg.sender and the commission for the user will always be calculated with maximum fee.

Recommendation:
Update “stakeBegin” after rewards are calculated.

From the client:
The calculated rewards in _stake() are saved in userInfo[msg.sender].unclaimedAmount without the fee subtracted, so the user does not lose funds but the “fee-advantage”.

CRITICAL | RESOLVED

User’s rewards are stored incorrectly during staking.
MyStaking.sol: function _stake().
During staking, the rewards are calculated with function claimableAmount(). This function calculates current accrued rewards and adds “userInfo[user].unclaimedAmount” to the claimable amount. After claimable rewards are calculated, they are added to “userInfo[msg.sender].unclaimedAmount” (Line 245), though unclaimed amount is already calculated. This way, malicious users can illegally increase their rewards.

Recommendation:
Store rewards correctly so that there can’t be any extra rewards during staking.

CRITICAL | RESOLVED

Lock of user funds.
MyStaking.sol: function _withdraw().
When the user stakes, there is the increase of the counter `stakerCount` on line 227. After that, when the user redeems their funds, there is the subtraction of the counter in the function _withdraw() on line 319. However, there is no check that the user is a staker. Due to this, the user can call redeem() as many times as they like, thereby resetting the counter `stakerCount` and blocking withdrawals for all users.

Recommendation:
Add the validation that user's share is greater than 0, thus, verifying that the user is actually a staker.

HIGH | VERIFIED

A fee can be applied to the same rewards more than once.
MyStaking.sol: function claimableAmount().
The function calculates user's rewards and subtracts fees. However, the claimable amount is firstly calculated with the unclaimed amount and then a fee is applied to it. Since an unclaimed amount is stored with an already subtracted fee, the commission can be applied to user's rewards more than once, causing users to lose funds.

Recommendation:
Apply the fee only to rewards that don't include unclaimed amount.

From the client:
The amount without fees is already stored to unclaimed amount so that fees are applied only one time during claiming.

HIGH | RESOLVED

Deprecated ETH transfer.
MyStaking.sol: function daoWithdraw().
MyShare.sol: function withdrawToken().
MyExchange.sol: function withdraw().
MyWrapper.sol: Lines 201, 345, 463, 504, 530.
StakingFactory.sol: function withdrawToken().
Due to the Istanbul update, there were several changes provided to the EVM, which made .transfer() and .send() methods deprecated for the ETH transfer. Thus, it is highly recommended to use .call() functionality with mandatory result check or the built-in functionality of the Address contract from OpenZeppelin library.
Also, the contract has neither payable function nor receive() function implemented, thus it isn't supposed to store ETH on its balance.

Recommendation:
Correct the ETH sending functionality and verify its necessity in the contract.

HIGH | RESOLVED

Liquidity pool can be set more than once.
MyShare.sol: function setLP().
In the comment section, it is declared that a liquidity pool can be set only once, and the “txnLiqPoolIsSet” variable is checked to be false to verify this. However, the stake of this variable doesn’t change, meaning that it will always be equal to false and could be set more than once.

Recommendation:
Assign true to “txnLiqPoolIsSet” after a liquidity pool is set.

Post-audit:
In order to validate that a pool can only be set once, the “txnLiqPool” storage variable is checked not to be zero address.

HIGH | RESOLVED

A transfer to zero address might revert
MyStaking.sol: function _withdraw(), line 314.
Transferring fees to zero address might revert, depending on the implementation of ERC20. For example, standard implementation by OpenZeppelin has such a restriction. As a result, with some of the ERC20 tokens, withdrawal won't be possible. Thessue is not marked as critical since a custom implementation of ERC20 can be used, which allows transfers to zero address.

Recommendation:
Verify that the team will only use a custom implementation of ERC20 that allows transferring to zero address. Otherwise, transfer fees to the other address. For example, to address(1).

Post-audit:
The issue was resolved by transferring fees to address(1).

MEDIUM | VERIFIED

SafeERC20 should be used.
Transferring ofERC20 tokens across all the contrracts is performed with regular transfer() and transferFrom() methods from the OpenZeppeling IERC20 interface. Yet, in general, the ERC20 token may have no return value for these methods (see USDT implementation) and lead to the failure of calls and contract blocking. Thus, the SafeERC20 library should be used or the token should be verified to implement return values for transfer() and transferFrom() methods.

Recommendation:
Use the SafeERC20 library for transferring ERC20 tokens.

Post-audit:
SafeERC20 is used in all the necessary parts of the code.

MEDIUM | RESOLVED

The variable is written twice.
MyShare.sol: function endOfEpoch().
The “endTime” variable is written either in line 320 or in line 322 first time. After that it is written again in line 325.
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#write-after-write

Recommendation:
Verify the correctness of the function so that the value of “endTime” is calculated correctly.

LOW | RESOLVED

Unused storage variables and constants.
MyStaking.sol: variables myToken, pegToken, pairedToken, stakingFactory, router, zapPath.
MyStaking.sol: constants WETH.

Recommendation:
Remove unused variables.

LOW | RESOLVED

Staker count is not calculated correctly.
MyStaking.sol: function _stake(), line 237.
stakerCount is increased every time anyone stakes tokens, including the cases where msg.sender is already a staker. This way, the variable can contain the wrong number of stakers. The issue is marked as low since stakerCount is not used within the contract and doesn’t take part in any calculations. However, it might be used by the dApp.

Recommendation:
Increase stakerCount only in case msg.sender is not already a staker.

LOW | VERIFIED

Route is not verified.
FeeReducer.sol: function buyBackBurn().
The function is supposed to swap TEN token and burn the balance of MyShare token. Thus, the route is supposed to start with TEN token and end with MyShare token and should be validated.

Recommendation:
Verify that the first element in the route is TEN token and the last one is MyShare token.

From the client:
This feature is intended so that it is possible for the DAO to buy back and burn other tokens than TEN if they were sent to the contract.

LOW | RESOLVED

The total supply of MyShare token is checked when setting TEN burn amount.
FeeReducer.sol: function setTenBurnFixAmount(), line 121.
In this function, the total supply of MyShare token is checked instead of the TEN token. However, the function sets a burn amount for the TEN token.
The issue is marked as low since it is unclear whether MyShare or TEN token's supply should be checked. Thus, the issue should be verified by the team.

Recommendation:
Verify which token's supply should be checked and correct the validation in case TEN token's supply should be checked instead of MyShare.

INFORMATIONAL | RESOLVED

The owner can withdraw any ERC20 tokens.
MyStaking.sol: function emergencyWithdraw().
The owner can withdraw any tokens from the contract, including the stake token. Even though it is declared in the comment section that this function will be removed after the audit, we recommend to remove this function during the audit or restrict the owner from withdrawing the stake token.

Recommendation:
Remove the function or add a restriction that the owner can't withdraw the stake token.

Post-audit:
The function was removed.

INFORMATIONAL | RESOLVED

The beginning of the stake is performed twice during staking.
MyStaking.sol: function _stake().
The call of the _claim(), function _setTimer() function updates "stakeBegin" variable in user's info (Line 243). The same action is performed in line 248. The issue doesn't affect code security but increases the gas spendings for users.

Recommendation:
Update "stakeBegin" only once during function _stake().

INFORMATIONAL | RESOLVED

Unnecessary validations.
MyToken.sol: function _doTransferFeeless().
The function _doTransferFeeless() is internal and is called only within the _interTransfer() function. The function performs the same validations as _interTransfer() (Lines 314-315 in _interTransfer() and Lines 375-376 in _doTransferFeeless()), which is unnecessary and only increases gas spendings.
Also, the validations that are performed in function _doTransfer() (Lines 336-339) are not performed in _doTransferFeeless().

Recommendation:
Remove unnecessary validations. Verify that same validations as in _doTransfer() should not be performed in _doTransferFeeless().

Post-audit:
Unnecessary validations were removed from function _doTransferFeeless().

INFORMATIONAL | RESOLVED

Unnecessary gas spendings.
MyToken.sol: function addSyncAddr().
In order to determine whether the provided “inAddr” parameter is present in the “_syncAddr” array, an iteration through the whole storage array is performed to find inAddr. However, the value from mapping “isLP” can be checked for “inAddr”. This way, gas spendings for the function can be significantly decreased.

Recommendation:
Read value from mapping instead of iterating through the storage array.

INFORMATIONAL | RESOLVED

Unused function.
MyShare.sol: function _mintMinter(), line 501.
The function is not used within MyShare contract. Though it is internal and can be used by children contracts, such contracts are not present within the scope of the audit.

Recommendation:
Remove the unused function or verify that it is needed for children contracts.

INFORMATIONAL | UNRESOLVED

The owner can blacklist any account.
MyToken.sol: function blacklistAccount().
In the comment section it is declared that the owner can blacklist a dead wallet to prevent it from collecting fees. However, it is not checked whether the provided account is active or inactive, which means that the owner can put any account to blacklist. The issue is marked as critical since such functionality allows the owner to withdraw funds from any account.

Recommendation:
Consider adding a mechanism of verifying whether an account is truly inactive and should be blacklisted.

Post-audit:
The issue was verified by the Magic Yearn team as a part of the protocol. It is covered with appropriate role and the team will provide necessary actions to cover all possible back-door usage and active users tracking. Yet, this issue should be still placed in the report, so it was marked as Informational.

Additional optimizations.
The following optimizations were discovered during unit-testing and should be considered as suggestions:
MyShare.sol: function _checkTransfer(), line 784.
The “amount” parameter should be validated not to be 0.

Recommendation:
Add a validation that “amount” is not equal to 0.

MyShare.sol: function _burn(), line 850.

Validating that parameter “account” is not a zero address is unnecessary, since the _burn() function is only called within functions _transfer() and burn(). In the _transfer() function, such validation is already performed in line 689. In the burn() function, the address of the message sender is taken, which cannot be zero.

Recommendation:
Remove the unnecessary validation.

The wrong revert messages in functions setMinterActivity() (Line 212), removeMinter() (Line 242), setIsBlacklisted() (Line 106). In line 212, the revert message is about adding a minter, though the function sets the activity of the minter. The same is in the function that removes a minter. In line 106, the revert message is about whitelist, whereas the function sets value to blacklist.

The removeMinter() function lacks validation that the provided “minter_” parameter is actually a minter.

MyExchange.sol: function migrateable().

The function can never reach the code in the “else” branch, since “migrated” can’t be greater than “MAX_MIGRATE” and in the case when “migrated” is equal to “MAX_MIGRATE”, it will return 0 as a result of subtraction in line 64.

Recommendation:
Remove the “else” branch.

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

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

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

MyWrapper.sol	
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	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
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Magic Yearn in verifying the correctness of their contract 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 the review of the Magic Yearn contract requirements for details about issuance amounts and how the system handles these.

FeeReducer

- ✓ sets new router
- ✓ sets MyShare hold divisor (50ms)
- ✓ sets MyShare burn divisor
- ✓ sets MyShare burn fix amount
- ✓ sets Ten burn fix amount (1000ms)
- ✓ burns tokens from contract (49ms)
- ✓ swaps TenTen tokens for Ten token (2508ms)
- ✓ reduces fee for month (1888ms)
- ✓ reduces fee for year (200ms)
- ✓ withdraws tokens
- ✓ dao withdraws ethereum (193ms)

MyExchange

- ✓ Change activator
- ✓ Activate migration and check migratable amount (57ms)
- ✓ withdraw tokens
- ✓ dao withdraw ethereum (194ms)

MyShare

- ✓ sets lp address
- ✓ sends user to black list
- ✓ sends user to white list
- ✓ changes Dao Wallet (44ms)
- ✓ sets lp address
- ✓ sets fee
- ✓ sets factory
- ✓ adds minter
- ✓ sets minter activity (90ms)
- ✓ removes minter (129ms)
- ✓ checks active minters (56ms)
- ✓ checks last emission of minter
- ✓ checks emission per minter per second

- ✓ checks end of epoch
- ✓ checks actual epoch (41ms)
- ✓ checks emission of epoch
- ✓ checks emission per second of epoch
- ✓ checks max mintable for minter
- ✓ has transfer (129ms)
- ✓ has transferFrom (155ms)
- ✓ has increase/decrease allowance
- ✓ has mint/burn (73ms)
- ✓ withdraw tokens
- ✓ dao withdraw ethereum (211ms)

MyStaking

[Activation of a staking]

- ✓ Activates (52ms)
- ✓ Activates if no need to emit or calculate the emission (237ms)
- ✓ Deactivates (87ms)
- ✓ Prevents non-owners from (de)activation of the contract

[Staking]

- ✓ Stakes (91ms)
- ✓ Stakes twice (166ms)
- ✓ Reverts when staking if the pool is not active (54ms)
- ✓ Reverts when staking if too small amount

[Claim]

Claims

- ✓ when the `PERIOD0` period (of one day) (72ms)
- ✓ when the periods from `PERIOD0` to `PERIOD5`. (Here are 65 days taken) (588ms)

[Redemption]

Redeems

- ✓ when the `PERIOD0` period (of one day) (107ms)
- ✓ when the periods from `PERIOD0` to `PERIOD5`. (Here are 80 days taken) (710ms)
- ✓ Reverts when redemption if a user is not a staker (83ms)

emergency without a claim

- ✓ when the `PERIOD0` period (of one day) (49ms)
- ✓ when the periods from `PERIOD0` to `PERIOD5`. (Here are 80 days taken) (290ms)

[DAO withdrawal]

- ✓ Withdraws ERC20 tokens which are airdropped or send by mistake to the contract (66ms)
- ✓ Reverts when withdrawal of staked tokens
- ✓ Prevents non-owners from withdrawal of ERC20 tokens which are airdropped or send by mistake (50ms)
- ✓ Withdraws Ether which are airdropped or send by mistake to the contract
- ✓ Prevents non-owners from withdrawal of Ether which are airdropped or send by mistake
- ✓ Reverts when withdrawal of Ether if failed sending

MyToken

- ✓ set white list
- ✓ sync with UniswapV2 pair (2014ms)
- ✓ has allowance (50ms)
- ✓ transfer (93ms)
- ✓ black list account (509ms)
- ✓ get fee multiplier for account
- ✓ get global fee multiplier
- ✓ get/set reflow fee
- ✓ get/set dao fee
- ✓ get transaction fee
- ✓ get actual fee
- ✓ get reserve fee
- ✓ set dao wallet
- ✓ compound wallet
- ✓ reflow
- ✓ fountain tokens (76ms)
- ✓ burn tokens (58ms)
- ✓ add/remove/find address (151ms)
- ✓ withdraw tokens
- ✓ dao withdraw ethereum (188ms)
- ✓ transfer token and check fee (57ms)
- ✓ increase dao fee and transfer token (64ms)

MyWrapper

- ✓ set fee reducer
- ✓ set staking factory
- ✓ return myToken address
- ✓ set white list
- ✓ deposit/withdraw (234ms)
- ✓ check if fee reduced for user
- ✓ check users fee
- ✓ add sync address to MyToken
- ✓ add liquidity (4646ms)
- ✓ change wrapper (65ms)
- ✓ dao withdraw tokens
- ✓ dao withdraw ethereum (212ms)

StakingFactory

[Creating of a staking]

- ✓ Creates (246ms)
- ✓ Reverts when creating if the LP token has a staking
- ✓ Prevents non-owners from creating

[Change of a staking]
- ✓ Changes (267ms)
- ✓ Prevents non-owners from changing

[Management of minters]
- ✓ Adds a minter
- ✓ Prevents non-owners from adding
- ✓ Removes a minter (46ms)
- ✓ Prevents non-owners from removing

[Withdrawal]
- ✓ Withdraws ERC20 tokens (532ms)
- ✓ Prevents non-owners from withdrawal of ERC20 tokens (50ms)
- ✓ Withdraws Ether which are airdropped or send by mistake to the contract
- ✓ Prevents non-owners from withdrawal of Ether which are airdropped or send by mistake
- ✓ Reverts when withdrawal of Ether if failed sending

[Extra tests for MyWrapper]
[Adding of liquidity for a `myToken` to the pancake swap with ERC20 tokens]
- ✓ Adds if an LP is not a sync address of the `myToken` (2445ms)
- ✓ Prevents non-owners from adding if an LP is not a sync address of the `myToken` (67ms)

[Adding of liquidity for a `myToken` to the pancake swap with Ether]
- ✓ Adds if an LP is not a sync address of the `myToken` (1249ms)
- ✓ Prevents non-owners from adding if an LP is not a sync address of the `myToken` (59ms)

[Withdrawal]
- ✓ Reverts when withdrawal of tokens if too few locked pegged tokens (94ms)
- ✓ Reverts when withdrawal of Ether if too few locked pegged tokens (74ms)
- ✓ Reverts when withdrawal of Ether if it is failed to send Ether (323ms)

[Viewers]
- ✓ Returns false when check of a reduction if the address is an LP of the `myToken` (366ms)

[Change of the wrapper]
- ✓ Reverts when change if it is failed to send Ether (437ms)

[DAO withdrawal]
- ✓ Reverts when withdrawal of the ERC20 staked tokens

118 passing (48s)

FILE	% STMTS	% BRANCH	% FUNCS
MyStaking.sol	100	100	100
MyExchange.sol	100	100	100
MyShare.sol	100	92.86	100
MyToken.sol	99.39	92.42	98.18
FeeReducer.sol	100	95.65	100
StakingFactory.sol	100	100	100
MyWrapper.sol	82.43	67.39	100
All files	**96.33**	**87.64**	**99.42**

Zokyo Security has prepared unit-tests with additional scenarios during the audit. Unit-tests were written to validate the main logic of the contract and cover as much test scenarios as possible. All the tests were performed on the fork of BNB Smart chain to ensure that the contracts will operate as intended in the conditions close to the live mainnet. Additionaly, the team has verified the corectness of the integration between the MyWrapper contract and the Pancakeswap protocol.

We are grateful for the opportunity to work with the Magic Yearn 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 Magic Yearn team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo