Cyberscope

Audit Report

SquadSwap

December 2023

Repository https://github.com/Bit5Tech

Projects MCV2, Syrup, SquadSwapClaimMerkle

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/Bit5Tech
SquadSwapClaimMerkle Commit	b77ad52c2b3fada84f6035aeee6c08d835038964
Syrup Commit	9f68dd153c2b384e2aaadb275fe2d6914d8bb474
MCv2 Commit	137ec382085e6d10da3f6412fd5319974520e051

Audit Updates

Initial Audit	28 Dec 2023

Source Files

Filename	SHA256
SquadAirdrop.sol	5f64f4abac581ad7a0969427a240a6bbc05bf5cc679da074373d342e0a855ee5
SafeBEP20.sol	f01f2ad629742079a931aa84f034cdeabf80779bb33a3b23a3d7afb37345ba9c
ReentrancyGuard.sol	ebc56e91dbbca5585a31a3238b65cc61e25c68479911b37673abdff2dc828327
MasterChefV2.sol	946435bf6e49538cc1fc92543f0c65c695782da1160571a79218acf9f2e9dded
interfaces/IMasterChef.sol	49aff9dcf50ddc05652074a728949e3246e66d9fcf5bed9f24e173c1b552bf39
interfaces/IBEP20.sol	3cf00ef4f36ad64814aeb752058f03ec91786cc3d803265d0db8d6b33bb34f05
/SquadProfile.sol	ba247726ecb89626dcb952e61f927d9e77303348b6fb0087b4148a7d84b887a1

/SmartChefInitializable.sol	be02d5afa0cd89ffe8e83c8b5319d7ce4d91aefb7087ca6d92f45aa2e9f41d23
/SmartChefFactory.sol	07fcf71a4c33185f69dfcb1a7e36b318acca1eb2dd2a091cddb43dd166fc35fa
/interfaces/IPancakeProfile.sol	4224afd5e553b50b3fa6911190bc3b3b6e68d1ae7662e79a45877812934801b2

Overview

SquadAirdrop Contract

The SquadAirdrop contract is designed to manage a token airdrop process. It is structured to allow users to claim tokens based on certain criteria, ensuring a fair and transparent distribution. The contract employs a Merkle Tree, a data structure that enables efficient and secure verification of large data sets, to manage claims, ensuring that only eligible participants can claim the tokens.

Key features of the contract include the tracking of the total amount of tokens claimed, a mechanism to prevent double claiming by a single address, and time-bound conditions for claiming tokens. The contract also incorporates administrative functions, allowing the owner to manage the airdrop process. These functions include opening and closing the claim process, recovering unclaimed tokens after the claim period, and updating various parameters like the claim period and wallet addresses. The contract's design reflects a focus on security and efficient management of the airdrop process, with an emphasis on ensuring that only eligible participants can claim their tokens and that the process is conducted within a specified timeframe.

Syrup Contracts

The Syrup Contracts consist of three interconnected smart contracts, each serving a distinct purpose within a decentralized application ecosystem, related to token staking and user profiles in a blockchain-based platform.

The `SmartChefFactory` , acts as a factory for creating new staking pool contracts. It allows the deployment of individual staking pool contracts (SmartChefInitializable) with specific parameters such as staked and reward tokens, reward per block, start and end blocks, and user limits. This contract ensures that each staking pool has its unique settings and addresses, facilitating the management and creation of multiple staking pools by the contract owner.

The `SmartChefInitializable` , is a staking contract where users can deposit (stake) tokens to earn rewards. It includes mechanisms for depositing and withdrawing staked tokens, calculating rewards, and handling emergency withdrawals. This contract also

integrates with the Pancake Profile system, allowing for additional features like user point systems and profile management based on certain conditions.

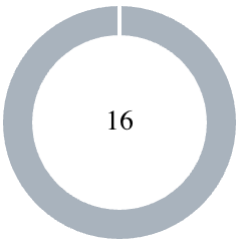
The `SquadProfile`, is designed for user profile management within the ecosystem. It allows users to create and manage their profiles. The contract includes functions for creating, pausing, updating, and reactivating user profiles, as well as managing user points and teams within the platform.

Overall, these contracts work together to create a comprehensive system for token staking and user engagement, with features like reward distribution, profile management, and team participation, enhancing the functionality and interactivity of a decentralized platform.

MasterChefV2 Contract

The `MasterChefV2` contrac, which is an advanced iteration of a yield farming contract. This contract is designed to manage the distribution of the `SQUAD` token to various liquidity pools. It allows users to deposit liquidity provider (LP) tokens into different pools and earn rewards in SQUAD tokens based on the amount staked and the duration of the stake. The contract features mechanisms for adding new pools, updating pool parameters, and managing user stakes and rewards, including handling deposits, withdrawals, and emergency withdrawals. It also incorporates a system for reward calculation based on user shares and pool allocation points, with an added complexity of boost multipliers to incentivize certain behaviors or stakeholders. Additionally, the contract includes safety features and administrative functions, such as reward rate adjustments and whitelist management, to ensure flexibility and control over the farming ecosystem.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	16

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	16	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CCR	Contract Centralization Risk	Unresolved
	RC	Repetitive Calculations	Unresolved
	IWL	Inconsistent Withdrawal Logic	Unresolved
	MC	Missing Check	Unresolved
	MEM	Missing Error Messages	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L06	Missing Events Access Control	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L08	Tautology or Contradiction	Unresolved
	L09	Dead Code Elimination	Unresolved
	L11	Unnecessary Boolean equality	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L16	Validate Variable Setters	Unresolved

●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L32,33,45,50,56,61
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

The contract uses a Merkle Proof mechanism in order to define many applicable addresses. The verification process is based on an off-chain configuration. The contract owner is responsible for updating the in-chain “Merkle Root” in order to validate correctly the provided message.

Additionally, the tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

Finally, the contract owner has the authority to start and dtop the claim functionality.

```
    constructor(
        bytes32 _merkleRoot,
        address _tokenAddress,
        address _squadNFTWallet,
        address _devWallet
    ) {
        owner = msg.sender;
        merkleRoot = _merkleRoot;
        ...
    }

    function claim(uint256 amount, bytes32[] calldata merkleProof)
public {
        ...
        bytes32 leaf = keccak256(abi.encodePacked(msg.sender,
amount));
        require(
            verifyMerkleProof(merkleProof, merkleRoot, leaf),
            "Invalid proof"
        );
        ...
    }
```

```
 constructor(
        bytes32 _merkleRoot,
        address _tokenAddress,
        ...
    ) {
        ...
        token = IERC20(_tokenAddress);
        ...
    }

    function claim(uint256 amount, bytes32[] calldata merkleProof)
public {
        ...
        token.transfer(msg.sender, amount);
        ...
        }
```

```
    function openClaim() external onlyOwner {
        claimOpened = true;
        claimStartTime = block.timestamp;
    }

    function closeClaim() external onlyOwner {
        claimOpened = false;
    }
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

We state that the Merkle Proof algorithm is required for proper protocol operations and gas consumption decrease. Thus, we emphasize that the Merkle proof algorithm is based on an off-chain mechanism. Any off-chain mechanism could potentially be compromised and affect the on-chain state unexpectedly. The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Additionally, it is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the presale tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L71,72
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive. Specifically, the division to the `balance` variable is calculated twice.

```
token.transfer(squadNFTWallet, balance / 2);
token.transfer(devWallet, balance - balance / 2);
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

IWL - Inconsistent Withdrawal Logic

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L65,87
Status	Unresolved

Description

The contract is currently designed with two separate functions, `recoverAirdropBalance` and `withdrawTokens`. The `recoverAirdropBalance` function is specifically intended for recovering the airdrop tokens, and it includes additional checks in order to prevent the recover before a certain timestamp. On the other hand, the `withdrawTokens` function allows the owner to withdraw any token (including the airdrop tokens) directly to the owner address without any specific checks. As a result the `withdrawTokens` function can bypass the checks and the intended distribution logic in `recoverAirdropBalance`. This could lead to scenarios where the airdrop tokens are withdrawn prematurely and not distributed as intended.

```
    function recoverAirdropBalance() external onlyOwner {
        require(
            claimStartTime + claimPeriod < block.timestamp,
            "still claiming"
        );
        uint balance = token.balanceOf(address(this));
        token.transfer(squadNFTWallet, balance / 2);
        token.transfer(devWallet, balance - balance / 2);
    }

  function withdrawTokens(IERC20 _token) external onlyOwner {
        uint256 balance = _token.balanceOf(address(this));
        require(_token.transfer(msg.sender, balance), "Transfer
failed");
    }
```

Recommendation

It is recommended to consolidate these two functions into one to streamline the withdrawal process and ensure consistent handling of tokens. This consolidated function should include necessary checks and conditions to handle different scenarios, such as airdrop recovery and general token withdrawal, appropriately. If the intention is to maintain separate functions for specific use cases, then it is crucial to modify the `withdrawTokens` function to prevent the withdrawal of airdrop tokens. This can be achieved by implementing additional checks within `withdrawTokens` to prevent the withdraw of the airdrop token. This approach will enhance the contract's security and integrity by preventing unauthorized or unintended token withdrawals.

MC - Missing Check

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L76
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues. Specifically the `_newPeriod` varaible should be greater than the current block timestamp in order to ensure that time-based conditions are set correctly and to avoid potential issues with setting periods in the past.

```
    function updateClaimPeriod(uint _newPeriod) external
onlyOwner {
        claimPeriod = _newPeriod;
    }
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L40,42
Status	Unresolved

Description

The contract is missing error messages. There is no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(claimStartTime + claimPeriod >= block.timestamp)
require(claimOpened == true)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	MasterChefV2.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L75,79,83,87,92MasterChefV2.sol#L61,79,167,168,169,170,205,206,207,228,260,276,298,334,361,378,397,398,399,400,425,436,445,460,461,462,492,502,503,504
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint _newPeriod
address _squadNFTWallet
address _devWallet
IERC20 _token
address _newAddress
IBEP20 public immutable SQUAD
uint256 public immutable MASTER_PID
uint256 _allocPoint
IBEP20 _lpToken
bool _isRegular
bool _withUpdate
uint256 _pid
address _user
uint256 _amount

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L93
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
owner = _newAddress
```

Recommendation

To avoid this issue, it's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L76
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
claimPeriod = _newPeriod
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	MasterChefV2.sol#L172
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(_lpToken.balanceOf(address(this)) >= 0, "None BEP20
tokens")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	SafeBEP20.sol#L19,30,49,68,80,99ReentrancyGuard.sol#L61,71,81
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function safeTransfer(
        IBEP20 token,
        address to,
        uint256 value
    ) internal {
        _callOptionalReturn(
            token,
            abi.encodeWithSelector(token.transfer.selector, to,
value)
        );
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L42
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(claimOpened == true)
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	MasterChefV2.sol#L237,240,243,244,284,287,323,349,477,508,509
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
user.rewardDebt =
user.amount.mul(multiplier).div(BOOST_PRECISION).mul(pool.accSquadPerShar
e).div(
            ACC_SQUAD_PRECISION
        )
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L35,36,80,84,93MasterChefV2.sol#L128
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
devWallet = _devWallet
squadNFTWallet = _squadNFTWallet
owner = _newAddress
burnAdmin = _burnAdmin
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L2SafeBEP20.sol#L2ReentrancyGuard.sol#L4MasterChefV2.sol#L3
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.12;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	SquadAirdrop.sol#L50,71,72
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
token.transfer(msg.sender, amount)
token.transfer(squadNFTWallet, balance / 2)
token.transfer(devWallet, balance - balance / 2)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

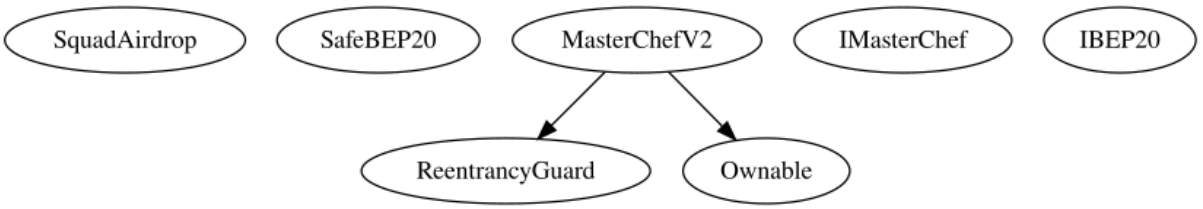
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SquadAirdrop	Implementation			
		Public	✓	-
	claim	Public	✓	-
	openClaim	External	✓	onlyOwner
	closeClaim	External	✓	onlyOwner
	recoverAirdropBalance	External	✓	onlyOwner
	updateClaimPeriod	External	✓	onlyOwner
	updateSquadNFTWallet	External	✓	onlyOwner
	updateDevWallet	External	✓	onlyOwner
	withdrawTokens	External	✓	onlyOwner
	transferOwnership	External	✓	onlyOwner
	viewTokenAddress	Public		-
	viewSquadNFTWallet	Public		-
	viewDevWallet	Public		-
	viewOwner	Public		-
	viewTotalClaimedAmount	Public		-
	verifyMerkleProof	Internal		

SafeBEP20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
ReentrancyGuard	Implementation			
		Public	✓	-
	_nonReentrantBefore	Private	✓	
	_nonReentrantAfter	Private	✓	
	_reentrancyGuardEntered	Internal		
MasterChefV2	Implementation	Ownable, ReentrancyGuard		
		Public	✓	-
	poolLength	Public		-
	add	External	✓	onlyOwner
	set	External	✓	onlyOwner
	pendingSquad	External		-
	massUpdatePools	Public	✓	-
	squadPerBlock	Public		-
	squadPerBlockToBurn	Public		-

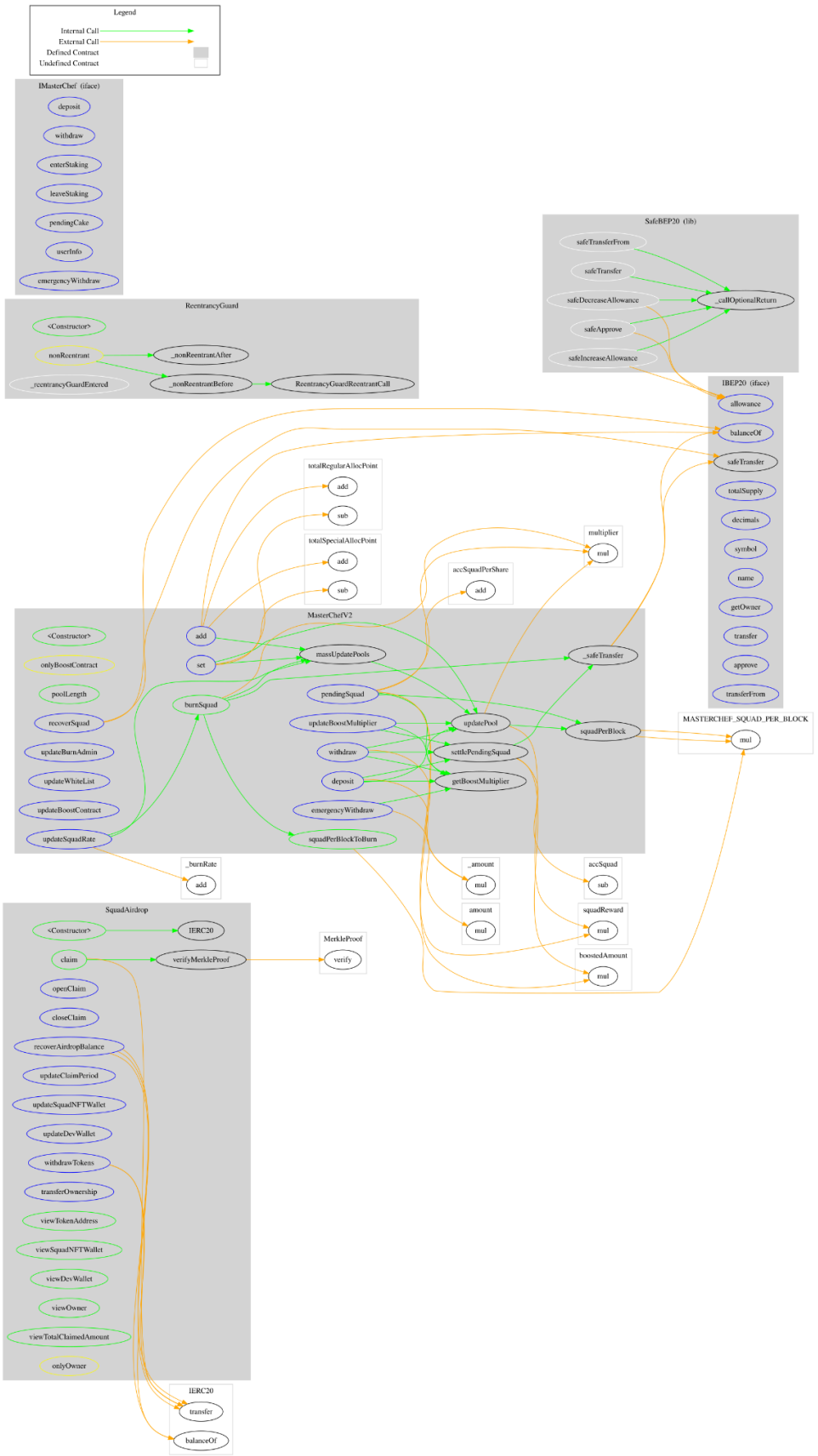
	updatePool	Public	✓	-
	deposit	External	✓	nonReentrant
	withdraw	External	✓	nonReentrant
	emergencyWithdraw	External	✓	nonReentrant
	burnSquad	Public	✓	onlyOwner
	updateSquadRate	External	✓	onlyOwner
	updateBurnAdmin	External	✓	onlyOwner
	updateWhiteList	External	✓	onlyOwner
	updateBoostContract	External	✓	onlyOwner
	updateBoostMultiplier	External	✓	onlyBoostContract nonReentrant
	getBoostMultiplier	Public		-
	settlePendingSquad	Internal	✓	
	_safeTransfer	Internal	✓	
	recoverSquad	External	✓	onlyOwner
IMasterChef	Interface			
	deposit	External	✓	-
	withdraw	External	✓	-
	enterStaking	External	✓	-
	leaveStaking	External	✓	-
	pendingCake	External		-
	userInfo	External		-
	emergencyWithdraw	External	✓	-

IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IMasterChef (iface)
deposit
withdraw
enterStaking
leaveStaking
pendingCake
userInfo
emergencyWithdraw
ReentrancyGuard
<Constructor>
nonReentrant
_nonReentrantAfter
_reentrancyGuardEntered
_nonReentrantBefore
ReentrancyGuardReentrantCall
SafeBEP20 (lib)
safeTransferFrom
safeTransfer
safeDecreaseAllowance
_callOptionalReturn
safeApprove
safeIncreaseAllowance
IBEP20 (iface)
allowance
balanceOf
safeTransfer
totalSupply
decimals
symbol
name
getOwner
transfer
approve
transferFrom
totalRegularAllocPoint
add
sub
totalSpecialAllocPoint
add
sub
multiplier
mul
accSquadPerShare
add
MasterChefV2
<Constructor>
onlyBoostContract
poolLength
recoverSquad
updateBurnAdmin
updateWhiteList
updateBoostContract
updateSquadRate
add
set
burnSquad
massUpdatePools
pendingSquad
updateBoostMultiplier
withdraw
deposit
emergencyWithdraw
squadPerBlockToBurn
updatePool
settlePendingSquad
getBoostMultiplier
_safeTransfer
squadPerBlock
MASTERCHEF_SQUAD_PER_BLOCK
mul
_burnRate
add
_amount
mul
amount
mul
accSquad
sub
squadReward
mul
boostedAmount
mul
SquadAirdrop
<Constructor>
IERC20
claim
verifyMerkleProof
openClaim
closeClaim
recoverAirdropBalance
updateClaimPeriod
updateSquadNFTWallet
updateDevWallet
withdrawTokens
transferOwnership
viewTokenAddress
viewSquadNFTWallet
viewDevWallet
viewOwner
viewTotalClaimedAmount
onlyOwner
MerkleProof
verify
IERC20
transfer
balanceOf

Summary

SquadSwap contract implements a decentralized yield farming mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io