SMART CONTRACT AUDIT

zokyo

December 1st 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.

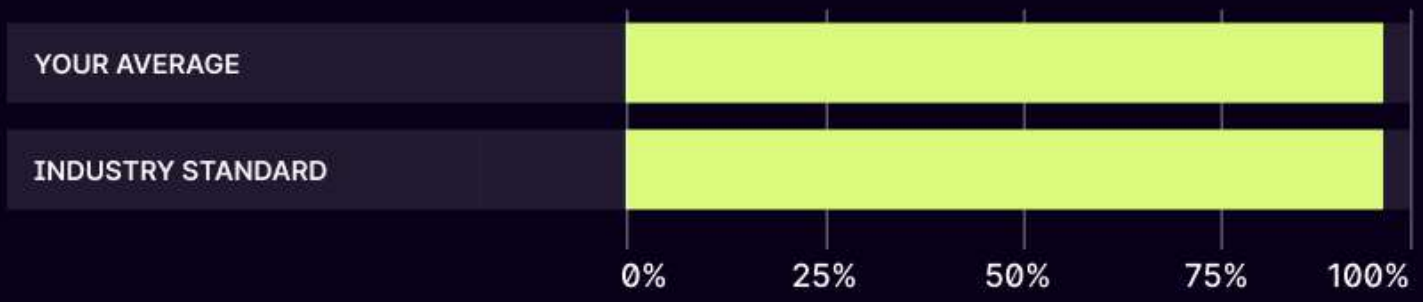
TECHNICAL SUMMARY

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

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

Contract Status

Testable Code



The testable code is 95%, which corresponds to the industry 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, Zokyo Security recommends that the Magpie team launches e a bug bounty program to encourage further and 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 Magpie repository:
https://github.com/magpiexyz/magpie_contracts

Initial commit: 7a64966e3297bd1aa620a267a3d0b2489d9eb200

Final commit: f8978e2686214ee45bef58f9e260807f56c70aef

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

- contracts\rewards
- contracts\wombat
- contract\Mgp.sol
- contracts\ProxyAdmin.sol
- contracts\TransparentUpgradeableProxy.sol
- contract\VLMGP.sol

The source code of the smart contract was taken from the Magpie repository:
https://github.com/magpiexyz/magpie_contracts/tree/bribery

Initial commit: 0476cfb1e4110c1f2f297857e803f20cc31657cc, bribe branch

Final commit: https://github.com/magpiexyz/magpie_contracts/pull/64, 0abe005f3822578a842249a2906a7329276a053e. https://github.com/magpiexyz/magpie_contracts/pull/67, 070b1bab3450e05b603822d89cd022fd9817f046.

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

- VLMGP.sol (13 changed lines of code)
- BNBZapper.sol
- contract\Mgp.sol
- BribeRewardPool.sol
- WombatBribeManager.sol
- WombatStaking.sol (~ 90 changed lines of code, compared to WombatStaking)
- MasterMagpie.sol
- SmartWomConvert.sol
- ManuallCompound.sol

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 resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Magpie smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor 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 consisted mostly 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

During the audit, Zokyo Security audited the whole set of contracts provided by the Magpie team. The protocol is built on top of the Wombat exchange and provides boosted yield to users, as well as a staking platform for earning extra rewards.

The goal of the audit was to validate the security of contracts against the list of common security vulnerabilities, verify that contracts interact with the Wombat exchange in the most secure and optimized way, and check that the best Solidity practises are applied in smart contracts.

The team of auditors did not detect any high-severity issues during the audit. Most of the issues, which are not connected to the admin capabilities, were succesfully fixed by the Magpie team. However, there are still some minor issues that weren't verified or resolved by the team, which might lead to the loss of funds only in case of any specific actions by the contract admins.

The overall security of the contracts is high enough. The contracts contain a good natspec documentation and are well-tested by the Magpie team. Zokyo Security has prepared their own set of unit tests, which are maintained on the BSC mainnet-fork in order to verify the correctness of interaction with the Wombat exchange. There weren't any critical issues found as well and all the tests passed after most issues were resolved.

At the end of the audit, the Magpie team also asked the team of auditors to check the gas optimization applied in MasterMagpie.sol. During its validation, an Info-9 issue was added to the report. The last audited commit is f8978e2686214ee45bef58f9e260807f56c70aef.

Executive Summary

During the second audit iteration, Zokyo Security audited all the changes made in the contracts and the entire new contracts. The goal of the second-iteration audit was to verify that the changes within the contract do not contain any high-severity issues and also verify the correctness of the new contracts.

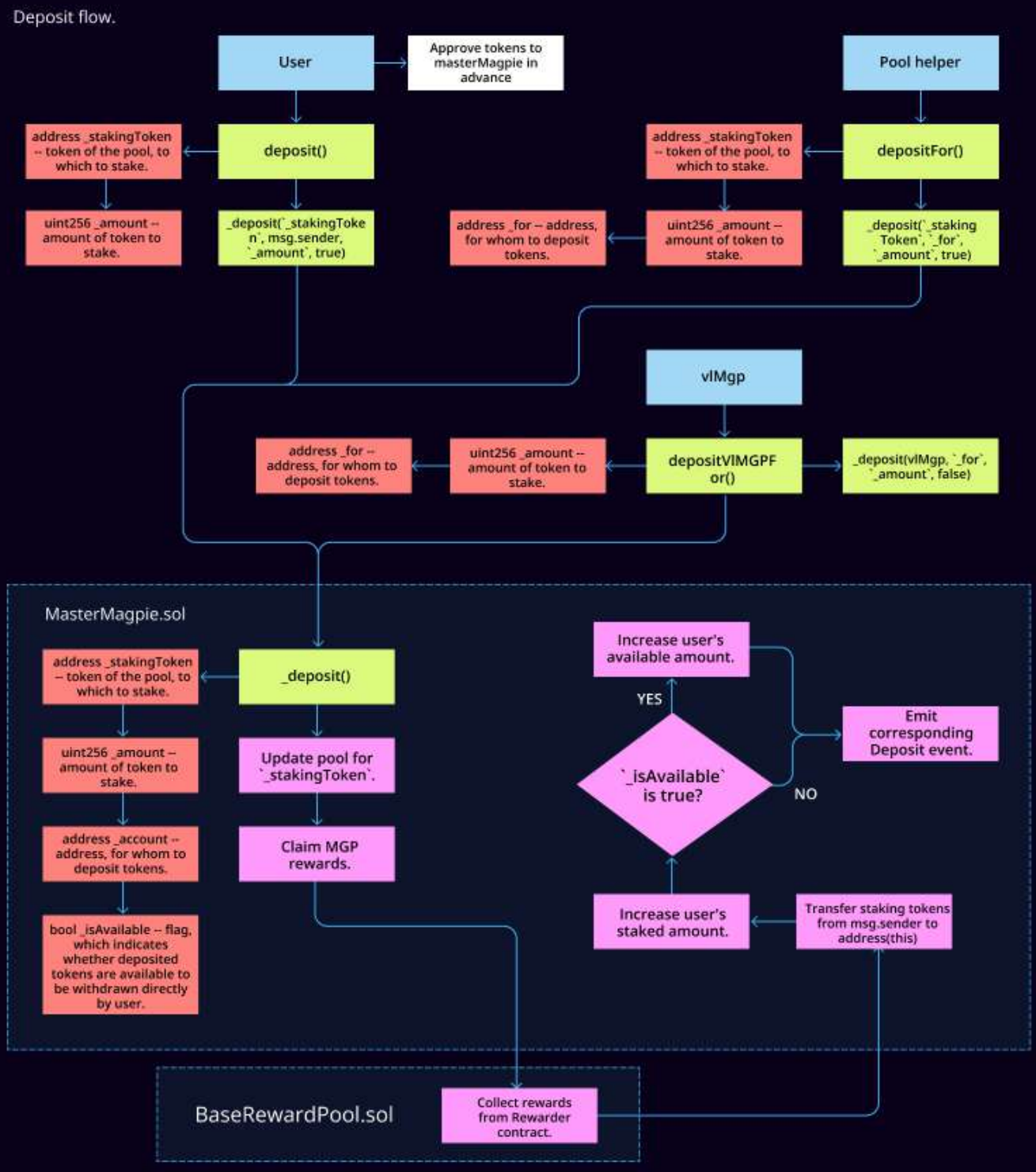
During the manual part of the audit, 3 high-severity issues were found. All of them were succesfully fixed by the Magpie team. All medium, low and info issues were fixed as well. It should be mentioned that one of the high-severity issues were connected to the possible frontrun attack when swapping assets, as minimum output amount parameter was ignored. Though the issue was fixed by providing this value as a function parameter, there is a still a swapping of rewards performed in the protocol without passing a valid minimum output parameter. Magpie XYZ team has verified this concern so that there won't be any large values of tokens swapped, so that a potential loss in case of attack will be negligible.

The Zokyo Security team has also prepared their own set of unit tests in order to validate the core logic of the contracts. The tests were written on the fork of the BSC mainnet network to validate that contracts interact with other protocols such as PancakeSwap.

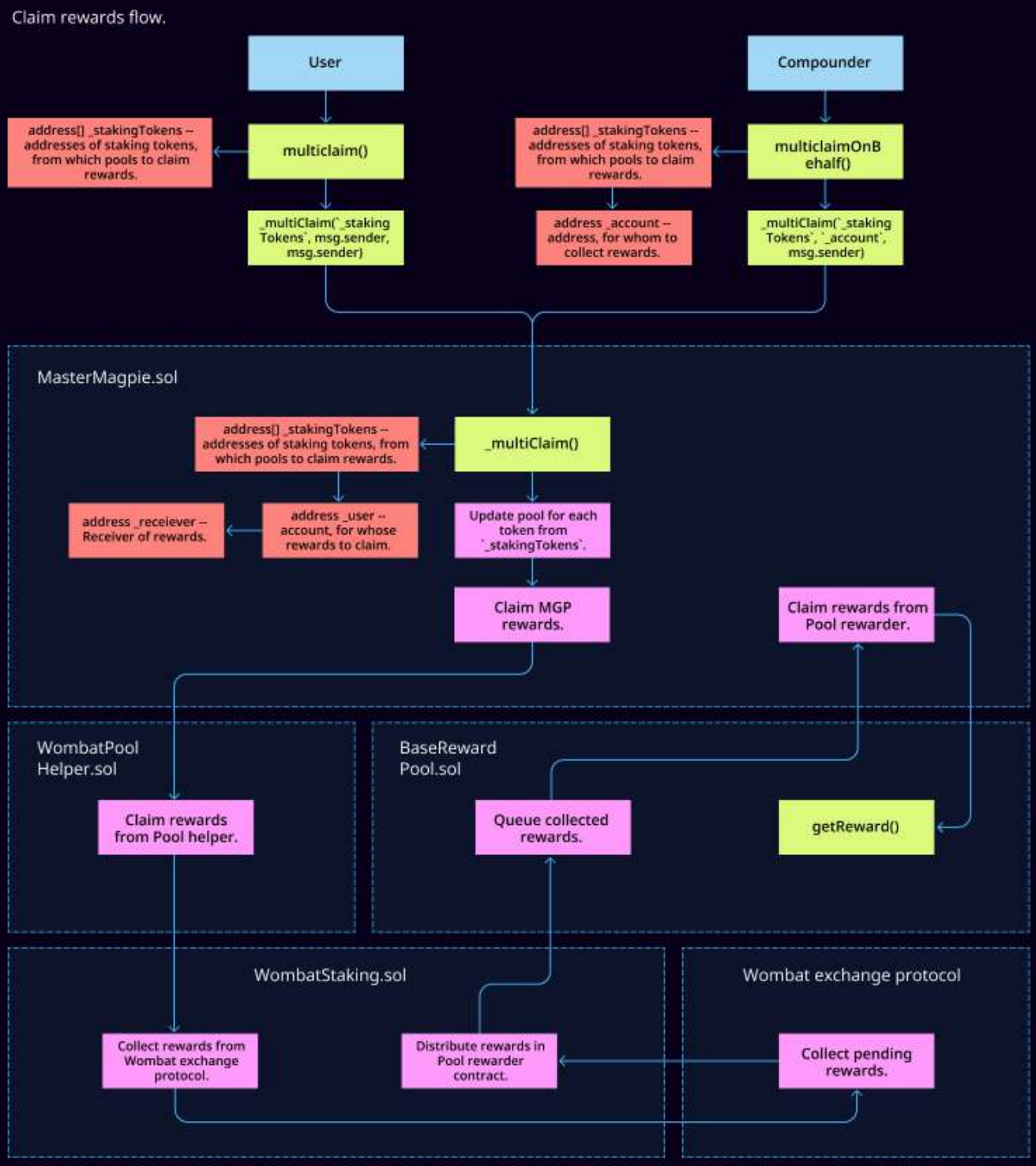
The overall security of the protocol is high enough, though it lacks documentation. There weren't any critical issues detected in the contracts, which could potentially cause loss of users' funds, though the code contains a quite complex logic, connected to the interaction with external protocols and rewards distrbution which might behave unexpectedly in production, which is why the total score of the protocol is 90.

PROTOCOL OVERVIEW

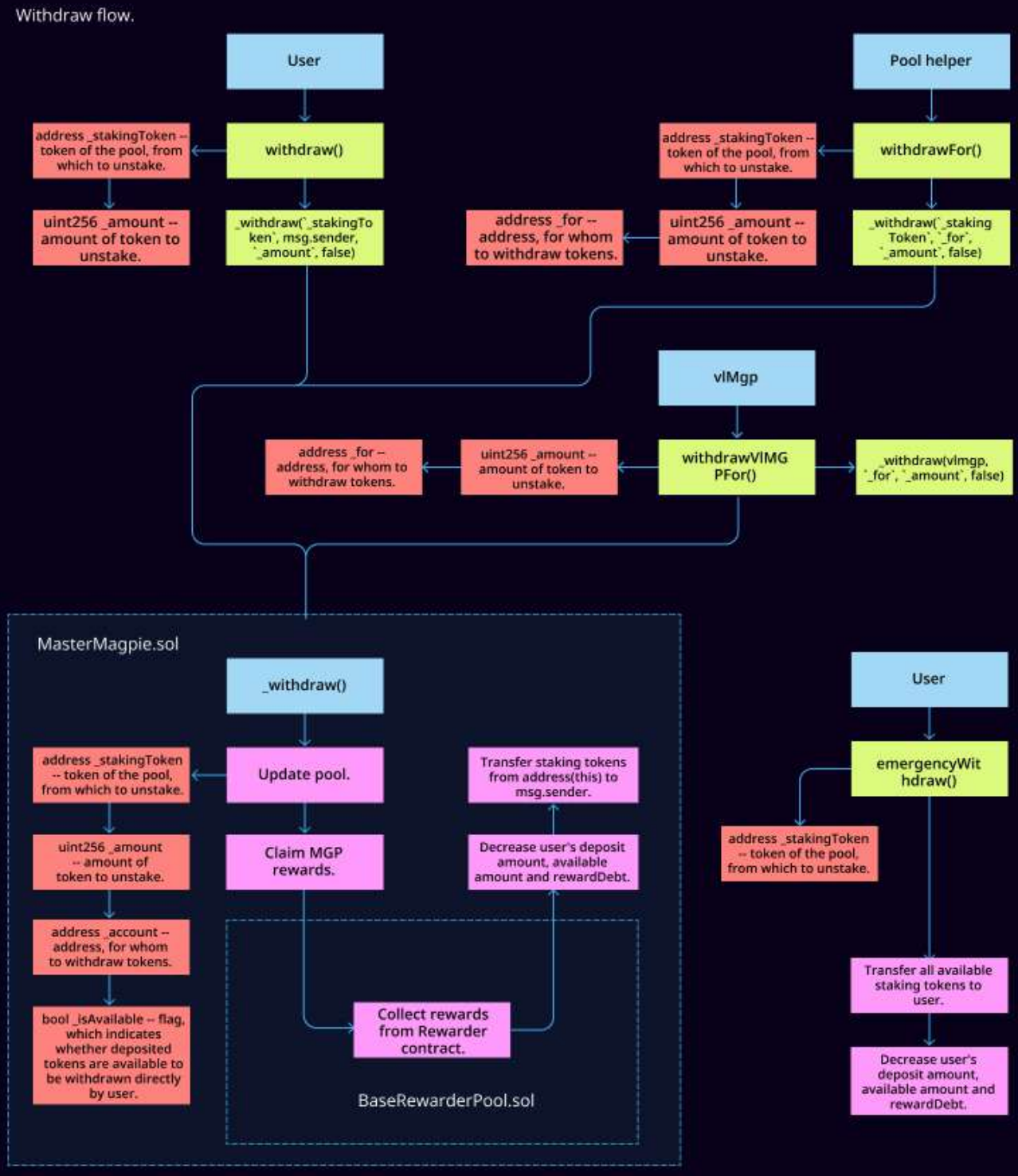
MASTERMAGPIE.SOL



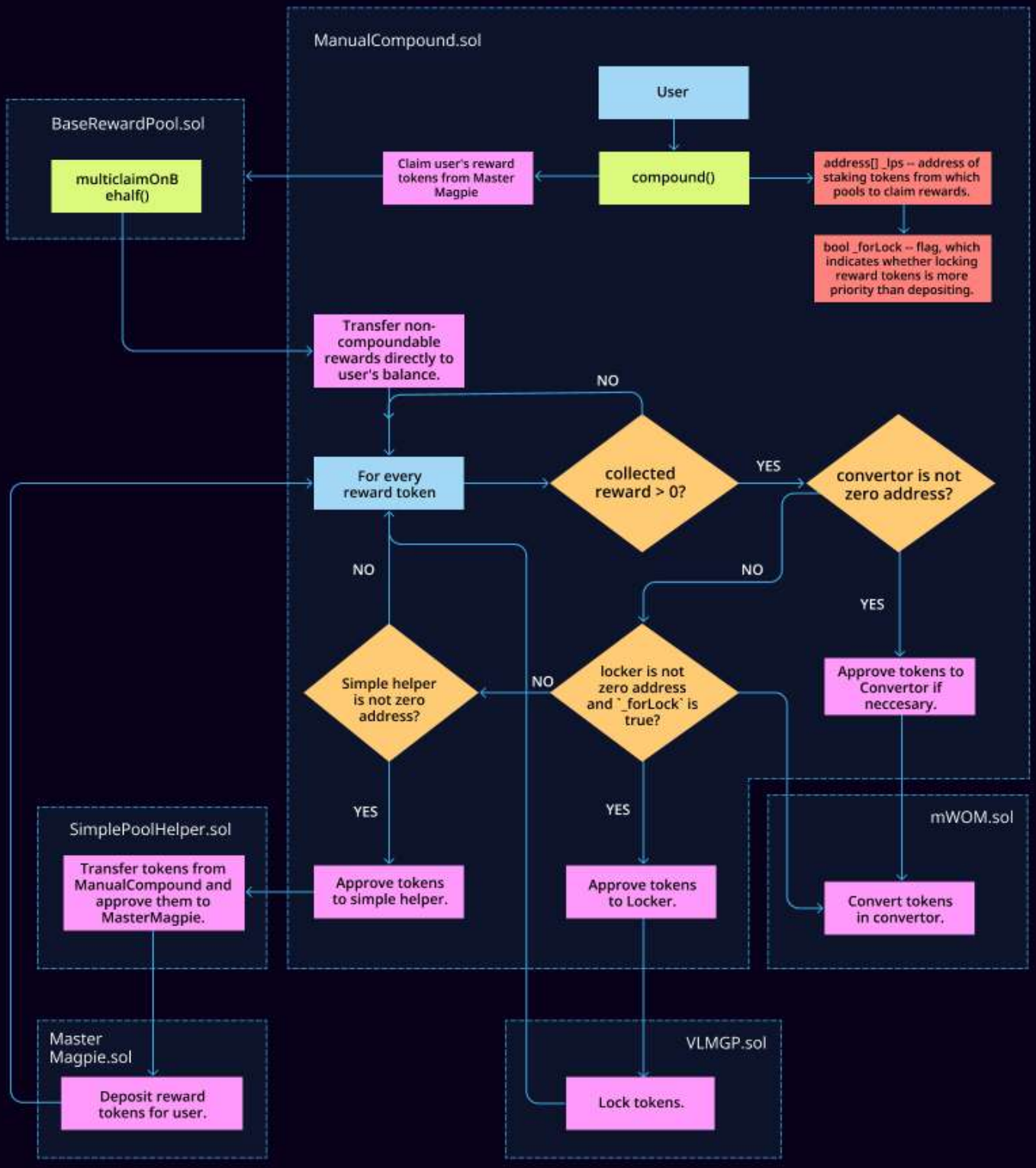
MASTERMAGPIE.SOL

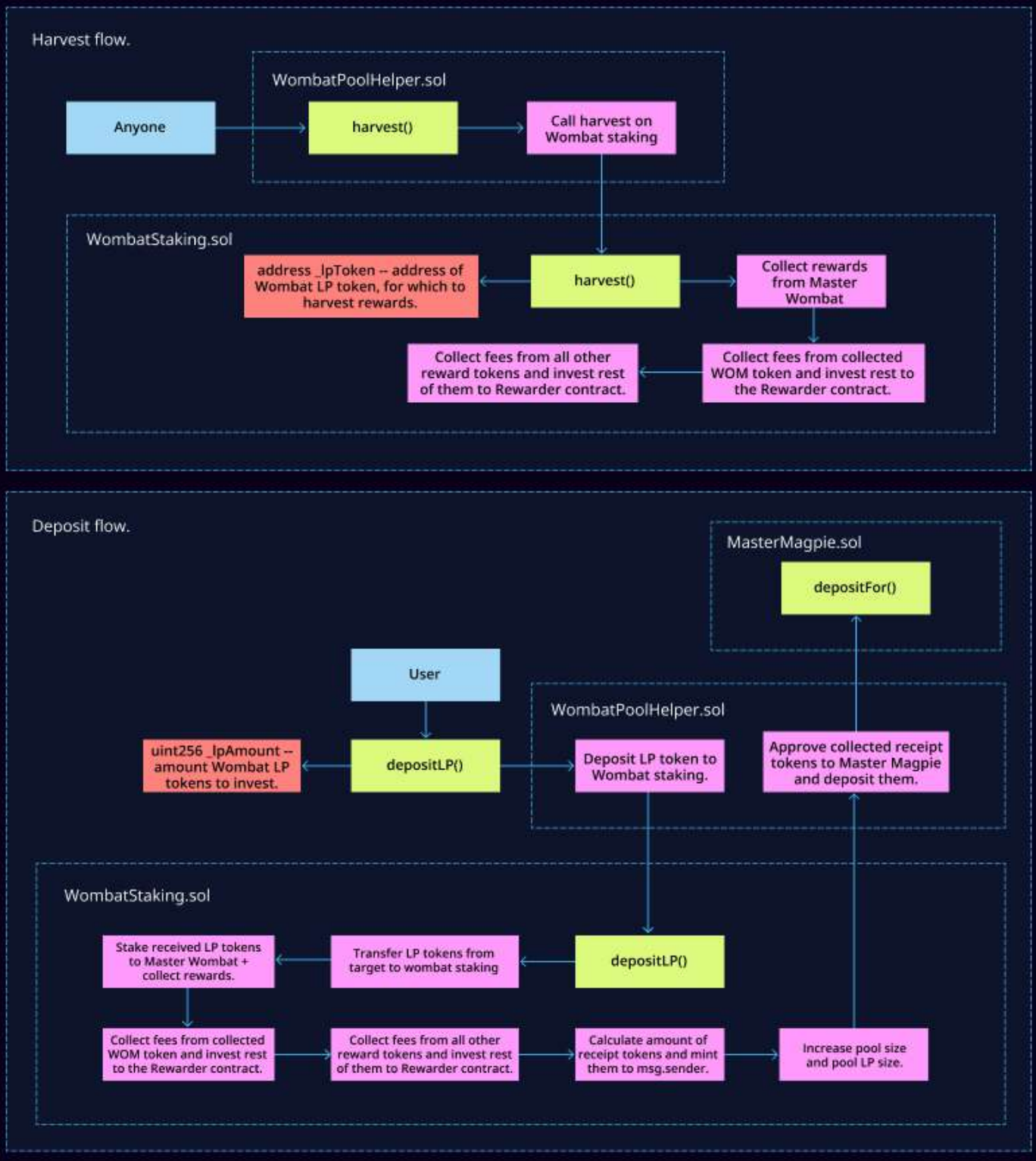


MASTERMAGPIE.SOL

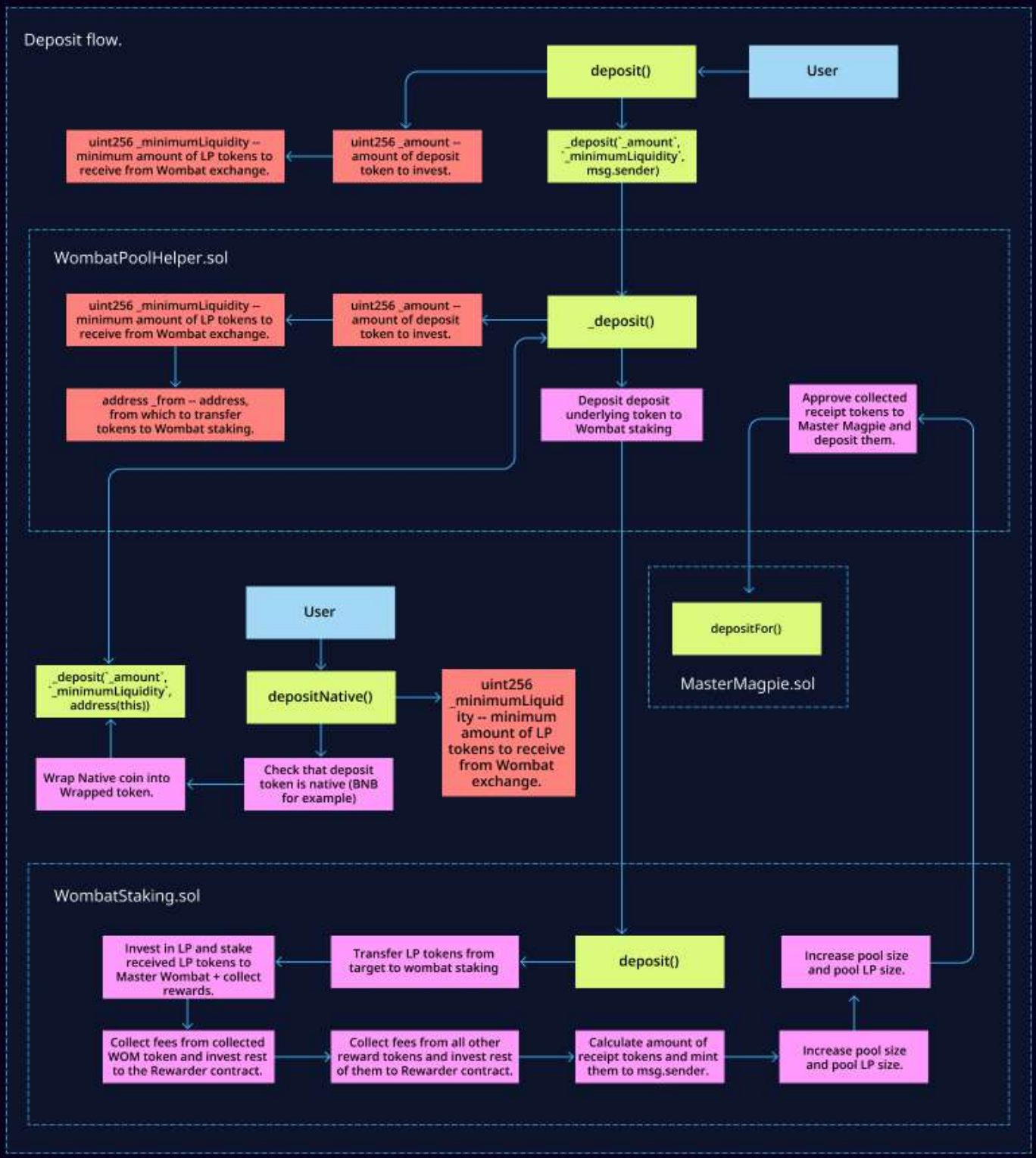


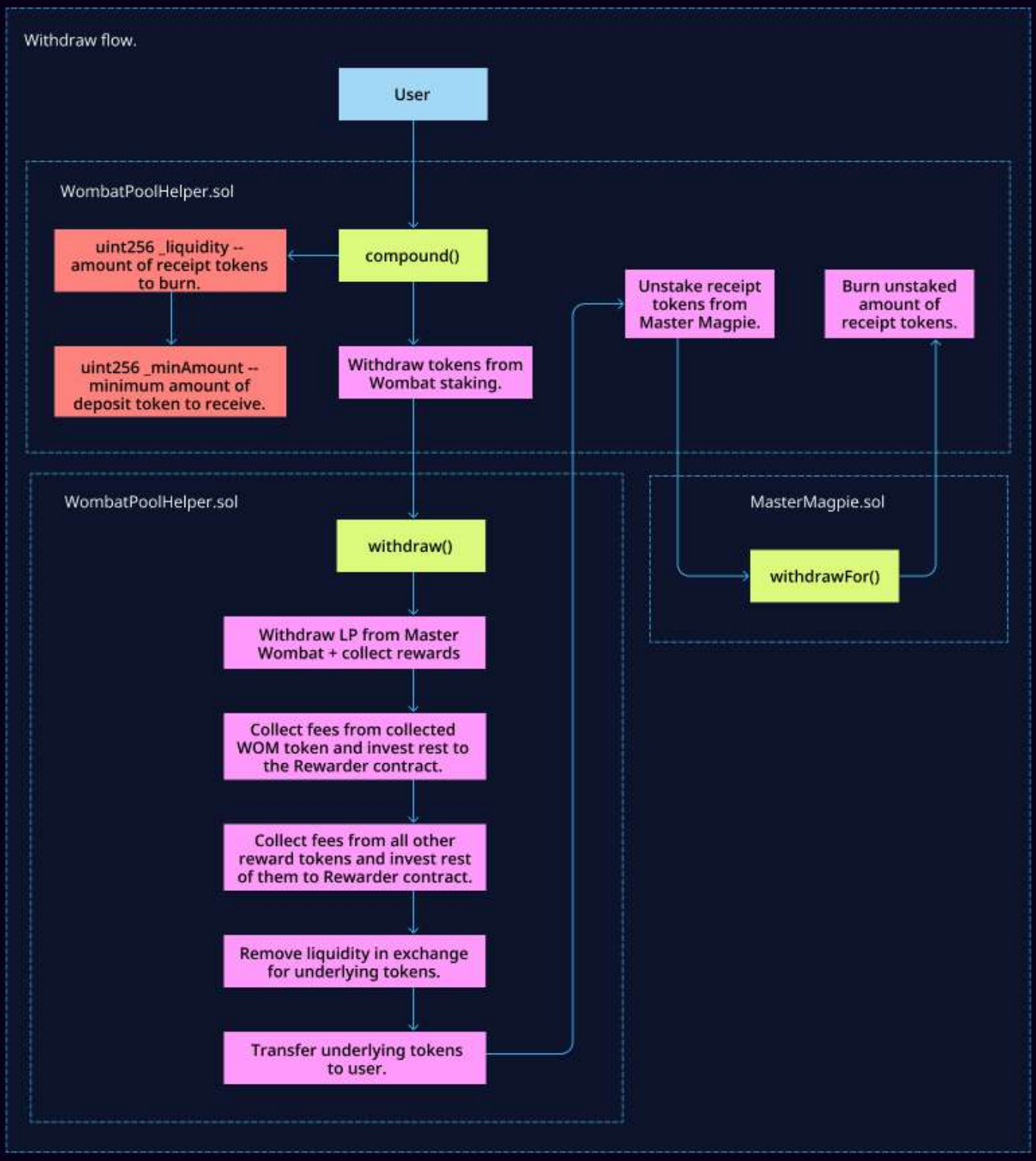
MANUALCOMPOUND.SOL

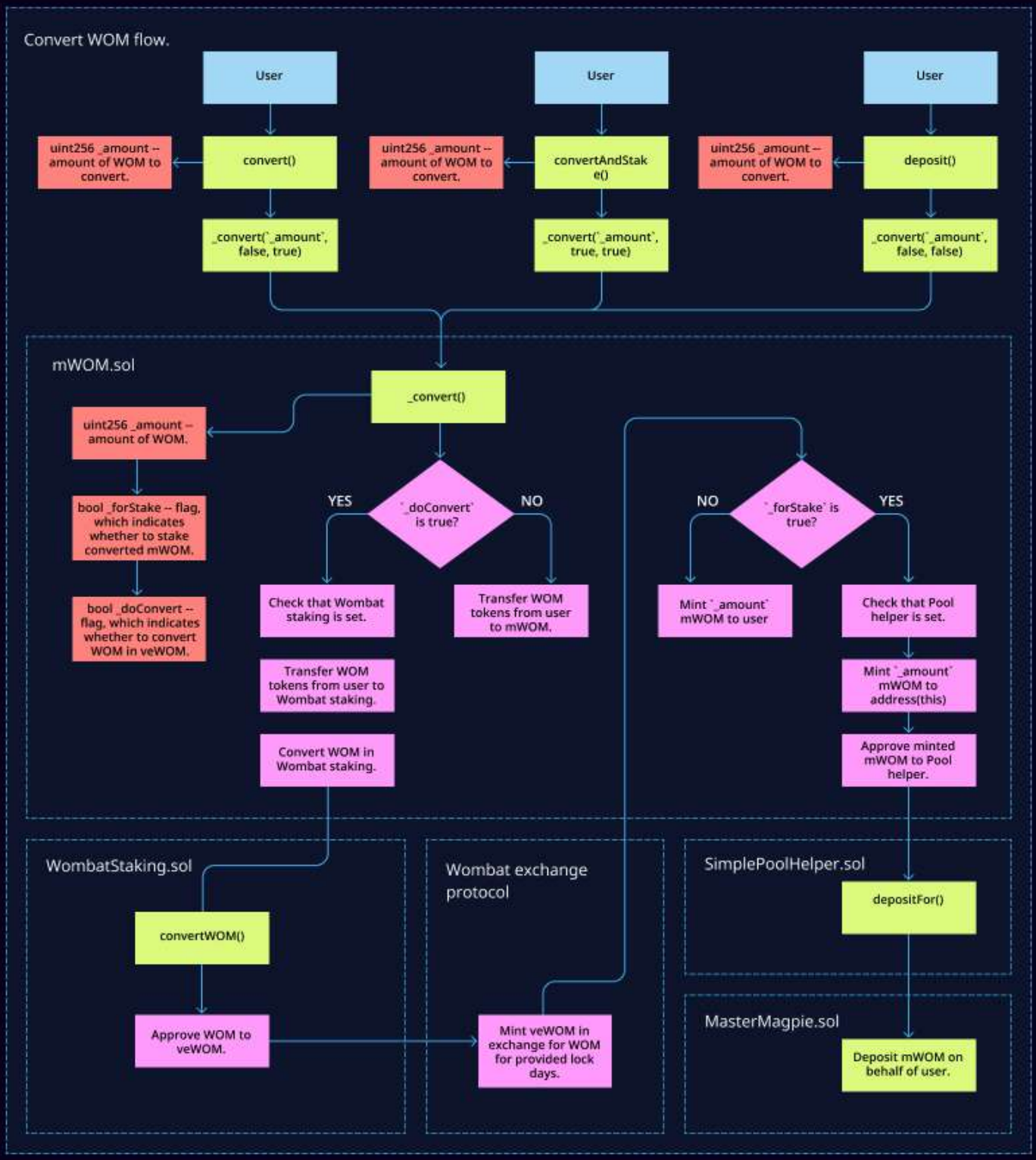



Harvest flow.
WombatPoolHelper.sol
Anyone
harvest()
Call harvest on Wombat staking
WombatStaking.sol
address _lpToken -- address of Wombat LP token, for which to harvest rewards.
harvest()
Collect rewards from Master Wombat
Collect fees from all other reward tokens and invest rest of them to Rewarder contract.
Collect fees from collected WOM token and invest rest to the Rewarder contract.
Deposit flow.
MasterMagpie.sol
depositFor()
User
WombatPoolHelper.sol
uint256 _lpAmount -- amount Wombat LP tokens to invest.
depositLP()
Deposit LP token to Wombat staking.
Approve collected receipt tokens to Master Magpie and deposit them.
WombatStaking.sol
Stake received LP tokens to Master Wombat + collect rewards.
Transfer LP tokens from target to wombat staking
depositLP()
Collect fees from collected WOM token and invest rest to the Rewarder contract.
Collect fees from all other reward tokens and invest rest of them to Rewarder contract.
Calculate amount of receipt tokens and mint them to msg.sender.
Increase pool size and pool LP size.

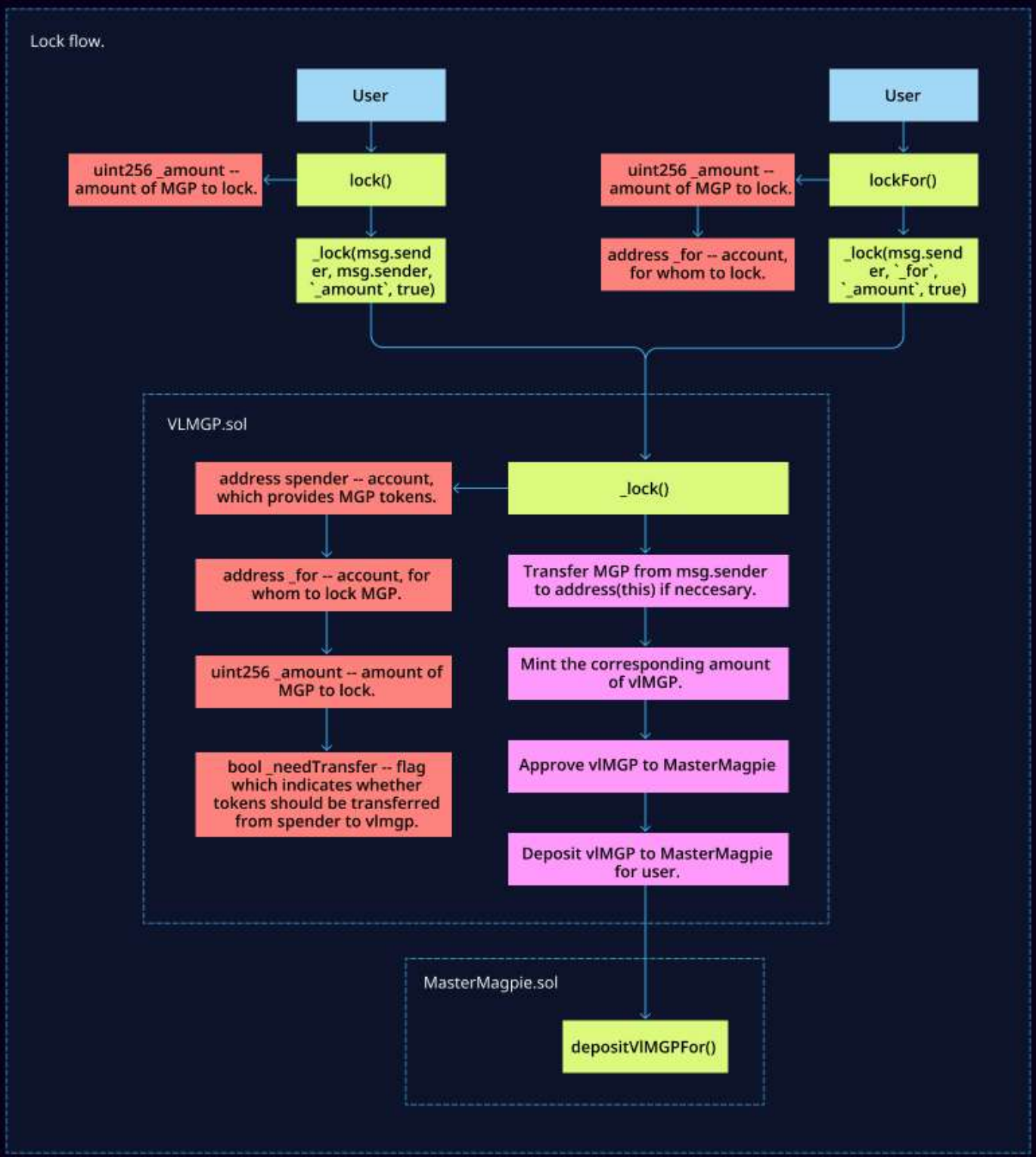
WOMBATPOOLHELPER.SOL, WOMBATSTAKING.SOL, VLMGP.SOL



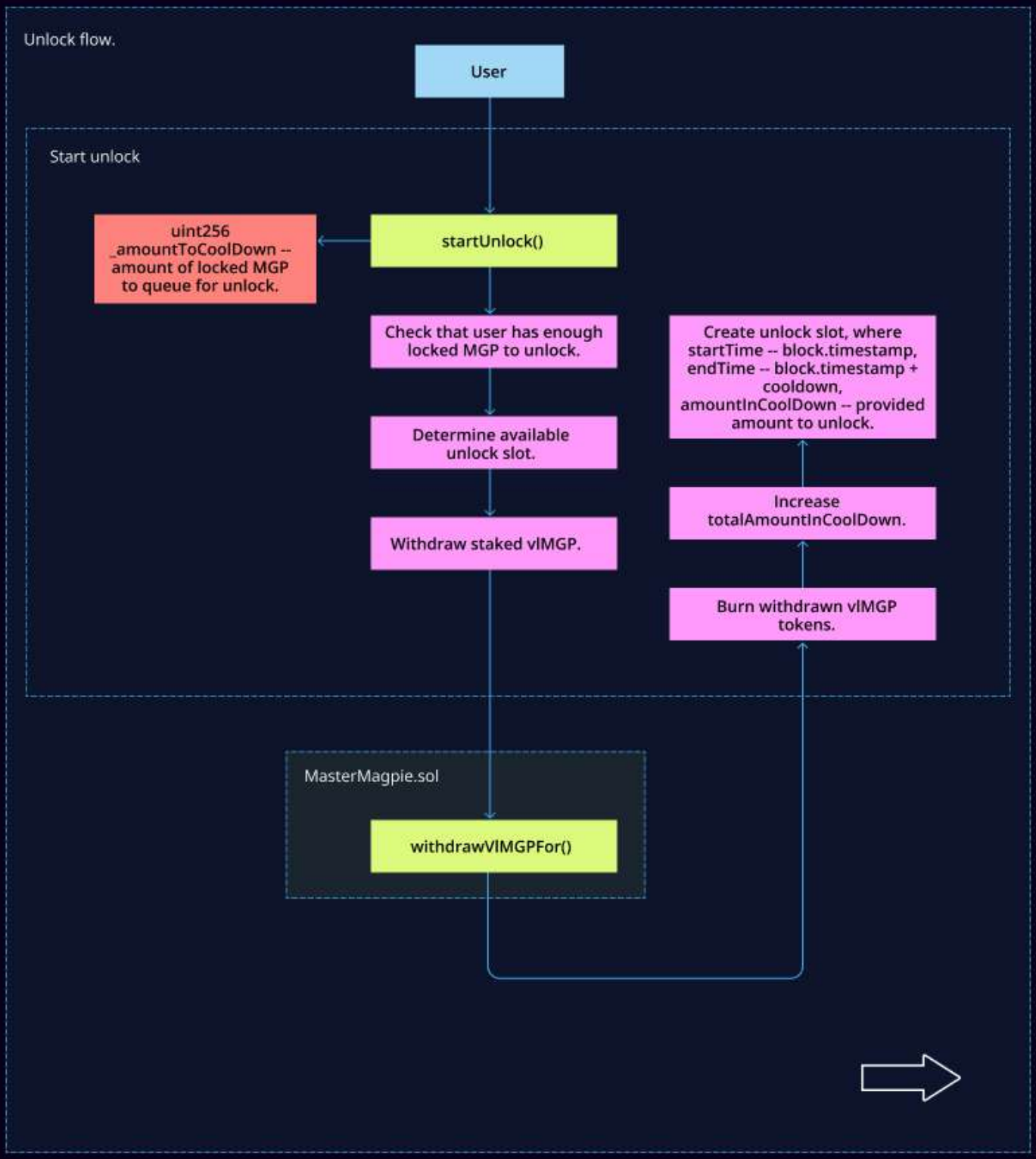

Withdraw flow.
User
WombatPoolHelper.sol
compound()
uint256 _liquidity -- amount of receipt tokens to burn.
uint256 _minAmount -- minimum amount of deposit token to receive.
Withdraw tokens from Wombat staking.
Unstake receipt tokens from Master Magpie.
Burn unstaked amount of receipt tokens.
WombatPoolHelper.sol
withdraw()
Withdraw LP from Master Wombat + collect rewards
Collect fees from collected WOM token and invest rest to the Rewarder contract.
Collect fees from all other reward tokens and invest rest of them to Rewarder contract.
Remove liquidity in exchange for underlying tokens.
Transfer underlying tokens to user.
MasterMagpie.sol
withdrawFor()


Convert WOM flow.
User
User
User
uint256 _amount -- amount of WOM to convert.
convert()
uint256 _amount -- amount of WOM to convert.
convertAndStake()
uint256 _amount -- amount of WOM to convert.
deposit()
_convert(`_amount`, false, true)
_convert(`_amount`, true, true)
_convert(`_amount`, false, false)
mWOM.sol
_convert()
uint256 _amount -- amount of WOM.
bool _forStake -- flag, which indicates whether to stake converted mWOM.
bool _doConvert -- flag, which indicates whether to convert WOM in veWOM.
YES
`_doConvert` is true?
NO
Check that Wombat staking is set.
Transfer WOM tokens from user to mWOM.
Transfer WOM tokens from user to Wombat staking.
Convert WOM in Wombat staking.
NO
`_forStake` is true?
YES
Mint `_amount` mWOM to user
Check that Pool helper is set.
Mint `_amount` mWOM to address(this)
Approve minted mWOM to Pool helper.
WombatStaking.sol
convertWOM()
Approve WOM to veWOM.
Wombat exchange protocol
Mint veWOM in exchange for WOM for provided lock days.
SimplePoolHelper.sol
depositFor()
MasterMagpie.sol
Deposit mWOM on behalf of user.

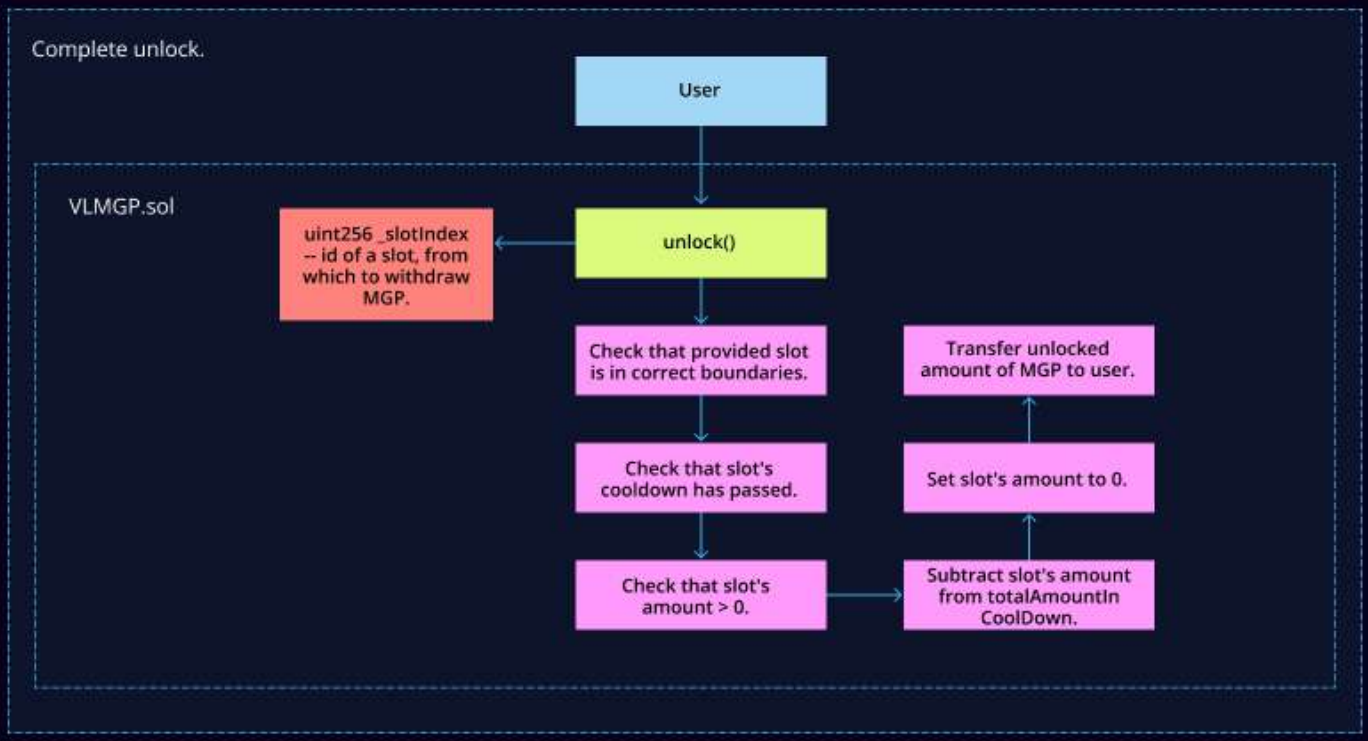
VLMGP.SOL

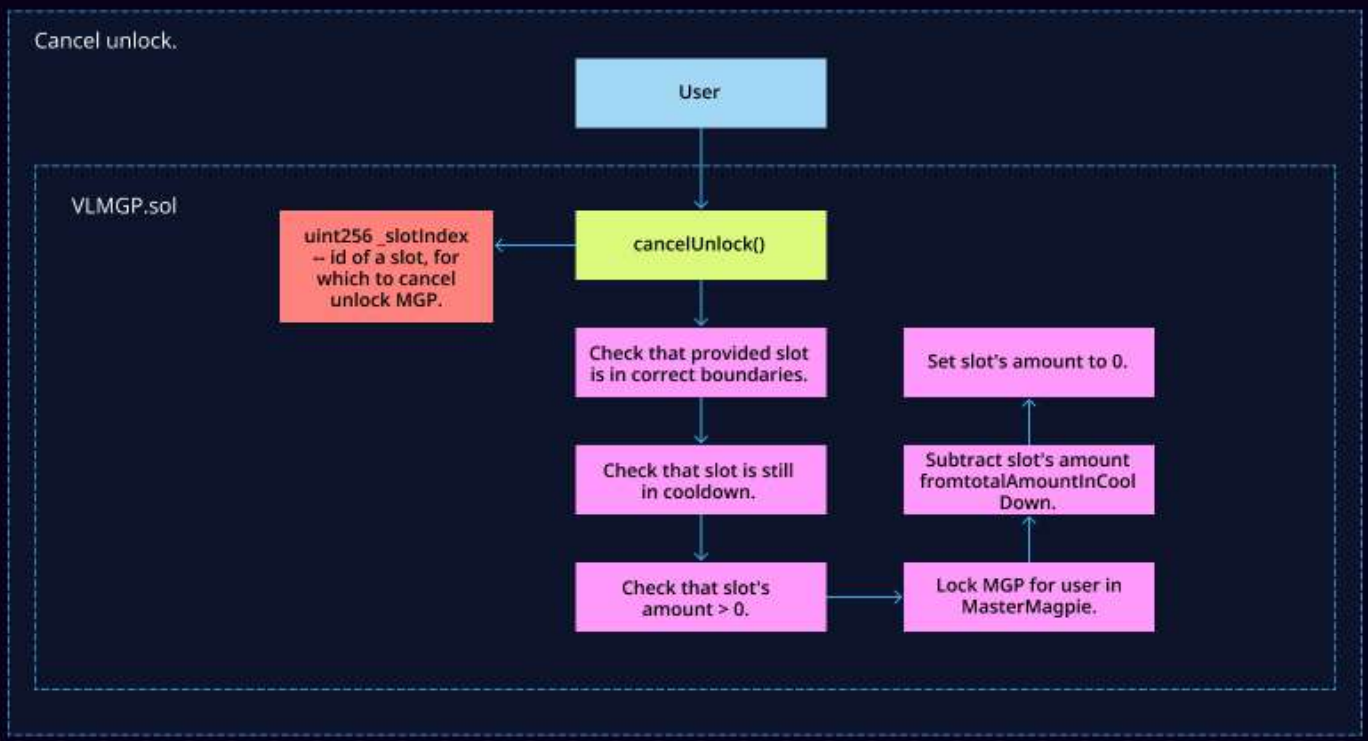

Lock flow.
User
uint256 _amount -- amount of MGP to lock.
lock()
_lock(msg.sender, msg.sender, `_amount`, true)
User
uint256 _amount -- amount of MGP to lock.
lockFor()
address _for -- account, for whom to lock.
_lock(msg.sender, `_for`, `_amount`, true)
VLMGP.sol
address spender -- account, which provides MGP tokens.
_lock()
address _for -- account, for whom to lock MGP.
Transfer MGP from msg.sender to address(this) if neccesary.
uint256 _amount -- amount of MGP to lock.
Mint the corresponding amount of vlMGP.
bool _needTransfer -- flag which indicates whether tokens should be transferred from spender to vlmgp.
Approve vlMGP to MasterMagpie
Deposit vlMGP to MasterMagpie for user.
MasterMagpie.sol
depositVlMGPFor()

VLMGP.SOL

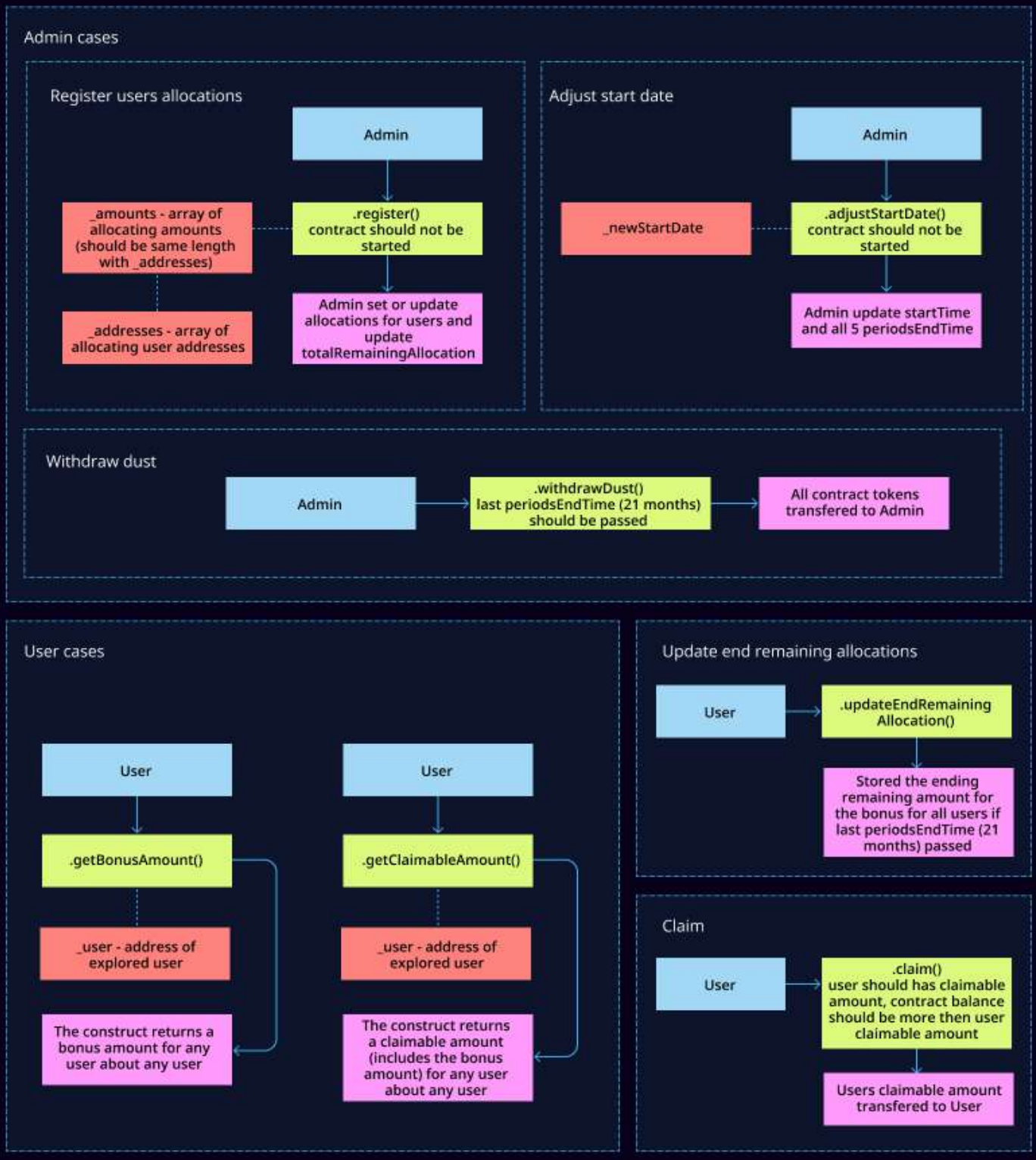

Unlock flow.
User
Start unlock
uint256 _amountToCoolDown -- amount of locked MGP to queue for unlock.
startUnlock()
Check that user has enough locked MGP to unlock.
Determine available unlock slot.
Withdraw staked vlMGP.
Create unlock slot, where startTime -- block.timestamp, endTime -- block.timestamp + cooldown, amountInCoolDown -- provided amount to unlock.
Increase totalAmountInCoolDown.
Burn withdrawn vlMGP tokens.
MasterMagpie.sol
withdrawVlMGPFor()

VLMGP.SOL


Complete unlock.
User
VLMGP.sol
uint256 _slotIndex -- id of a slot, from which to withdraw MGP.
unlock()
Check that provided slot is in correct boundaries.
Check that slot's cooldown has passed.
Check that slot's amount > 0.
Subtract slot's amount from totalAmountIn CoolDown.
Set slot's amount to 0.
Transfer unlocked amount of MGP to user.

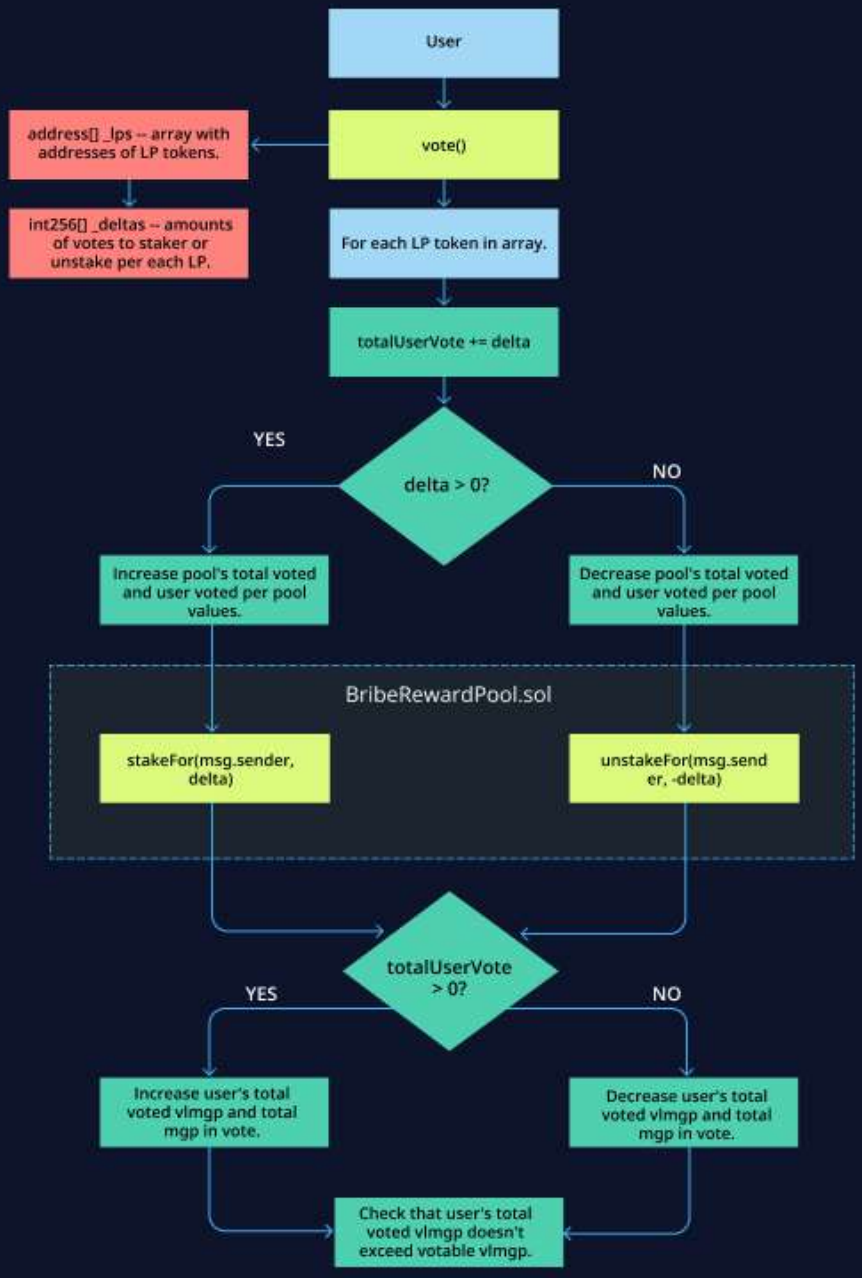

Cancel unlock.
User
VLMGP.sol
uint256 _slotIndex -- id of a slot, for which to cancel unlock MGP.
cancelUnlock()
Check that provided slot is in correct boundaries.
Check that slot is still in cooldown.
Check that slot's amount > 0.
Lock MGP for user in MasterMagpie.
Subtract slot's amount fromtotalAmountInCool Down.
Set slot's amount to 0.

AIRDROP.SOL


Admin cases
Register users allocations
Admin
_amounts - array of allocating amounts (should be same length with _addresses)
.register() contract should not be started
_addresses - array of allocating user addresses
Admin set or update allocations for users and update totalRemainingAllocation
Adjust start date
Admin
_newStartDate
.adjustStartDate() contract should not be started
Admin update startTime and all 5 periodsEndTime
Withdraw dust
Admin
.withdrawDust() last periodsEndTime (21 months) should be passed
All contract tokens transfered to Admin
User cases
User
.getBonusAmount()
_user - address of explored user
The construct returns a bonus amount for any user about any user
User
.getClaimableAmount()
_user - address of explored user
The construct returns a claimable amount (includes the bonus amount) for any user about any user
Update end remaining allocations
User
.updateEndRemaining Allocation()
Stored the ending remaining amount for the bonus for all users if last periodsEndTime (21 months) passed
Claim
User
.claim() user should has claimable amount, contract balance should be more then user claimable amount
Users claimable amount transfered to User

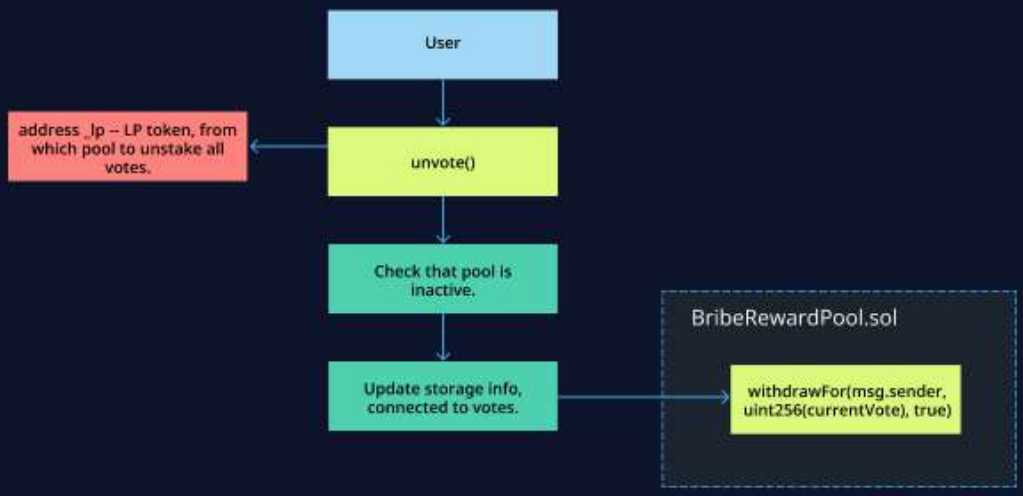
Voting flow.

Allows to stake and unstake votes, based on stakd amount of VLMGP, in active pool.



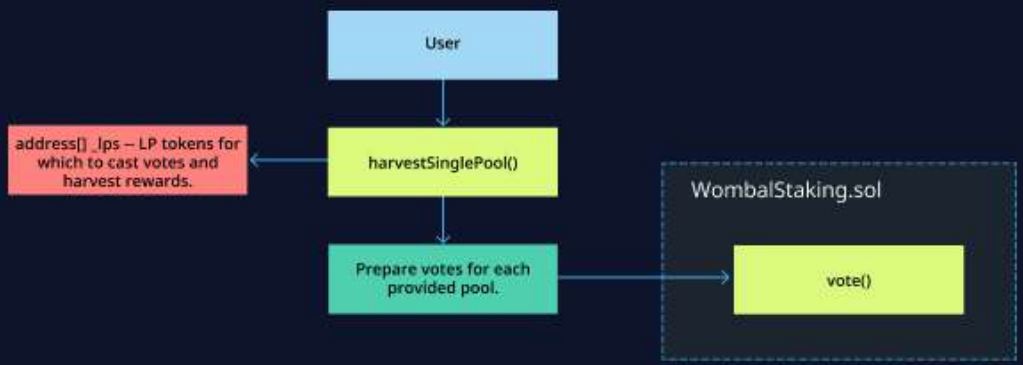
Unvoting flow.

Allows to unstake all the votes from inactive pool.

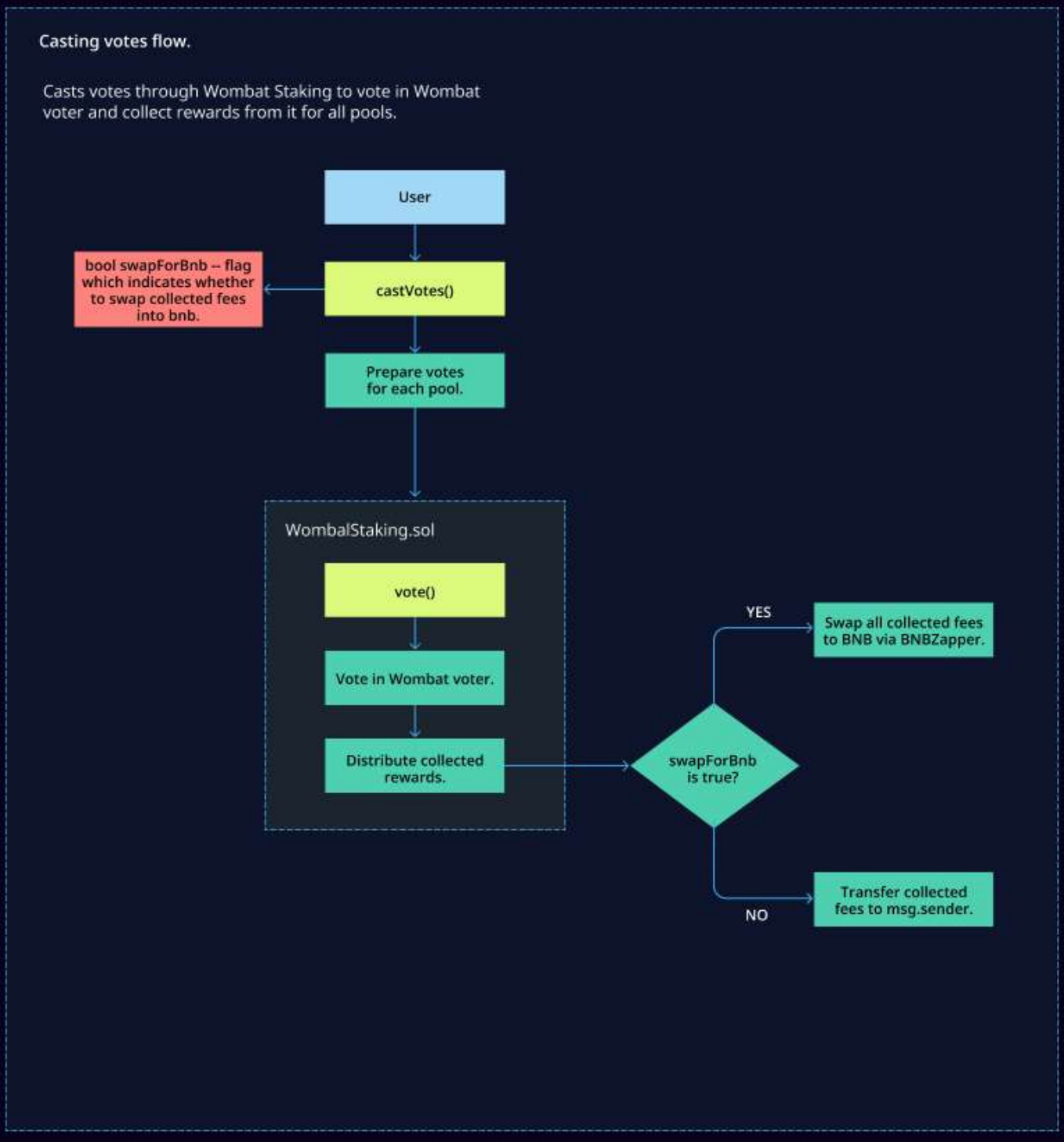


Harvesting single pool flow

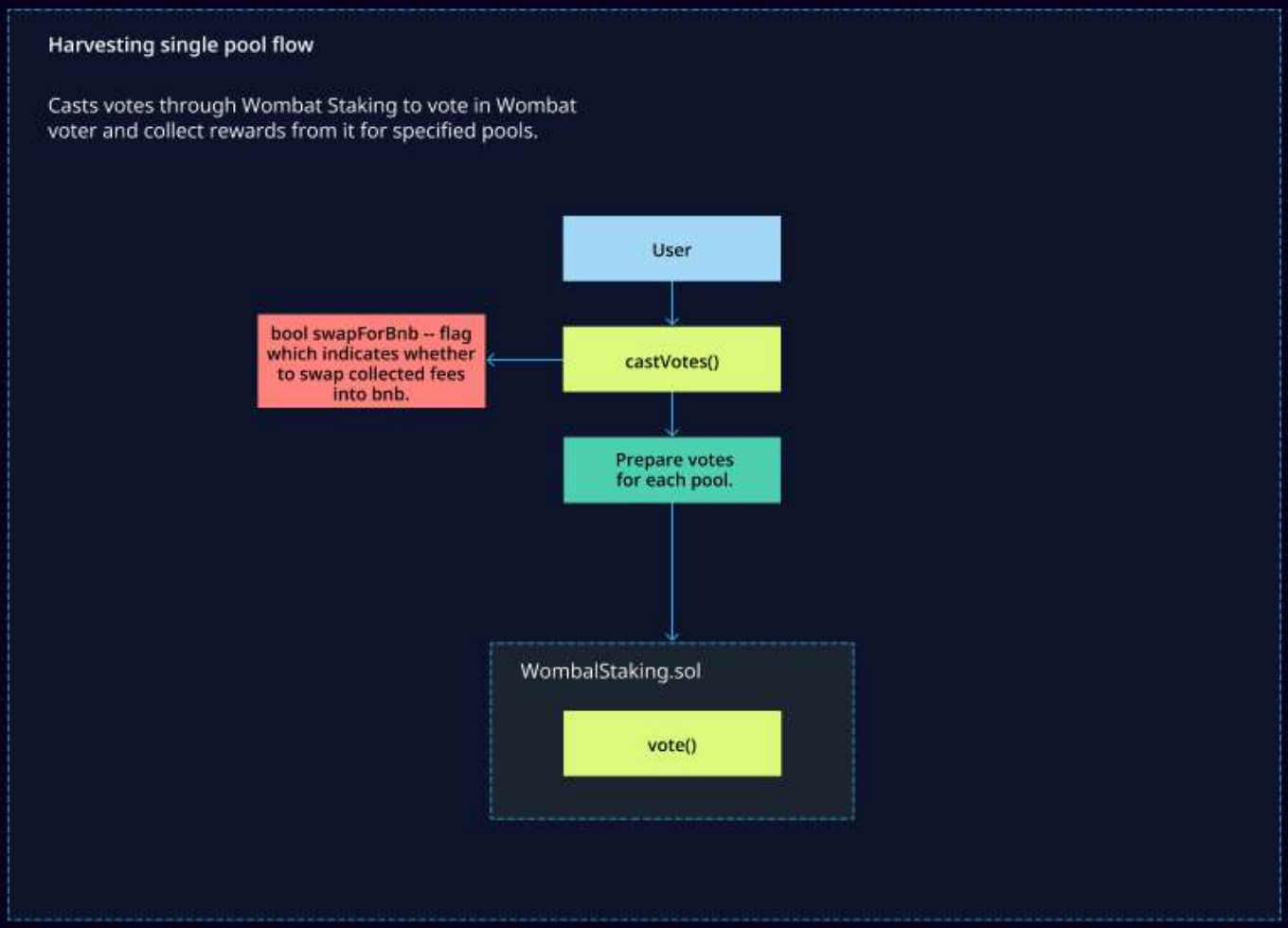
Casts votes through Wombat Staking to vote in Wombat voter and collect rewards from it for specified pools.



Unvoting flow.



Unvoting flow.



STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, 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.

1ST AUDIT ITERATION COMPLETE ANALYSIS

MEDIUM-1 | RESOLVED

Reward tokens might be stuck on the contract.

ManualCompound.sol: function compound().
During the execution of this function, all reward tokens are either locked in the reward locker or deposited in MasterMagpie via the pool helper. However, in case both reward locker and pool helper are set as zero addresses, rewards will not be deposited or sent to the user and will stay on the contract's balance.

Recommendation:
Transfer rewards to the user's balance in case both the reward locker and pool helper are set as zero address.

LOW-1 | RESOLVED

Users' allocations can be overwritten.

Airdrop.sol: function register(), line 67.
There is no validation that `_addresses[i]` account has already assigned allocations. This way, the owner can change allocations for the same users more than once. Thus, `totalRemainingAllocation` is increased every time and might eventually contain the wrong total number of allocations. The issue is marked as low since only the owner can call this function, but due to this issue, the airdrop rewards might be calculated incorrectly.

Recommendation:
Validate that users' allocations can't be changed once they are set.

LOW-2 | UNRESOLVED

Variables lack validation.

1) Airdrop.sol: function constructor(), adjustStartDate(), register.
The `_token` parameter should be validated not to be equal to zero address.
The `_startTime` and `_newStartDate` parameters should be validated to be greater than block.timestamp.
Elements from the `_addresses` array should not be zero addresses.
2) BaseRewardPool.sol: constructor().
All constructor parameters except for `_rewardToken` should be validated not to be equal to zero address.
3) MasterMagpie.sol: __MasterMagpie_init()
The `_mgp` parameter should be validated not to be equal to zero address.
The `_startTimestamp` parameter should be validated to be greater than block.timestamp.
4) MGPRelease.sol: function register()
Elements from the `_addresses` array should not be zero addresses.

It is recommended to validate some of the function parameters before setting them in storage or executing further function code, especially those which can be set only once.

Recommendation:
Validate the function parameters.

Post-audit:
Point 3 was not fixed. All other points were successfully fixed.

LOW-3 | UNRESOLVED

Unclear way of how rewards are sent to the contract's balance.

1) Airdrop.sol, MGPRelease.sol
There is no mandatory transfer of the reward when assigning allocations with the register() function. Thus, there might not be enough rewards to pay.
2) There is no mandatory transfer of rewards when setting the emission. Thus, there might not be enough tokens to distribute to users.

Recommendation:
Verify how reward tokens are sent to the contract's balance.

LOW-4 | RESOLVED

Unchecked transfers.

Airdrop.sol: function claim(), line 160.
MasterMagpie.sol: function _safeMGPTransfer(), lines 551, 553.
Though the provided transfers are unlikely to fail, it is still recommended to use SafeERC20 library in order to check the returned value of transfers.

Recommendation:
Use SafeERC20 library.

LOW-5 | RESOLVED

Fee value might exceed denominator.

WombatStaking.sol
addFee(), lines 540-558
setFee(), lines 565 - 572
The sum of fees (the `totalFees` variable) should not exceed DENOMINATOR.
In case `totalFees` is bigger than DENOMINATOR, there will be an error when calculating the commission in the _sendRewards() function.

Recommendation:
Add relevant checks.

INFO-1 | VERIFIED

User's deposited tokens can be withdrawn by other accounts.

MasterMagpie.sol
It should be noted that not only the user can withdraw their deposited tokens and claim rewards. Accounts with the helper role for a specific pool can withdraw tokens on behalf of users to the helper's balance. Accounts with the compounder role for a specific pool can claim rewards on behalf of users to compounder's balance. These addresses can be set by pool managers at any time. Though specific smart contracts are supposed to have these roles, managers can set any account as a helper or a compounder.

Post-audit.
According to the Magpie team, only special contracts like PoolHelper and Compounder can have access to users' deposits and rewards. Both contracts have been audited. Though, the ability of masterMagpie's manager to set these addresses to any accounts still remains, Magpie team verified that the manager role will be granted to a multisig wallet.

INFO-2 | RESOLVED

Unlimited allowance.

ManualCompound.sol: function _approveTokenIfNeeded(), line 61.
Increasing the allowance for spending ERC20 tokens to maximum uint256 is considered unsafe since the spender could spend any amount at any time. The issue is marked as informational since the allowance is granted to those contracts that can be set only by the owner, though it is recommended not to set the allowance to maximum uint256.

Recommendation:
Approve the amount necessary to perform the current transfer instead of making an unlimited allowance.

INFO-3 | VERIFIED

Receipt tokens can't be withdrawn from users' balances.

WombatPoolHelper.sol
The only way to withdraw receipt tokens and receive underlying tokens is to call the withdraw() function. This function unstakes receipt tokens during the execution (line 134, _unstake()) and in case there are not enough receipt tokens staked, the function will revert. This forces the user to stake their receipt tokens first, in case they have withdrawn them earlier. The issue is marked as informational and should be verified by the team.

Post-audit.
It was verified by the Magpie team that it is intended functionality - before redeeming receipt tokens, users should leave them in the MasterMagpie contract.

INFO-4 | VERIFIED

Rewards might be distributed from users' deposits in case there is a pool for MGP.

MasterMagpie.sol: function _safeMGPTransfer().
In case there is a dedicated pool created for an MGP token, all users' deposits to this pool might be distributed as MGP rewards.

Recommendation.
Verify that the pool for staking MGP won't be created or verify that any deposited MGP won't be distributed as rewards.

From the client.
According to the Magpie team, there won't be a pool with an MGP token as a staking token.

INFO-5 | RESOLVED

Unreachable code.

VLMGP.sol
1) startUnlock(), line 221-222.
This revert can't be reached since in case all the slots have value, the function will revert on line 176 (function getNextAvailableUnlockSlot()).
2) _checkIdexInBoundary(), lines 327-328.
This revert can't be reached since it will always revert in lines 324-325 if `_slotIdx` > `maxSlot`.
3) the onlyMaster() modifier is never used.

Recommendation.
Remove all the unreachable code.

INFO-6 | RESOLVED

Zero address can be marked as a reward token.

BaseRewardPool.sol
On line 65, the reward token is checked for a zero address. The reward token is added to the rewardTokens array only when it's non-zero address. At the same time, even being a zero address, the reward token will be true in the isRewardToken mapping after being assigned on line 75.

Recommendation.
Set the reward token to true in isRewardToken only when it's non-zero address.

INFO-7 | RESOLVED

Documentation mismatch.

WombatStaking.sol
convertAllWom(), lines 365-368.
The comment is about staking the entire MGP from the balance of the contract, but the function performs the conversion of wombats.

Recommendation:
Check the content of the function, write an appropriate comment to it.

INFO-8 | RESOLVED

Typos.

mWom.sol
1) wombatStakikng - wombatStaking. (lines 25, 49)

Recommendation:
Fix typos.

INFO-9 | RESOLVED

Function state mutability can be restricted to view.

MasterMagpie.sol: function _calMGPPending().
There is no storage modification in this function, so it can be restricted to view.

Recommendation:
Restrict function state mutability to view.

2ND AUDIT ITERATION COMPLETE ANALYSIS

CRITICAL-1 | VERIFIED

Depositing through Smart WOM Converter reverts.

SmartWomConvert.sol: function depositFor(), line 98; function _convertFor, line 136.
The depositFor() function of MasterMagpie.sol has the `_onlyPoolHelper` modifier that is supposed to be a separate contract WombatPoolHelper.sol. Though, SmartWomConvert calls this function directly without the usage of WombatPoolHelper.sol. Thus, the call will revert unless WombatPoolHelper has an appropriate role on Master Magpie. The issue is marked as critical in order to attract the attention of the team as it is unclear whether SmartWomConvert.sol should have such a role on MasterMagpie. Due to this, depositing to mWOM to the Master Magpie via SmartWomConvert is currently unavailable.

Recommendation:
Use the WombatPoolHelper.sol contract or verify that SmartWomConvert should have a pool helper role.

From the client.
According to the team, the absence of WombatPoolHelper is intended in order to remove an unnecessary call. For the mWOM token, SmartWomConvert.sol will be set as a helper and the corresponding changes will be added to the deployment scripts.

HIGH-1 | RESOLVED

Deprecated ETH transfer.

BNBZapper.sol: function withdraw(), line 121.
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 the .call() functionality with a mandatory result check or the built-in functionality of the Address contract from OpenZeppelin library. The issue is marked as high since in case the owner is a multisig contract, withdrawals of ETH might be blocked.

Recommendation:
Correct the ETH sending functionality.

HIGH-2 | RESOLVED

Possible frontrun attack.

BNBZapper.sol: function _swapTokenForBNB(), line 103.
The second `minAmountOut`parameter of swapExactTokensForETH() is passed as 0. This parameter is necessary to mark the minimum amount of output token to receive. Thanks to this parameter, such threats as abnormal slippage or frontrunning can be omitted. As this parameter isn't used, an attacker can temporarily drain funds from the pool, causing the slippage to be abnormal for this swap. As a result, all funds of input token will be lost and the `receiver` address won't receive an appropriate amount of output token. Thus, it is highly recommended not to ignore the `minAmountOut` parameter.

Recommendation:
Pass the `minAmountOut` parameter when performing a swap to protect the contract from a possible frontrun attack.
Post-audit.
`minRec` parameter is passed in zapInToken() function. Though it is called from the contract WombatBribeManager, function _swapFeesForBnb() where this parameter is passed as 0, it was verified by the Magpie XYZ team, that the amount of tokens to swap won't be worthy more than 5$, thus a frontrun attack on such transactions is unlikely to happen.

HIGH-3 | RESOLVED

Total MGP in vote is not decreased during unvote.

WombatBribeManager.sol: function unvote().
The `totalMgpInVote` storage variable is changed every time in the vote() function and represents the total amount of MGP voted with. This variable is also used in the castVotes() function to calculate target votes. However, this variable isn't updated in the unvote() function, which can lead to inaccurate calculations in the contract.

Recommendation:
Update the `totalMgpInVote` variable in the unvote function.

MEDIUM-1 | RESOLVED

Unchecked transfer.

BNBZapper.sol: function withdraw(), line 125; zapInToken(), line 69.
In order to prevent any unexpected errors that can occur during a transfer, it is recommended to validate the return value of the transfer(), transferFrom() functions. In this case, it is better to use SafeERC20 library by OpenZeppelin. SafeERC20 has a built-in mandatory checks on any transfer, including non-standard implementations of ERC20 tokens (e.g. USDT token).

Recommendation:
Use SafeERC20 library or ensure that a return value of the transfers is checked.

LOW-1 | RESOLVED

Variables lack validation.

Some of the variables should be validated, especially the ones that are set in the storage only once.

- BribeRewardPool.sol: constructor().

The `_stakingToken`, `_operator`, `_rewardManager` parameters should be validated not to be equal to zero address.

Recommendation:
Validate variables.

LOW-2 | RESOLVED

Fee percentages are not validated.

WombatStaking.sol: function setBribe().
The summation of `_bribeCallerFee` and `_bribeProtocolFee` is not validated in order not to exceed DENOMINATOR. In case their summation is greater than DENOMINATOR, the vote() function will revert as there are fee calculations in this function.

Recommendation:
Validate that the summation of `_bribeCallerFee` and `_bribeProtocolFee` doesn't exceed DENOMINATOR.

Post-audit.
Each variable is checked separately, though their summation still can exceed DENOMINATOR.

INFO-1 | RESOLVED

Unlimited allowance.

• BNBZapper.sol: function _approveTokenIfNeeded().
Making an unlimited allowance of token might be potentially dangerous and lead to funds loss in case of exploitation. It is recommended to approve tokens before each transfer on the amount necessary for a particular transfer. The issue is marked as informational since the allowance is granted to the Pancake router, which is not upgradable.

Recommendation:
Approve the tokens necessary for the transfer before each transfer operation instead of granting unlimited allowance.

INFO-2 | VERIFIED

More complicated routes can't be set.

BNBZapper.sol: function _findRouteToBnb().
The contract allows only one intermediate token between `token` and BNB. In most of the cases, one intermediate token should be enough. However, there are some cases where more complicated routes should be set, where more than an intermediate token is necessary to perform a swap.

Recommendation:
Consider allowing for setting more complicated router for a swap with more than one intermediate token.

From the client.
According to the team, one intermediate token will be enough for all reward tokens so far. The corresponding changes might be applied later, when more complicated routes will be necessary.

INFO-3 | RESOLVED

Public functions can be marked as external.

Some of the public functions, which are never called within other functions in the contract, can be marked as external.

- BNBZapper.sol: routePair(), previewTotalAmount().
- BribeRewardPool.sol: rewardDecimals(), getRewardUser().
- WombatStaking.sol: getPoolTokenList(), expectedVeWomAmount().
- VLMGP.sol: function startUnlock().

Recommendation:
Mark functions as external.

INFO-4 | RESOLVED

Unnecessary validation.

- BNBZapper.sol: constructor(), line 28.

A validating that owner() is not zero address is redundant since in any Ownable contract the owner is assigned with msg.sender, which can't be zero address.

Recommendation:
Remove unnecessary validation.

INFO-5 | RESOLVED

Unnecessary function.

- BNBZapper.sol: function routePair().
- BribeRewardPool.sol: function getStakingToken().

Declaring a getter for a public storage variable is redundant since public variables have their own getter.

Recommendation:
Remove unnecessary functions.

INFO-6 | RESOLVED

Code duplication.

- BribeRewardPool.sol: functions updateFor(), _updateFor().

The code is duplicated in these two functions. It is recommended to avoid any duplications and call an internal function rather than an external one.

- BribeRewardPool.sol: getReward(), getRewardUser().

The core logic of these two functions is identical except for the user who these rewards are collected for. It is recommended to use an additional internal function with the core logic instead of duplicating the code.

Recommendation:
Remove the duplicated code.

INFO-7 | RESOLVED

Unnecessary gas consumption.

- BNBZapper.sol

In case the getReward() function is called from the withdraw() function, the updateReward() and onlyOperator modifiers are called twice. In order to avoid redundant gas spending, it is recommended to create the internal _getReward() function with the core logic and call it within the withdraw() and getReward() functions.

Recommendation:
Remove unnecessary gas consumption.

INFO-8 | RESOLVED

Same bonus reward tokens might be added.

WombatStaking.sol: function addBonusRewardForAsset().
There is no validation in this function that `_bonusToken` isn't already added for the provided `_lpToken`. Hence, the same bonus reward token can be added more than once.

Recommendation:
Validate that `_bonusToken` isn't already added for the provided `_lpToken`.

INFO-9 | RESOLVED

Unreachable case.

BNBZapper line 114-116 will always send 0 ETH since the contract has no receive, fallback, and payable functions, so no ETH can be "stuck".

Recommendation:
Remove unnecesary branch connected to the ETH transfer.

Optimization V2 PR review.

- MasterMagpie.sol line 194 variable used only once, so it can be removed and its value used directly
- MasterMagpie.sol line 378 variable used only once, so it can be removed and its value used directly
- MasterMagpie.sol line 492 variable used only once, so it can be removed and its value used directly
- MasterMagpie.sol line 502 variable used only once, so it can be removed and its value used directly
- MasterMagpie.sol line 529 variable used only once, so it can be removed and its value used directly
- MasterMagpie.sol line 603 variable used only once, so it can be removed and its value used directly
- MasterMagpie.sol line 437 availableAount ⇒ availableAmount
- BaseRewardPool.sol balanceOf function access modifier should be changed from external to public to use it in the same contract without calling this.balanceOf
- Same as above for totalStaked function
- ManualCompound.sol repetitive check
  if(_index >= rewards.length) revert InvalidIndex();
  it should be replaced with a modifier.
- MasterMagpie.sol line 523-525 uint256 have 0 value by default

INFO-11 | RESOLVED

No event in setter.

WombatStaking.sol: setSmartConvert().
In order to preserve the history of setting variables, it is better to emit events in setters.

Recommendation:
Add an appropriate event.

INFO-12 | RESOLVED

Unused code.

mWom.sol: error OnlyWomUp().
This error is not used anywhere in the code and takes up space in the contract, increasing the size of the contract.

Recommendation:
Remove unused code.

	contracts\rewards	contracts\wombat
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

	contract\Mgp.sol	contracts\ProxyAdmin.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

contracts\TransparentUpgradeableProxy.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

	contract\VLMGP.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

	VLMGP.sol	BNBZapper.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

	BribeRewardPool.sol	WombatBribeManager.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

	WombatStaking.sol	MasterMagpie.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

	SmartWomConvert.sol	ManuallCompound.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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES FOR THE 1ST AUDIT REVISION

Tests written by the Magpie team

As a part of our work assisting Magpie in verifying the correctness of their contract code, our team has checked the complete set of unit tests prepared by the Magpie team.

We need to mention that the original code has a significant original coverage with testing scenarios provided by the Magpie team. All of them were also carefully checked by the team of auditors.

Airdrop

Initialization

✓ airdrop should initialzied correctly (320ms)

Register

✓ should revert if account length and reward length is not same (99ms)

1) should revert if airdrop has been started

✓ should all reward be registered correctly (217ms)

Claim and withdraw

✓ should revert with 'InsufficientBalance()' when players claim if there is no MGP balance in the contract (57ms)

At TGE

✓ should claimable amount of all players will be 10% of total reward of each other (44ms)
✓ should bonus of all player is 0 before final period
✓ should totalEndRemainingAllocation always is 0 before final period
✓ should totalBonus is 0 before any player claim
✓ should players get 10% of reward when claim in the 1st period (1420ms)
✓ should totalBonus be increase if any player claim before final period (1498ms)
✓ should revert with 'AirdropNotEnded()' before end time (57ms)

At 3 month time

✓ should claimable amount of all players will be 20% of total reward of each other (174ms)
✓ should bonus of all player is 0 before final period (81ms)
✓ should totalEndRemainingAllocation always is 0 before final period
✓ should totalBonus is 0 before any player claim (43ms)
✓ should players get 20% of reward when claim in the 2nd period (2657ms)
✓ should totalBonus be increase if any player claim before final period (1806ms)
✓ should revert with 'AirdropNotEnded()' before end time (128ms)

At 6 month time

✓ should claimable amount of all players will be 40% of total reward of each other (191ms)
✓ should bonus of all player is 0 before final period (88ms)
✓ should totalEndRemainingAllocation always is 0 before final period

✓ should totalBonus is 0 before any player claim (38ms)
✓ should players get 40% of reward when claim in the 3rd period (1879ms)
✓ should totalBonus be increase if any player claim before final period (3959ms)
✓ should revert with 'AirdropNotEnded()' before end time

At 9 month time

✓ should claimable amount of all players will be 70% of total reward of each other (56ms)
✓ should bonus of all player is 0 before final period
✓ should totalEndRemainingAllocation always is 0 before final period
✓ should totalBonus is 0 before any player claim
✓ should players get 70% of reward when claim in the 4th period (379ms)
✓ should totalBonus be increase if any player claim before final period (537ms)
✓ should revert with 'AirdropNotEnded()' before end time

At 12 month time

✓ should claimable amount of all players will be 100% of total reward of each other (52ms)
✓ should bonus of all player is 0 if on one claim before final period (76ms)
✓ should totalEndRemainingAllocation is total amount of all players in the final period (68ms)
✓ should players get 100% of reward when claim in the final period (2893ms)
✓ should totalBonus won't increase if players only claim in the final period (1051ms)
✓ should revert with 'AirdropNotEnded()' when withdrawDust before the end time (41ms)

withdrawDust after 21 months from the start time

✓ if no one claim, should get all of reward (400ms)
✓ if player1 claim in the final period, should get all of reward exclude player1's (2952ms)

each player claim in the different period case

✓ should get the expected result (6435ms)

Admin functions

✓ register should only by admin (58ms)
✓ adjustStartDate should only by admin (40ms)
✓ adjustStartDate should only called before airdrop started (57ms)
✓ withdrawDust should only by admin (65ms)

Base Rewarder Pool

Initialization

✓ Base rewarder should created correctly (6271ms)
✓ queueNewRewards should only be called by manager (106ms)
✓ getReward should only be called by master magpie (43ms)

Test Stake/Withdraw for functionatily

✓ total supply: total amount of staked token (33517ms)
✓ balanceOf: balance of staked token by a user (30853ms)

2) withdraw for should udpate total suuply

3) withdraw for should udpate user balances

Test reward distribution

4) "before each" hook for "Trying"

Manual Compound

Initialization

✓ ManualCompound contract should initialzied correctly

Rewards

✓ reward should be added correctly (47ms)
✓ mWom should stake back to masterMgpie after compound execution (11331ms)
✓ compound: 2 players case (28849ms)
✓ compound with locking MGP (8098ms)
✓ Compound with reward should go to user (9253ms)
✓ should emit event 'Compounded' after compound (10707ms)

Admin functions

setHelper

✓ should revert with 'Ownable: caller is not the owner'
✓ should revert with 'InvalidIndex()'
✓ helper should be set (85ms)

setConvertor

✓ should revert with 'Ownable: caller is not the owner'
✓ should revert with 'InvalidIndex()'
✓ helper should be set (85ms)

setLocker

✓ should revert with 'Ownable: caller is not the owner'
✓ should revert with 'InvalidIndex()'
✓ locker should be set (81ms)

addReward

✓ should revert with 'Ownable: caller is not the owner'
✓ new reward should be added (38ms)
✓ reward length should be 2 (39ms)

removeReward

✓ should revert with 'Ownable: caller is not the owner'
✓ should revert with 'InvalidReward()' (460ms)
✓ the 2nd reward should be removed, and the order should be kept (927ms)

Master Magpie

Initialization

✓ transparent proxy should initialzied correctly

pool manager

✓ Master Chef Onwer is a valid pool manager
✓ Maing Staking should be one of pool manager during fixture construction
✓ Create rewarder should only allowed by pool manager
✓ add pool should only allowed by pool manager
✓ set pool should only allowed by pool manager

add pool

✓ pool added should have correct information (463ms)

✓ Added existing pool should fail (484ms)
✓ pool length should return correctly for multiple pool (2922ms)
✓ Get poolInfo should return correclty (473ms)
✓ should emit event 'Add' (443ms)
✓ should emit event 'PoolManagerStatus' (110ms)

deposit

✓ the event 'Deposit' should be emitted after deposit (49ms)
✓ the event 'UpdatePool' should be emitted after deposit (6929ms)
✓ user deposited balance must increase with amount (3154ms)
✓ the size of pool must increase with amount (2993ms)
✓ deposit twice and the user deposited balance and the size of pool must increase double amount (6916ms)
✓ user won't get any mpg after first deposit, but he'll get after second deposit (7723ms)
✓ 2 players deposit test, the deposited balance of each others must be expected amount (6373ms)
✓ 2 players deposit test, the total balance of each others must be expected amount (6932ms)

withdraw

✓ the event 'Withdraw' should be emitted after withdraw (6102ms)
✓ the event 'UpdatePool' should be emitted after withdraw (5982ms)
✓ deposit and withdraw half of amount, user deposited balance must decrease with deposited amount of 1/2 (6686ms)
✓ deposit and withdraw half of amount, the size of pool must decrease with deposited amount of 1/2 (6085ms)
✓ deposit and withdraw all, the pool size, user's deposited balance shold be 0 (6067ms)
✓ deposit and withdraw twice with half amount, the pool size, user's deposited balance shold be 0 (9358ms)
✓ user won't get any mpg after first deposit, but he'll get it after withdraw (6112ms)
✓ 2 players deposit, and 1 player withdraw half of amount he deposited, the total balance of each others must be expected amount (9589ms)

MGP Emission

✓ Emission Rate should be correct
✓ Emission Rate should be updated (182ms)
✓ alloc point and total point should be updated (974ms)
✓ set pool alloc should also update alloc point and total alloc point and emit the event 'Set' (1578ms)
✓ pending MGP should be correct based timestamp increaed (892ms)
✓ pending MGP should be correct based timestamp increaed in 2 players case (1346ms)

reward

✓ should nothing happens after massUpdatePools if total supply of pools is 0, but lastRewardTimestamp will be updated (1656ms)
✓ should emit the event 'UpdatePool' with expect results if there are some deposited tokens after massUpdatePools or updatePool or deposit (3737ms)

✓ should emit the event 'UpdatePool' for each pool with expect results after claim the rewards and player should get it (2746ms)

When pause

✓ Emergency withdraw not allowed as default
✓ Emergency withdraw allowed when paused (333ms)

multi claim

✓ Deposit then multiclaim (4862ms)
✓ Multiclaim for VLMGP in masterMagpie (2427ms)
✓ should compounder updated (127ms)
✓ multiclaimOnBehalf only can be called by compounder (117ms)
✓ Deposit then multiclaimOnBehalf (5006ms)

set MPG

✓ MGP to be set must be Contract
✓ MGP should be set successfully (180ms)
✓ MGP can not be updated once set (241ms)

Admin functions

✓ setPoolManagerStatus should only by admin
✓ pause should only by admin
✓ updateEmissionRate should only by admin
✓ set should only by admin
✓ set MGP only by admin
✓ setCompounder should only by admin
✓ createRewarder should only by pool manager
✓ add should only by pool manager

mgpRelease

✓ MGP release should be initialzied correctly
✓ 10% should be claimable before vesting start (221ms)
✓ 10% should be claimable when vesting start (252ms)
✓ Dont claim init unlock but wait till 70% vested (258ms)
✓ Dont claim init untill fully vested (232ms)
✓ Claim should transfer token and claimed should be update (249ms)
✓ Claim 10% each time with initially claimed vefore vesting start (2644ms)
✓ multiple users (6606ms)
✓ register: only can call by owner
✓ register: should addresses and rewards be match
✓ register not allowed once fully invetsed
✓ revoke: only can call by owner
✓ revoke: should emit event 'RevokedUpdated' after updated
✓ withdrawDust: revert WithdrawDustNotAllowedYet if trying to Dust withdraw before allowed
✓ withdrawDust: should only admin can call
✓ withdrawDust: should admin can withdraw all dust after the specified time (187ms)

mWom

veWom

✓ stake WOM and mint mWOM (1688ms)

convert wom to mWom and stake mWom into masterMagpie

✓ should mWom stake (2665ms)

✓ Convert and stake should work as normally after wom up is not set (2827ms)

wom up campagin

✓ Wom up should be true initially

✓ Depoist should simply deposit wom then mint mWom (967ms)

✓ Accumulated Wom should be locked by wombatStaking once wombatStaking is set (1925ms)

✓ If wom up is not set, deposit is not allwoed (151ms)

✓ Convert should work as normally after wom up is not set (1785ms)

events

✓ convert should emit 'mWomMinted' (1532ms)

✓ convertAndStake should emit 'mWomMinted' (2870ms)

✓ deposit should emit 'mWomMinted' (2664ms)

✓ Admin: should emit 'HelperSet' (87ms)

✓ Admin: should emit 'WombatStakingSet' (90ms)

✓ Admin: should emit 'WomUpSet' (89ms)

Negative Test Cases

Illegal withdrawal

✓ Player2 tries to withdraw Player1's balance... (2914ms)

✓ Player2 tries to withdraw something from wombatStaking directly... (3358ms)

✓ Player2 tries to withdraw something from masterMagpie directly...

✓ Player2 tries to call withdrawFor directly...

✓ Player2 tries to call withdrawVIMGPFor directly...

✓ Player2 tries to call masterMagpie multiclaimOnBehalf directly to get player1's reward...

✓ Player2 tries to call baseRewardPool getReward directly...

Admin functions Hacking

✓ WombatStaking: Try to registerPool

✓ WombatStaking: Try to setMWom

✓ WombatStaking: Try to setLockDays

✓ WombatStaking: Try to removePool

✓ WombatStaking: Try to updatePoolHelper

✓ WombatStaking: Try to setMasterMagpie

✓ WombatStaking: Try to setMasterWombat

✓ WombatStaking: Try to unlockAllVeWom (119ms)

✓ WombatStaking: Try to pause

✓ WombatStaking: Try to unpause (122ms)

✓ WombatStaking: Try to addFee

✓ WombatStaking: Try to setFee

✓ WombatStaking: Try to removeFee

✓ WombatStaking: Try to addBonusRewardForAsset
✓ MasterMagpie: Try to setPoolManagerStatus
✓ MasterMagpie: Try to setCompounder
✓ MasterMagpie: Try to setVlmgp
✓ MasterMagpie: Try to pause
✓ MasterMagpie: Try to unpause (102ms)
✓ MasterMagpie: Try to createRewarder
✓ MasterMagpie: Try to add
✓ MasterMagpie: Try to set
✓ MasterMagpie: Try to updateEmissionRate
✓ BaseRewarder: Try to updateEmissionRate
✓ BaseRewarder: Try to queueNewRewards
✓ mWom: Try to setHelper
✓ mWom: Try to setWombatStaking
✓ mWom: Try to setWomUp
✓ mWom: Try to lockAllWom
✓ vlMGP: Try to pause
✓ vlMGP: Try to pause (111ms)
✓ vlMGP: Try to setWhitelistForTransfer
✓ vlMGP: Try to setMasterChief
✓ vlMGP: Try to setCoolDownInSecs
✓ vlMGP: Try to setMaxSlots
✓ Compounder: Try to setHelper
✓ Compounder: Try to setConvertor
✓ Compounder: Try to setLocker
✓ Compounder: Try to addReward
✓ Compounder: Try to removeReward
✓ Airdrop: Try to register
✓ Airdrop: Try to adjustStartDate
✓ Airdrop: Try to withdrawDust
✓ mgpPoolHelper: Try to depositFor
✓ mgpPoolHelper: Try to authorize
✓ mgpPoolHelper: Try to unauthorize
✓ mWomPoolHelper: Try to depositFor
✓ mWomPoolHelper: Try to authorize
✓ mWomPoolHelper: Try to unauthorize

VLMGP

Initialization

✓ vlMgp transparent proxy should initialzied correctly

Sinlge Lock

✓ lock should put MPG in VLMGP and VLMGP should live in Master Magpie (759ms)
✓ totallock() and getUserTotalLocked() should return correctly (758ms)

✓ should emit NewLock (735ms)
✓ should get MGP and bonus reward after lock and start unlock (5429ms)
✓ should staked info of vlMGP is corrected (733ms)
✓ should emit UnlockStarts (1433ms)
✓ should revert if withdraw vlMGP from masterMagpie directly (735ms)
✓ should revert if unlock amount exceed locked amount (724ms)

multiple user multiple Lock

✓ lock should put MPG in VLMGP and VLMGP should live in Master Magpie (2116ms)
✓ totallock() and getUserTotalLocked() should return correctly (2135ms)

StartUnlock

✓ getUserTotalLocked should return 0 and getUserAmountInCoolDown should return correctly (2088ms)
✓ Master Magpie should have not MGP staked for users (1456ms)
✓ getUserUnlockSlotLength should return correctly (1497ms)
✓ getUserNthUnlockSlot shuold return correctly (1458ms)
✓ getUserUnlockingSchedule should return correctly (1439ms)
✓ getNextAvailableUnlockSlot should return correctly case 1 (1429ms)
✓ getNextAvailableUnlockSlot should return correctly case 2 (5310ms)
✓ should revert with AllUnlockSlotOccupied() if all slot are under unlocking (4958ms)
✓ After multiple start Unlock reading functions should return correctly (5108ms)
✓ start unlock with insufficient locked MPG (1461ms)

getNextAvailableUnlockSlot

✓ getNextAvailableUnlockSlot should return 0 if there is no unlock request
✓ getNextAvailableUnlockSlot should try extend array if there hasn't reacehd max slot (3473ms)
✓ if all lock occupied, getNextAvailableUnlockSlot should revert (4115ms)
✓ If array reached max slot and there is a slot unlocked (4480ms)

unlock

✓ There should be No MGP in master magpie even after unlock (1035ms)
✓ should revert with StillInCoolDown (695ms)
✓ should revert with BeyondUnlockLength (694ms)
✓ should revert with UnlockedAlready (925ms)

Cancel unlock

✓ cancel unlock should put vlMGP back to masterMagpie and lock MGP in vlMGP (2136ms)

admin functions

pause

✓ only owner can call

unpause

✓ only owner can call
✓ only paused

setWhitelistForTransfer

✓ only owner can call
✓ should emit WhitelistSet (121ms)

setMasterChief

✓ only owner can call
✓ should revert with InvalidAddress()
✓ should emit NewMasterChiefUpdated (184ms)

setCoolDownInSecs

✓ only owner can call
✓ should revert with InvalidCoolDownPeriod()
✓ should emit CoolDownInSecsUpdated (146ms)

setMaxSlots

✓ only owner can call
✓ should revert with MaxSlotCantLowered()
✓ should emit MaxSlotUpdated (121ms)

Wombat Poool Helepr

Initialization

✓ Pool Helper should created correctly (1582ms)

Deposit stablecoin and native token

✓ pool helper should stake receipt token to MasterMagpie with amount as deposit amount (3536ms)
✓ pool helper balance should return based on amount recorded in rewarder (3490ms)
✓ Pool total supply should return total deposited amount (3463ms)
✓ Master Magpie should have new lp token staked (3504ms)
✓ User should not receive WOM lp recepit token (3773ms)
✓ There should be some pending WOM reward (3902ms)
✓ Second Deposit should harvest (11552ms)
✓ should bnb be deposited into the pool (4097ms)

Deposit wombat LP token directly

✓ should has the balance in masterMagpie (3072ms)

Withdraw

✓ pool helper should unstake receipt token from MasterMagpie with amount as withdraw amount (4411ms)
✓ pool helper balance should return based on amount remained in rewarder (4033ms)
✓ Pool total supply should return total remained amount (4644ms)
✓ Master magpie should have correct remained lp token staked (4032ms)
✓ There should be some pending WOM reward (4889ms)
✓ Withdraw for should harvest (3879ms)

complex cases which someone deposit with stable and someone deposit with LP

✓ should every one deposit and withdraw successfully (34680ms)

WombatStaking

Initialization

✓ Wombat Staking should initiated correctly

Register Pool

✓ should creat a pool helper and rewarder (1533ms)

✓ tokenToPool should return wombat pool address by deposit token (1540ms)
✓ should add a pool on MasterMagpie (1479ms)
✓ register an active token should fail (1627ms)
✓ register pool should emit event (1581ms)
✓ only owner is allowed to register pool

Fees addition and set

✓ Fee should be added correctly (224ms)
✓ Fee should be set correctly (485ms)
✓ remove fee shuold substract totalFee (604ms)

Charge fee when wombatStaking send rewards

✓ Fee charge as WOM to a certain address (6397ms)
✓ Fee charge as mWOM to a certain address (9112ms)
✓ Fee charge as WOM to rewarder (9267ms)
✓ Fee charge as mWOM to rewarder (14687ms)
✓ Fee charge as mWOM to rewarder (11935ms)
✓ multitple fee (15634ms)
✓ multitple fee with multiple rewards (20013ms)

Only Pool Helper functions

✓ Deposit should only by pool helper (106ms)
✓ Withdraw should only by pool helper
✓ Deposit should only on active pool (304ms)

Deposit with 1 tester

✓ shuld revert with 'OnlyPoolHelper()' if call the contract directly
✓ shuld revert with 'OnlyActivePool()' if the pool is inactivate
✓ receipt token should 1 : 1 as wom lp token (5103ms)

Deposit with multi testers

✓ receipt token of testers should equal to their deposited (33618ms)

Withdraw with 1 tester

✓ shuld revert with 'OnlyPoolHelper()' if call the contract directly
✓ should revert with 'WithdrawAmountExceedsStaked()' (4714ms)
✓ receipt token should 1 : 1 as wom lp token (4743ms)

Withdraw with mulit testers

✓ remaining balance of tester1 must be 1/2 before withdraw (4944ms)
✓ remaining balance of tester2 must be 2/3 before withdraw (4942ms)
✓ remaining balance of tester3 must be 0 (5292ms)
✓ remaining totalDeposit of pool must be 1/2 of original after all tester withdraw once (16399ms)

Harvest

✓ rewardPerToken should equal to rewardWOMAmount / depositAmount after harvest (1500ms)
✓ earnedWomAmt should equal to rewardWOMAmount after harvest (1641ms)
✓ earnedWomBal should equal to rewardWOMAmount * 2 after deposit (7487ms)

Events for deposit, withdraw and harvest

5) result balance of testers should equal to their remaining balance

6) total result balance of all testers should equal to total remaining balance
7) the earned amount of testers should equal to estimated amount
8) total harvested wom amount of testers should equal to estimated amount

Events for deposit, withdraw and harvest

✓ should trigger corrected events after deposited (6187ms)
✓ should trigger corrected events after harvested before 2nd deposit (12872ms)
✓ should trigger corrected events after withdrew (10925ms)
✓ should trigger corrected events after register the pool (2984ms)
✓ should trigger corrected events after remove the pool (1932ms)
✓ should trigger corrected events after update the pool (2848ms)
✓ should trigger corrected events after setMasterMagpie (189ms)
✓ should trigger corrected events after setMasterWombat (188ms)
✓ should trigger corrected events after setMWom (195ms)
✓ should trigger corrected events after addFee (243ms)
✓ should trigger corrected events after setFee (477ms)
✓ should trigger corrected events after removeFee (466ms)
✓ should trigger corrected events after harvest if fee is setted (12786ms)

Admin functions

✓ registerPool should only by admin (673ms)
✓ setMWom should only by admin (679ms)
✓ removePool should only by admin
✓ updatePoolHelper should only by admin
✓ setMasterMagpie should only by admin
✓ setMasterWombat should only by admin
✓ addFee should only by admin
✓ setFee should only by admin
✓ removeFee should only by admin
✓ addBonusRewardForAsset should only by admin

When there are 2 different wombat pool for the same depositToken

✓ deposit into main pool and side pool should both work well (8378ms)

332 passing (49m)
8 failing

FILE	% STMTS	% BRANCH	% FUNCS
Mgp.sol	100	50	100
ProxyAdmin.sol	0	100	0
TransparentUpgradeableProxy.sol	0	0	0
VLMGP.sol	96.51	73.68	95.65
		rewards	
.../Airdrop.sol	100	87.5	100
.../BaseRewardPool.sol	78.46	92.86	77.78
.../MGPRelease.sol	89.74	77.78	85.71
.../ManualCompound.sol	100	91.67	100
.../MasterMagpie.sol	95.21	77.42	95
	93.18	83.1	91.57
		wombat	
.../SimplePoolHelper.sol	87.5	100	80
.../WombatPoolHelper.sol	100	50	100
.../WombatStaking.sol	91.67	80.77	84.38
.../mWOM.sol	95	83.33	83.33
	93.45	80.95	87.1
All files	**78.7**	**73.94**	**76.35**

CODE COVERAGE AND TEST RESULTS FOR ALL FILES FOR THE 2ND AUDIT REVISION (BY MAGPIE TEAM)

Airdrop2

Initialize

✓ Airdrop2 should created correctly
✓ should verify merkle tree ok

Users to claim

✓ should the user claim the first 5% reward at the start (264ms)
✓ should the user claim the next amount of the reward in the next week (523ms)
✓ should the user claim the correct reward after 3 weeks if he hasn't claimed before (250ms)
✓ should the user get full rewards if he claim once during and after vesting finished once (511ms)
✓ should the user claim the correct reward after vesting finished if he hasn't claimed before (255ms)
✓ should emit the correct event after claim (247ms)
✓ should revert if the user not in the list (330ms)
✓ should revert if the proof is not match or is forged

The same cases but with lock after claim

✓ should the user get the correct amount of vlMGP at the start (1693ms)
✓ should the user get the correct amount of vlMGP in the next week (2302ms)
✓ should the user get the correct amount of vlMGP after 3 weeks if he hasn't claimed before (2688ms)
✓ should emit the correct event after claim and lock (1236ms)

Admin functions

✓ setVlmgp: should emit 'VLMGPUpdated' (107ms)

should owner can call admin functions

✓ setVlmgp
✓ emergencyWithdraw

Base Rewarder Pool

Test Stake/Withdraw for functionatily

✓ withdraw for should udpate total suuply (8149ms)
✓ withdraw for should udpate user balances (12304ms)

Test reward distribution

✓ A more complicated case (14318ms)

Test reward distribution with additional reward

✓ should get all reward infos and players' earned amount (10395ms)

Admin functions

✓ should emit the event 'ManagerUpdated' with manual deployed pool (97ms)
✓ test for mWom rewarding case (3590ms)

WombatBribeManager

Initialization

✓ bribeManager should initialzied correctly
✓ should WombatVoter is set correctly
✓ should veWom is set correctly
✓ should WombatStaking is set correctly
✓ should vlMGP is set correctly
✓ should PancakeZapper is set correctly

✓ should the pool length is 2

should vote information is correct before any voting

✓ should totalVotes equals veWom balance
✓ should remainingVotes equals veWom balance (39ms)
✓ should totalVotes equals remainingVotes (38ms)
✓ should usedVote equals 0 before any vote
✓ should the ratio of vlMGP and veWOM is correct (71ms)
✓ should player total locked get from bribeManager is same with he locked (60ms)

Vote and unvote

✓ should vote operation is correct (4239ms)
✓ should vote amount be 0 after unvote (828ms)
✓ should players can vote with the amount within the locked amount exclude unlocking (1798ms)
✓ should vote amount can't exceed the locked amount exclude unlocking (1679ms)
✓ should revert if do vote after unlock all vlMGP (1484ms)
✓ should players can unlock MGP within the locked amount exclude unlocking (1794ms)
✓ should revert if unlock the amount in voted (815ms)
✓ should revert if unlock the amount in voted (492ms)

Cast vote

✓ vote 2 bribe pools (2905ms)
✓ should claim rewards after castVote and the caller should receive part of reward as fee (7508ms)
✓ same case of above with claimBribeFor (7865ms)
✓ same case of above with claimAllBribes (7863ms)
✓ vote, then cast votes, and unvote, and then cast votes again (6266ms)
✓ cast votes and claim bribes (7672ms)

multi pools, multi players case

✓ 2 players 2 pools vote and cast votes and unvote and unlock (12620ms)

Admin functions test

✓ setPancakeZapper (99ms)
✓ addPool: should the new bribe add correctly (212ms)
✓ addPool: should the lp not be address 0
✓ should the pool be removed by removePool (169ms)
✓ should revert with 'OutOfPoolIndex()' if the pool index to remove is out of range

Permission check

✓ should only owner can call setPancakeZapper
✓ should only owner can call addPool
✓ should only owner can call removePool

Master Magpie

✓ should userClaimableInfo updated correctly (9431ms)
✓ pending MGP should be correct after multiple deposit (1495ms)

Smart Wom Convert

✓ Initialization
✓ estimateTotalConversion (61ms)
✓ deposit with 70% convert (996ms)
✓ deposit with 70% convert with staking (1752ms)
✓ deposit with 0% convert (1002ms)
✓ deposit with 100% convert (665ms)
✓ deposit for with 30% convert (1022ms)
✓ deposit for with 30% convert with staking (1791ms)
✓ convert Ratio > DENOMINATOR should revert
✓ Revert if min Rec not met (131ms)

VLMGP

Cancel unlock

✓ cancel unlock should put vlMGP back to masterMagpie and lock MGP in vlMGP (2430ms)
✓ cancel unlock should fail if fully unlock (2276ms)
✓ cancel unlock should fail if already unlock (2370ms)

Force unlock

✓ force unlock should take penalty (3176ms)
✓ force unlock penalty (4018ms)
✓ should not unlock twice with the same slot (3195ms)
✓ should revert with 'NotInCoolDown()' if slot not in cool down (2257ms)

VLMgp can not withdraw by user from masterMagpie

✓ Try withdrawing vlMGP should fail (911ms)

VLMGP Base Rewarder Pool

Initialization

✓ should vlMGP is set correctly

Lock / Unlock

✓ should get the reward information (1793ms)
✓ should the totalStaked shuold remain the same after start unlocking (2272ms)
✓ mulit player unlocking (6623ms)
✓ should the player receive the unlocked MGP after unlock (4122ms)
✓ should the totalStaked will be correct after unlock (12131ms)

Force Unlock

✓ should receive correct MGP after force unlock (2767ms)
✓ should not unlock twice with the same slot (2411ms)
✓ should revert with 'NotInCoolDown()' if slot not in cool down (1367ms)

Reward

✓ should every players will get correct amount of rewards after harvest (12279ms)
✓ the case of startUnlock after queueNewRewards (15402ms)
✓ the case of startUnlock before queueNewRewards (13699ms)
✓ expectedPenaltyAmount (2302ms)

✓ calExpireForfeit (5763ms)
✓ 3 players unlock and claim case (29544ms)

queueMGP

✓ queue MGP with and without forfeit (5227ms)
✓ Multiple user recevie forfeit (9575ms)

get Reward

✓ get Reward without forfeit (14000ms)

WombatStaking

Charge fee when wombatStaking send rewards

✓ Fee should can be exempt (7592ms)

Admin functions

✓ setPoolRewardFeeFree should only by admin
✓ setBribe should only by admin
✓ setBribe should updated correctly (242ms)

432 passing (43m)

FILE	% STMTS	% BRANCH	% FUNCS
VLMGP.sol	100	75.61	100
BNBZapper.sol	0	0	0
BribeRewardPool.sol	72.09	45.24	72.22
ManualCompound.sol	90.24	87.5	100
MasterMagpie.sol	98.63	78.87	97.87
SmartWomConvert.sol	100	91.67	100
WombatBribeManager.sol	73.27	65.22	74.19
WombatStaking.sol	87.84	64.62	82.86
All files	**77.76**	**63.59**	**78.39**

CODE COVERAGE AND TEST RESULTS FOR ALL FILES FOR THE 1ST AUDIT REVISION

Tests written by Zokyo Security

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

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

Contract: Airdrop

Admin functions

- ✓ .register() Admin should register users allocations (77ms)
- ✓ .register() Not admin should not register users allocations (108ms)
- ✓ .register() Admin should not register uncorrect users allocations (53ms)
- ✓ .register() Admin should not register users allocations if Airdop already started
- ✓ .adjustStartDate() Admin should adjust start date (49ms)
- ✓ .adjustStartDate() Admin should not adjust start date if Airdrop already started
- ✓ .withdrawDust() Admin should withdraw dust after 21 months after start date (103ms)
- ✓ .withdrawDust() Admin should not withdraw dust before 21 months after start date (42ms)

External Functions

- ✓ .getClaimableAmount() User should get claimable amount for himself (82ms)
- ✓ .getClaimableAmount() User claimable amount = 0 if he is not register (47ms)
- ✓ .getBonusAmount() User should get bonus amount for himself end it = 0 if totalEndRemainingAllocation not updated
- ✓ .updateEndRemainingAllocation() User should update end remaining allocation if time passed (66ms)
- ✓ .getBonusAmount() User should get bonus amount for himself end it = 0 if totalBonus = 0 (69ms)

Contract: ManualCompound

- ✓ .compound() if locker = 0 and helperAddress = 0 transfer tokens to sender (10515ms)
- ✓ .compound() if locker != 0 and helperAddress = 0 approve and deposit (9785ms)
- ✓ .removeReward() Should revert if invalid index (65ms)

Contract: MasterMagpie

Pool and Mgp token operations

- ✓ Shouldn't add pool while staking token isn't contract (294ms)
- ✓ Shouldn't add pool while rewarder isn't contract (622ms)
- ✓ Shouldn't add pool while helper isn't contract (47ms)
- ✓ Shouldn't set already setted mgp token (86ms)
- ✓ Shouldn't set mgp token non contract (53ms)
- ✓ Shouldn't add existed pool (144ms)
- ✓ Shouldn't set exist pool with invalid staking token (108ms)

✓ Shouldn't set exist pool with invalid rewarder (100ms)
✓ Should set exist pool (229ms)
✓ Shouldn't set non exist pool (78ms)
✓ Should update emission rate (59ms)
✓ Should return nothing when pool have zero total points (121ms)
✓ Should get pool amount (56ms)
✓ Should get pool info (92ms)

Contract: MGPRelease

Test constructor

✓ Correct deploy (297ms)
✓ Error if startTimestamp before block.timestamp (1243ms)
✓ Error if endTimestamp before startTimestamp (267ms)
✓ Error if initialUnlockPercentage more than contract denominator (259ms)

Same specific functions

✓ .getVestingInfo()
✓ .claim() should revert AccountRevoked() if vesting revoked (61ms)

Contract: SimplePoolHelper

✓ .unauthorize() owner should unauthorize (89ms)

Contract: VLMGP

✓ Should lock MGP tokens (196ms)
✓ Should lock MGP for another user (232ms)
✓ Should start unlock (171ms)
✓ Should create maximum amount of unlock slots (441ms)
✓ Should not let create more unlock slots than limit (585ms)
✓ Should revert if trying to unlock more than locked balance (115ms)
✓ Should complete unlock (210ms)
✓ Should revert unlocking if provided slot is greater than users' number of slots (149ms)
✓ Should revert unlocking if provided slot is greater than maximum limit (452ms)
✓ Should cancel unlock (192ms)
✓ Should start unlock in an empty slot (661ms)
✓ Should not unlock if cooldown is not yet finished (183ms)
✓ Should not let unlock same slot more than once (261ms)
✓ Should not let cancel unlock if cooldown is ended (192ms)
✓ Should not let cancel same unlock slot more than once (198ms)
✓ Should pause/unpause (219ms)
✓ Setters work correctly (144ms)
✓ Should not let create vlMGP with zero max slots (1345ms)

Contract: WombatPoolHelper

Initialization

✓ Should setup pool helper correctly
✓ Should setup pool helper correctly

Deposit
✓ Should deposit native (14963ms)
✓ Should deposit (11199ms)
✓ Shouldn't deposit when wombat staking is paused (415ms)
✓ Shouldn't deposit not via pool helper (89ms)
✓ Shouldn't deposit in inactive pool (97ms)
✓ Shouldn't deposit when paused (252ms)
✓ Should deposit when unpaused (520ms)
✓ Shouldn't deposit native when isNative for pool is false (84ms)
✓ Should deposit LP (865ms)

Withdraw
✓ Should withdraw (2671ms)
✓ Should emergency withdraw via master magpie (373ms)
✓ Shouldn't withdraw via not pool helper (288ms)

Harvest
✓ Should harvest (854ms)
✓ Should claim rewards via master magpie (1373ms)
✓ Should harvest with fee (2176ms)
✓ Should harvest with fee while fee receiver is BaseRewardPool (1573ms)
✓ Shouldn't harvest from inactive pool (644ms)
✓ Should harvest without convert rewardToken to mWom (846ms)
✓ Should set and remove fee (88ms)
✓ Shouldn't set fee if fee is inactive (89ms)
✓ Should harvest with removed fee (1092ms)

Pool operations
✓ Should remove pool (2068ms)
✓ Should get pool token list
✓ Shouldn't register existed pool
✓ Should update pool helper (54ms)

Setters
✓ Should set mWom token
✓ Should set lock days
✓ Should set master magpie contract
✓ Should set master wombat contract

Base reward pool
✓ Should get reward token decimals
✓ Should get staking token
✓ Should get reward tokens length
✓ Should update manager (38ms)
✓ Should update reward info for user (895ms)
✓ Should get all earned correctly (913ms)
✓ Shouldn't push reward token zero address to rewardTokens during deploy (403ms)
✓ Shouldn't get reward by someone expect master magpie

✓ Shouldn't queue reward by someone expect reward manager

VeWom operations

✓ Should convert Wom to veWom (2072ms)

✓ Should unlock all veWom (594ms)

✓ Shouldn't unlock veWom if latest time bigger unlock time (303ms)

mWom operations

✓ Should set helper

✓ Should set wombat staking

✓ Shouldn't deposit if isWomUp false

✓ Should lock all wom (374ms)

✓ Shouldn't convert and stake if helper not set (451ms)

✓ Shouldn't convert and stake if wombat staking not set (131ms)

✓ Shouldn't convert when paused (264ms)

✓ Shouldn't convert and stake to non exist pool (1588ms)

✓ Should deposit (183ms)

WombatPoolHelper

mWom operations

✓ Should incentive deposit (5430ms)

✓ Should return maxSwapAmount via SmartWomConvert (684ms)

✓ Shouldn't set invalid ratio via SmartWomConvert

✓ Should incentive deposit with staking (16913ms)

107 passing (3m)

FILE	% STMTS	% BRANCH	% FUNCS
Mgp.sol	100	100	100
ProxyAdmin.sol	0	100	0
TransparentUpgradeableProxy.sol	0	0	0
VLMGP.sol	97.4	85.5	98.55
		rewards	
.../Airdrop.sol	100	100	100
.../BaseRewardPool.sol	96.92	87.5	100
.../ManualCompound.sol	100	96.15	100
.../MasterMagpie.sol	93.35	81.25	95.12
.../MGPRelease.sol	100	100	100
	98.45	**92.98**	**99.02**
		wombat	
.../SimplePoolHelper.sol	100	100	100
.../WombatPoolHelper.sol	100	100	100
.../WombatStaking.sol	96.53	92.31	96.88
.../mWOM.sol	97.5	100	100
	98.51	**98.8**	**99.22**
All files	**65.73**	**79.55**	**66.13**

Zokyo Security team has prepared a fork test on the BSC mainnet network to verify the security of contracts and interaction with the Wombat exchange in the conditions close to production. All the core logic connected to deposits, withdrawals, and reward distribution was carefully tested.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES FOR THE 2ND AUDIT REVISION (BY ZOKYO TEAM)

Contract: BNBZapper

Getters

✓ Should return route address (38ms)
✓ Should return preview amount for given token without setting up route (43ms)
✓ Should return preview amount for given token with setting up route (1248ms)
✓ Should return collective preview amount for multiple tokens (79ms)

Swap functions

✓ Should swap token to eth (976ms)
✓ Should swap token to eth with setted path (4530ms)
✓ Should do nothing if amount is 0

Admin functions

✓ Should revert add route for pair if called not from owner (66ms)
✓ Should add route for token
✓ Should withdraw erc20 token (267ms)
✓ Should withdraw eth
✓ Should revert if withdraw from non owner caller
✓ Should change max slippage (38ms)
✓ Should revert if new slippage is more than DENOMINATOR
✓ Should revert if try to set new slippage not from owner

Contract: SmartWomConvert

Initialization

✓ Should setup pool3 helper correctly

Converting

✓ Should convert wom to mWom (27759ms)
✓ Should convert wom to mWom and transfer to other user (406ms)
✓ Should revert convert wom to mWom if convert ratio is higher than DENOMINATOR (156ms)
✓ Should revert convert wom to mWom if min amount is too high (948ms)
✓ Should revert estimate if convert ratio is higher than DENOMINATOR (108ms)
✓ Should not revert estimate if amount is 0
✓ Should deposit converted mWom (846ms)
✓ Should convert mWom and stake it in the same transaction (282ms)
✓ Should do nothing if converting with 0 amount (62ms)

Contract: WombatBribeManager

Initialization

✓ Check getters

Pool operations

✓ Should remove pool (151ms)
✓ Should set pancake zapper (131ms)

Vote and unvote

✓ vote operation should be correct (5181ms)
✓ should harvest single pool (6761ms)
✓ should vote and cast (5267ms)
✓ should call vote via wombat staking only by bribe manager (2829ms)

✓ shouldn't vote via wombat staking with invalid params (2125ms)
✓ vote amount should be 0 after unvote (827ms)
✓ should claim rewards after claimAllBribes() (9681ms)
✓ should send rewards after donation (7524ms)
✓ should revert swap for bnb if pancake zapper isn't set (3416ms)
✓ should increase queued rewards with donation (248ms)
✓ should revert if token is unknown (62ms)

Branch tests

✓ should que new rewards (84ms)
✓ should que new rewards when total supply & queued rewards > 0 (141ms)
✓ should donate new rewards when total supply & queued rewards > 0 (128ms)
✓ modifier checks (85ms)
✓ wrong constructor arguments (2444ms)
✓ withdraw without claiming reward (149ms)

Contract: VLMGP

✓ Shouldn't start unlock if totalLock < userTotalVotedInVlmgp (126ms)
✓ Should set penalty destination only by owner
✓ Should set bribe manager only by owner
✓ Should not let create vlMGP with zero max slots (396ms)
✓ Should not transfer penalty when penalty destination is address zero
✓ Should transfer penalty only by owner
✓ Shouldn't call specific functions while pause (98ms)

FILE	% STMTS	% BRANCH	% FUNCS
VLMGP.sol	100	92.68	100
BNBZapper.sol	100	94.44	100
BribeRewardPool.sol	100	88.1	100
ManualCompound.sol	90.24	90.63	100
MasterMagpie.sol	98.63	85	97.87
SmartWomConvert.sol	100	100	100
WombatBribeManager.sol	73.27	65.22	74.19
WombatStaking.sol	93.92	85	94.29
All files	**94.51**	**87.63**	**95.79**

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

zokyo