SHARKSWAP

SMART CONTRACT AUDIT

zokyo

August 8th 2022 | v. 1.0

Security Audit Score

PASS

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

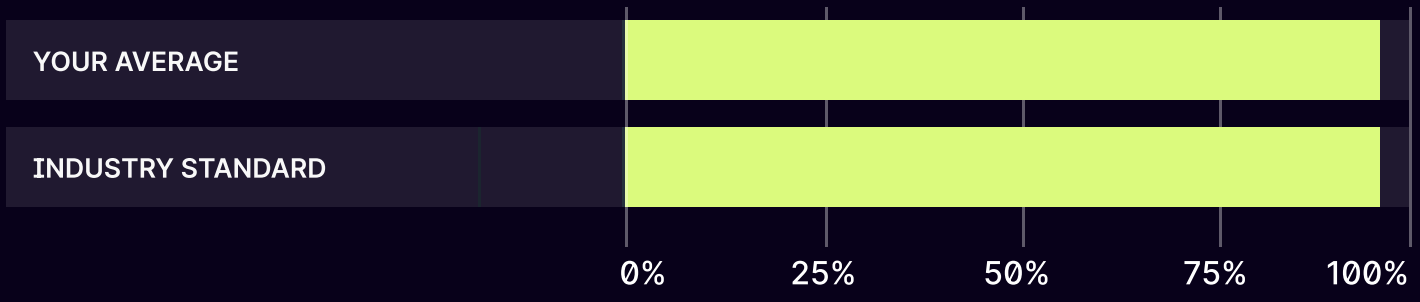
TECHNICAL SUMMARY

This document outlines the overall security of the SharkSwap smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

Testable Code



The testable code is 95%, which is corresponds 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, we at Zokyo recommend that the SharkSwap team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the SharkSwap repository.

Repository: https://github.com/mrwillis/sharkswap-contracts

Last audited commit: e41258510a340f6266cad1166a47e9dedb12a355

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

- CloneRewarderTime.sol
- MiniChefV2.sol
- SharkToken.sol
- SushiMaker.sol
- UniswapV2ERC20.sol
- UniswapV2Factory.sol
- UniswapV2Pair.sol
- UniswapV2Router02.sol
- UniswapV2Library.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Uses methods safe from reentrance attacks;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of SharkSwap smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes 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

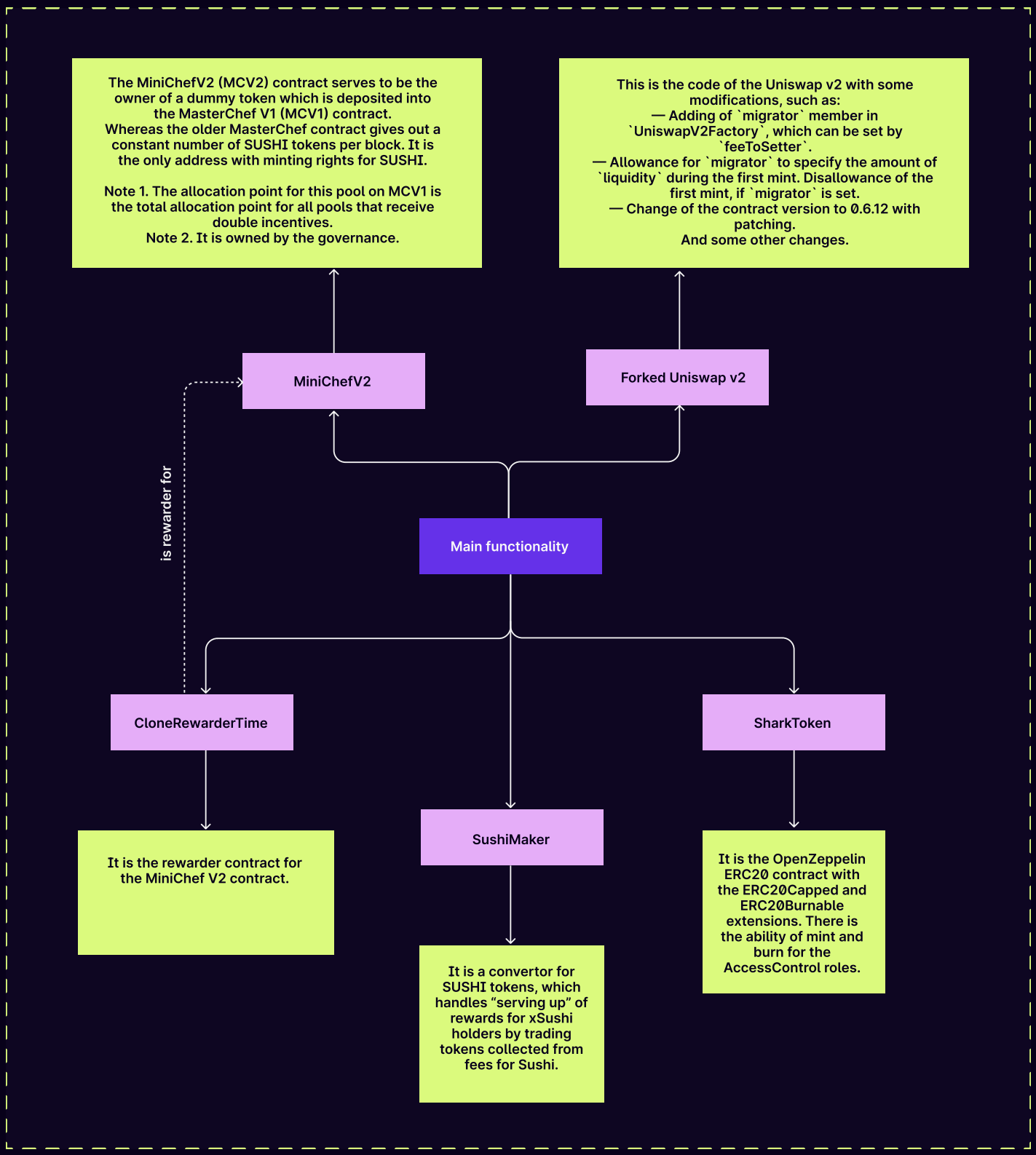
Zokyo Security has carefully audited the whole set of contracts provided by the SharkSwap team. First of all, a post mortem regarding the found vulnerability was studied to detect whether it is still in the code. The issue was successfully fixed by the SharkSwap team before the beginning of the audit process.

After that, all the amendments made to Uniswap V2 contracts were checked for any possible issues. There were no vulnerabilities found within the changes added to the Uniswap V2 contract by the SharkSwap team.

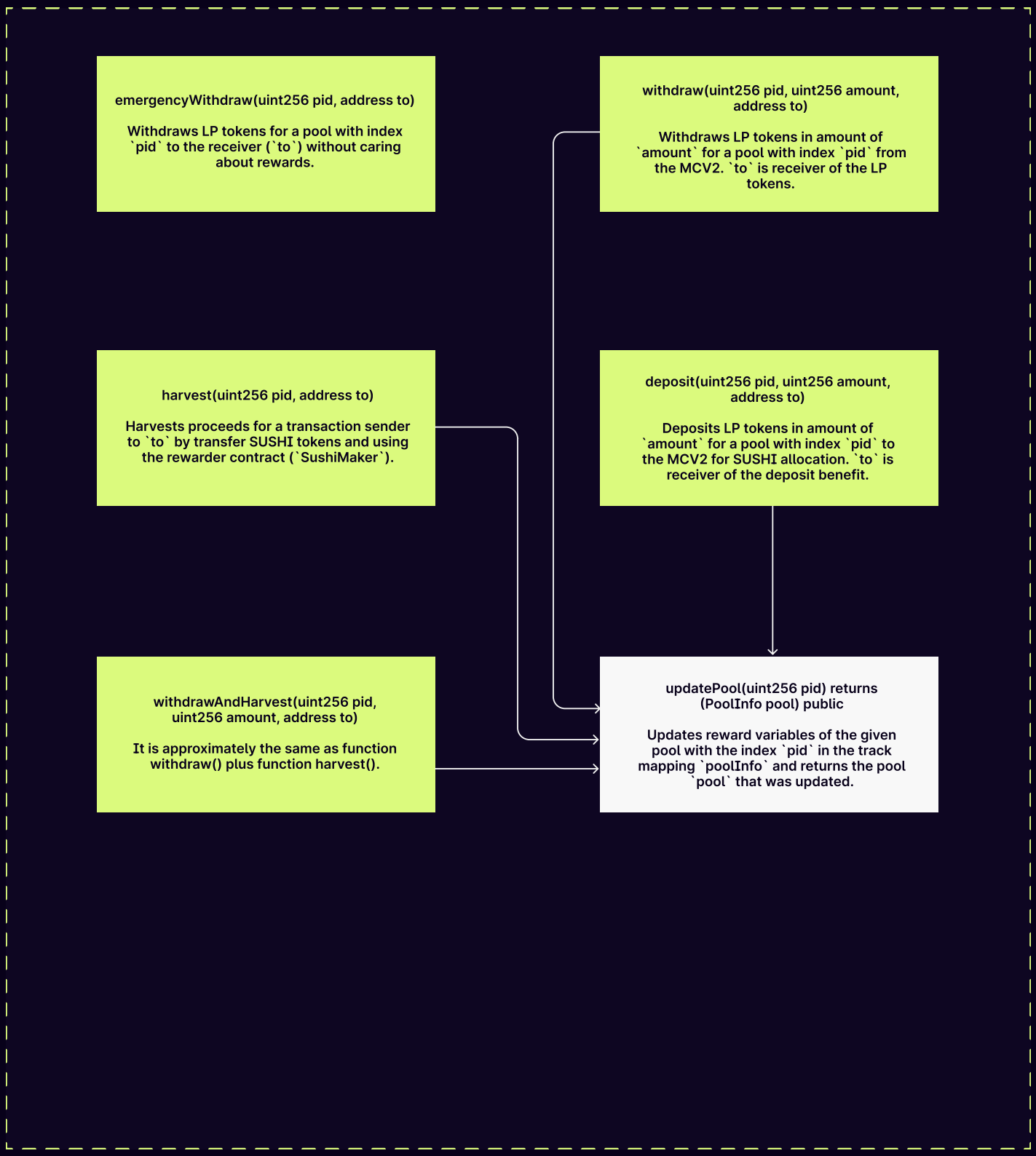
The scope of contracts also contains a MasterChief-like staking protocol. These contracts were also checked to safely store users' funds and distribute rewards correctly. No critical issues were found within these contracts. We found 2 high, 2 informational and 1 low issue connected to the correctness of rewards distribution, commented code, gas optimizations and staking logic. The issues of high severity were verified by the team, but no changes were made in the code. However, the issues have no security threat as they are connected to the correctness of rewards distribution. The SharkSwap team ensured that they will perform all the necessary actions and will manually send notifications to the users about the safety of their rewards. Other issues connected to code optimization were left for future updates.

The overall security of the protocol is high enough. The code is well documented and tested by the SharkSwap team. Zokyo Security has prepared additional unit-test suit to verify the correctness of the business logic of the contracts.

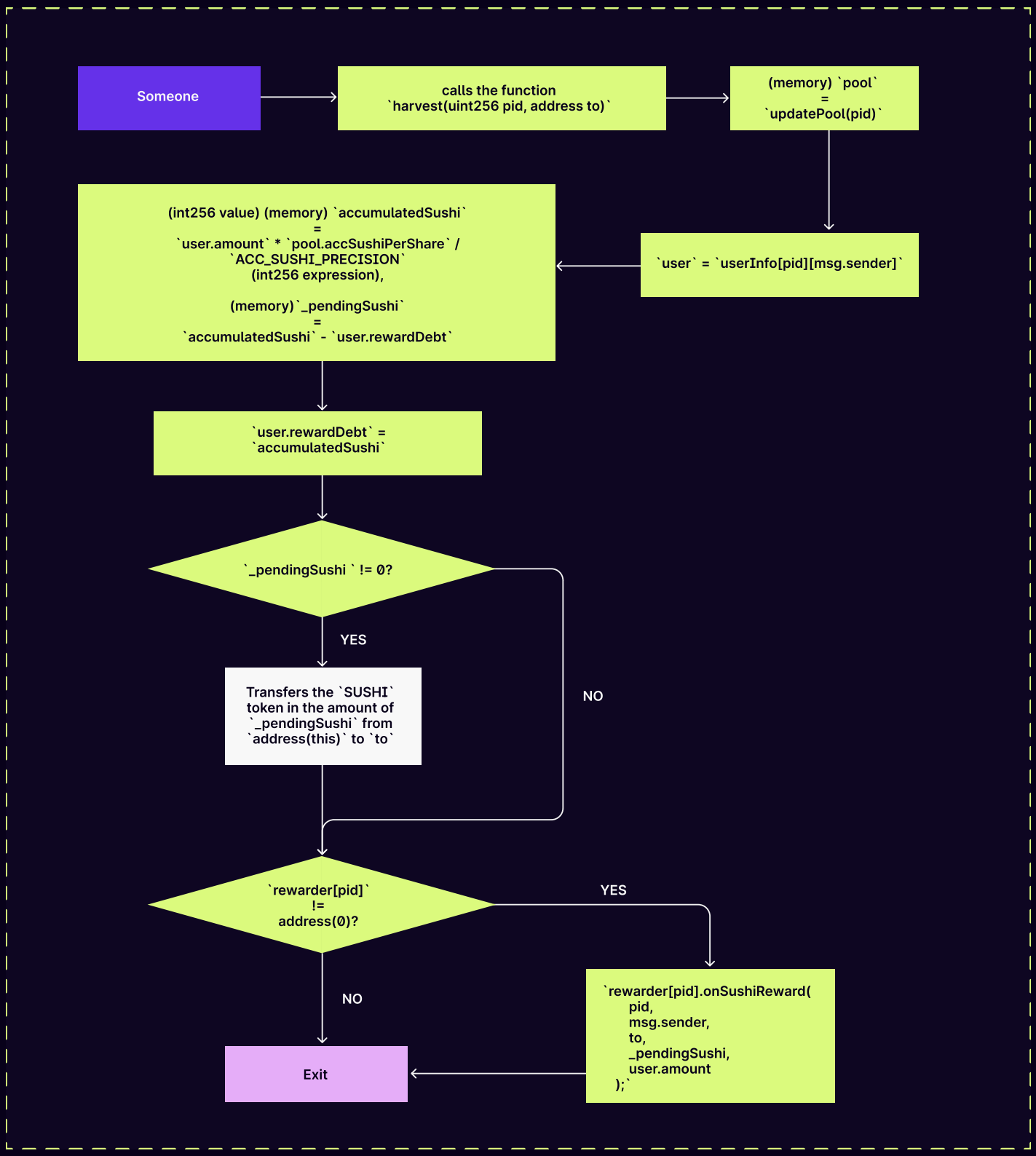
PROTOCOL OVERVIEW



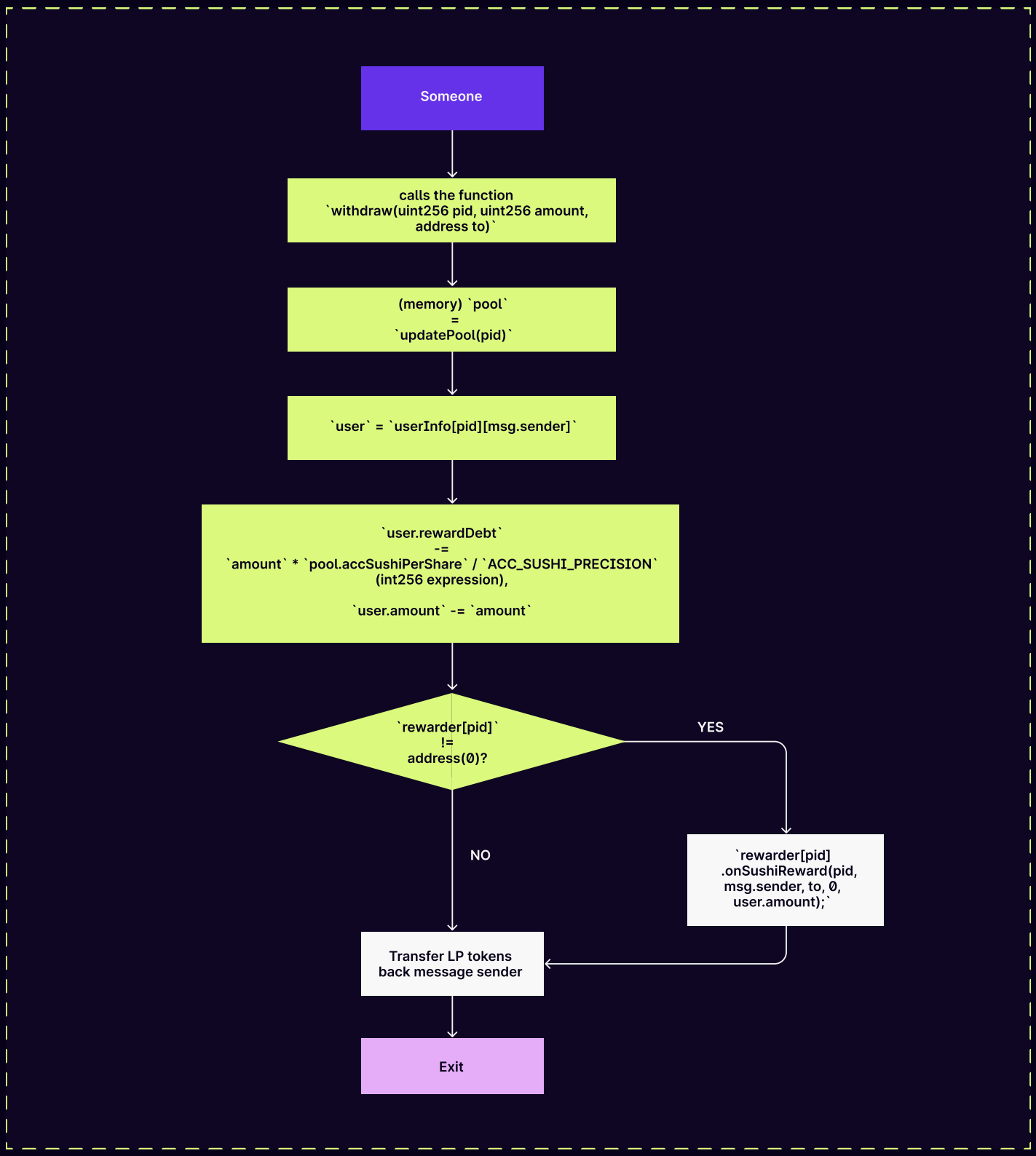
DESCRIPTION OF MINICHEFV2 MAIN FUNCTIONS



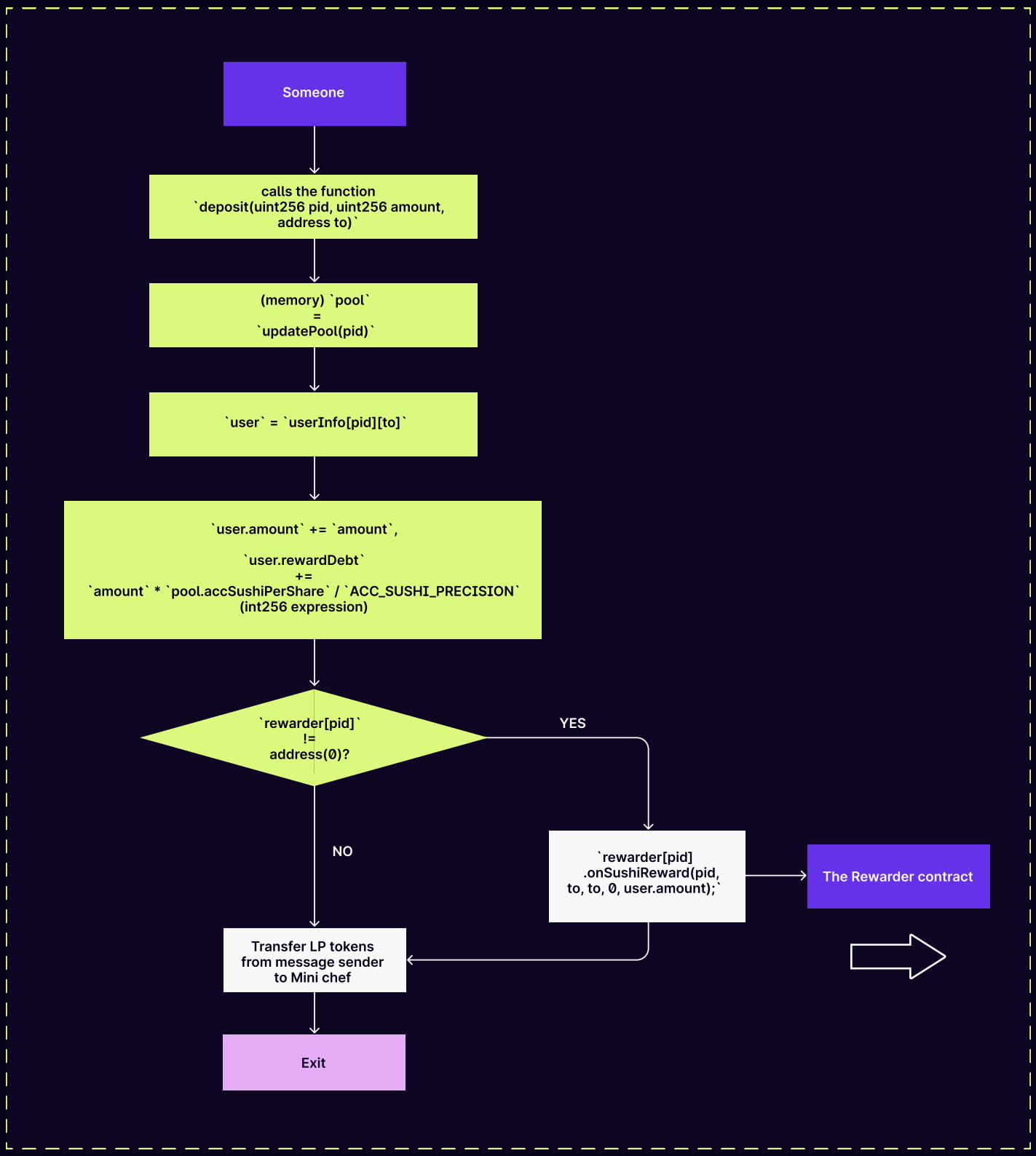
MINICHEFV2 MAIN USER FUNCTIONS



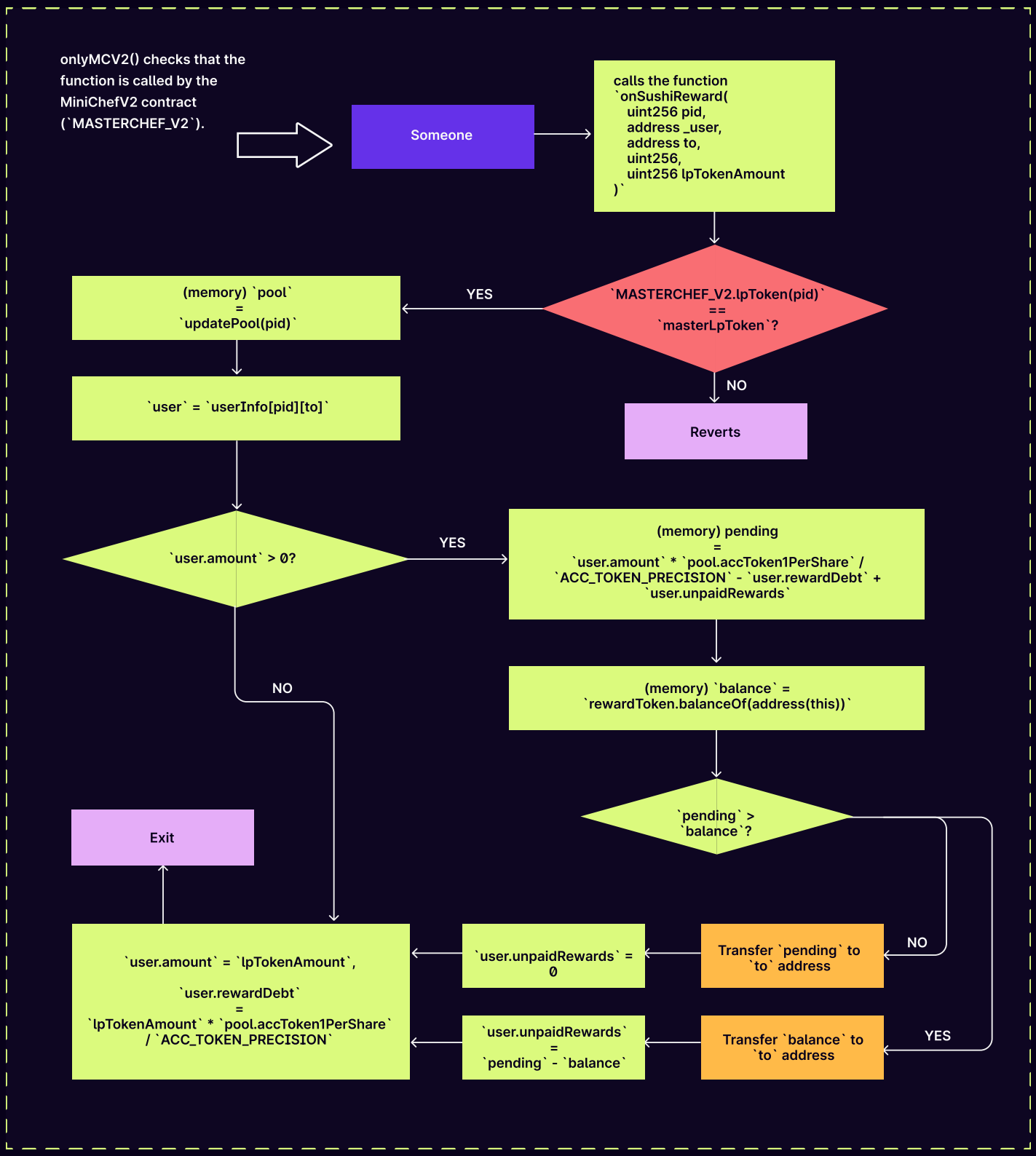
MINICHEFV2 MAIN USER FUNCTIONS



MINICHEFV2 MAIN USER FUNCTIONS



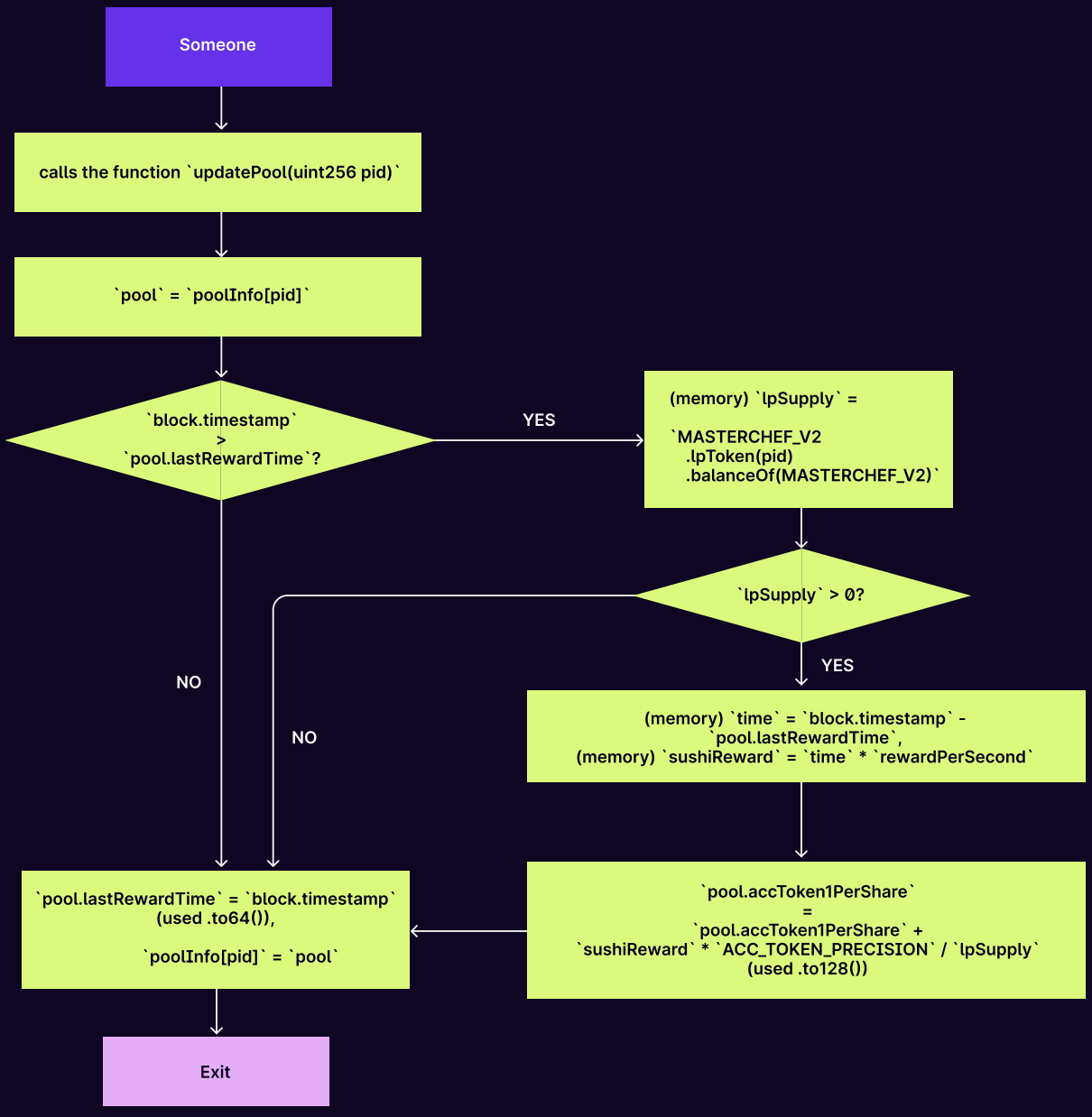
CLONE REWARDER FUNCTIONS



CLONE REWARDER FUNCTIONS

updatePool(uint256 pid) returns (PoolInfo pool) public

Updates reward variables of the given pool with the index `pid` in the track mapping `poolInfo` and returns the pool `pool` that was updated. Here, this pool is an `PoolInfo` struct object: uint128 `accToken1PerShare` and uint64 `lastRewardTime`.



STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of 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. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

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

High

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

Medium

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

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

HIGH | RESOLVED

The owner can change the reward rate.
CloneRewarderTime.sol: function setRewardPerSecond().
Function allows the owner to change reward per second without updating the current "accToken1PerShare" of the pools. This way, users might lose their pending rewards in case the owner sets a lower value or a `0`.

Recommendation:
Update all the pools before setting a new reward per second value.

Post audit:
The SharkSwap team verified that the Governance contract will always be calling "updatePool" function before setting new parameters and users will be notified within the dApp about all future changes in advance. However, such mandatories are not present in the current code.

HIGH | VERIFIED

The owner can set the pool settings without updating the pool.
MiniChefV2.sol: function set().
The owner can set new allocation points to the pool without updating the pool, which can lead to the loss of users' pending rewards. The owner can also reset the rewarder contract for the pool, which can also lead to the loss of pending rewards on the previous rewarder.

Recommendation:
Update the pool before setting new parameters. Consider transferring the ownership of the contract to the DAO in the future and notify users about all the changes in advance so that they can claim their rewards in time.

Post-audit:
The SharkSwap team verified that the ownership is set to multi-sig DAO contract and users will be notified about any changes in advance.

LOW | VIRIFIED

Rewards are claimed for “to” address instead of msg.sender.
MiniChefV2.sol: function deposit(), line 209.
Based on the logic of other functions, such as withdraw(), harvest(), the rewards should be claimed for msg.sender and transferred to “to” address. However, during depositing, rewards are claimed for “to” address by passing “to” as the second parameter in _rewarder.onSushiReward() call. The issue is marked as low since it doesn’t cause any fund loss for the provided address, but it should be verified whether this logic works as intended.

Recommendation:
Pass msg.sender as a second parameter in _rewarder.onSushiReward() to claim rewards for msg.sender or verify that rewards should be claimed for “to” address.

From the client.
According to the team, “to” parameter in the “deposit” function is a beneficiary, unlike the cases of withdrawal or harvest, when the beneficiary is msg.sender. Hence, the different params passed to onSushiReward.

INFORMATIONAL | UNRESOLVED

Incomplete code sections.
There are to-dos in the source code of the SushiMaker contract on lines 88, 208, 215.

Recommendation:
Resolve the issues with these sections.

From the client.
Fixes are left for future updates.

Unnecessary gas consumption.
The following functions are not called internally but have public visibility.

- CloneRewarderTime.sol: init(), setRewardPerSecond(), reclaimTokens(), transferLocks().
- MiniChefV2.sol: poolLength(), add(), set(), setSushiPerSecond(), setMigrator(), migrate(), deposit, withdraw(), harvest(), withdrawAndHarvest(), emergencyWithdraw().
- SharkToken.sol: mint().
- SushiMaker.sol: setDestination().
- UniswapV2Router02.sol: quote(), getAmountOut(), getAmountIn(), getAmountsOut(), getAmountsIn().

Recommendation:
They should be declared external.

From the client.
Fixes are left for future updates.

	CloneRewarderTime.sol	MiniChefV2.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

	SharkToken.sol	SushiMaker.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

	UniswapV2ERC20.sol	UniswapV2Factory.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

	UniswapV2Pair.sol	UniswapV2Router02.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

UniswapV2Library.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 SharkSwap 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 SharkSwap contract requirements for details about issuance amounts and how the system handles these.

- ✓ SharkToken should mint/burn/burnFrom (69ms)
- ✓ Add LP to the pool/set pool and deposit (80ms)
- ✓ Set sushiPerSec/update pools (46ms)
- ✓ Set migrator
- ✓ deposit and withdraw/harvest/emergency (402ms)
- ✓ clone rewards view functions (51ms)
- ✓ clone rewards set rewards
- ✓ set bridges
- ✓ convert assets (1317ms)
- ✓ clone rewards init and get sushi (167ms)
- ✓ uniswap (314ms)

FILE	% STMTS	% BRANCH	% FUNCS
CloneRewarderTime.sol	91.23	86.36	100
MiniChefV2.sol	100	92.86	100
SharkToken.sol	100	100	100
SushiMaker.so	98.41	96.88	100
All files	**97.41**	**94**	**100**

FILE	% STMTS	% BRANCH	% FUNCS
UniswapV2ERC20.sol	80.77	16.67	88.89
UniswapV2Factory.sol	100	100	100
UniswapV2Pair.sol	87.27	55.77	100
UniswapV2Router02.sol	28.07	17.5	43.48
UniswapV2Library.sol	100	60	100
All files	**79.22**	**49.98**	**86.47**

Note: Zokyo Security has written additional tests to verify the difference between the original Uniswap V2 and SharkSwap contracts. All additional implementations made by the SharkSwap team were tested by the team of auditors.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the SharkSwap team

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

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

MiniChefV2
- PoolLength
  - ✓ PoolLength should execute
- Set
  - ✓ Should emit event LogSetPool
  - ✓ Should revert if invalid pool
- PendingShark
  - ✓ PendingShark should equal ExpectedShark (64ms)
  - ✓ When time is lastRewardTime (56ms)
- MassUpdatePools
  - ✓ Should call updatePool
  - ✓ Updating invalid pools should fail
- Add
  - ✓ Should add pool with reward token multiplier
- UpdatePool
  - ✓ Should emit event LogUpdatePool
  - ✓ Should take else path
- Deposit
  - ✓ Depositing 0 amount
  - ✓ Depositing into non-existent pool should fail
- Withdraw
  - ✓ Withdraw 0 amount
- Harvest
  - ✓ Should give back the correct amount of SHARK and reward (98ms)
  - ✓ Harvest with empty user balance
  - ✓ Harvest for SHARK-only pool (72ms)
- EmergencyWithdraw
  - ✓ Should emit event EmergencyWithdraw (56ms)

SharkToken

- ✓ should have correct name, symbol, decimals, totalSupply
- ✓ should only allow MINTER_ROLE to mint token
- ✓ should be possible to burn token
- ✓ should be possible to burn from another account
- ✓ should only allow BURNER_ROLE to burn token
- ✓ should not exceed capped limit
- ✓ should supply token transfers properly
- ✓ should fail if you try to do bad transfers

SushiMaker

setBridge

- ✓ does not allow to set bridge for Sushi
- ✓ does not allow to set bridge for WETH
- ✓ does not allow to set bridge to itself
- ✓ emits correct event on bridge

convert

- ✓ should convert SUSHI - ETH (68ms)
- ✓ should convert USDC - ETH (85ms)
- ✓ should convert $TRDL - ETH (90ms)
- ✓ should convert USDC - SUSHI (68ms)
- ✓ should convert using standard ETH path (89ms)
- ✓ converts MIC/USDC using more complex path (98ms)
- ✓ converts DAI/USDC using more complex path (100ms)
- ✓ converts DAI/MIC using two step path (143ms)
- ✓ reverts if it loops back
- ✓ reverts if caller is not EOA
- ✓ reverts if pair does not exist
- ✓ reverts if no path is available (53ms)

convertMultiple

- ✓ should allow to convert multiple (162ms)

Uniswap

- ✓ should have params
- ✓ should have balances
- ✓ should be able to transfer
- ✓ should be able to add liquidity pair
- ✓ should use swap fee param
- ✓ should be able to swap
- ✓ should use maker fee param

FILE	% STMTS	% BRANCH	% FUNCS
CloneRewarderTime.sol	0	0	0
MiniChefV2.sol	73	60.71	80
SharkToken.sol	100	83.33	100
SushiMaker.so	96.83	96.88	90.91
All files	**67.45**	**60.23**	**67.72**

FILE	% STMTS	% BRANCH	% FUNCS
UniswapV2ERC20.sol	50	0	55.56
UniswapV2Factory.sol	75	37.5	55.56
UniswapV2Pair.sol	80	48.08	83.33
UniswapV2Router02.sol	5.26	2.5	17.39
UniswapV2Library.sol	25.71	20	25
All files	**47.19**	**21.61**	**47.36**

We are grateful to have been given the opportunity to work with the SharkSwap team.

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

Zokyo's Security Team recommends that the SharkSwap team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo