Cyberscope

Audit Report

BOBE

April 2025

Repository	https://github.com/bobe-tech
Commit	13240b197533eb9667df77da30c530fb9caf9f6a
Audited by	© cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope’s smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/bobe-tech
Commit	13240b197533eb9667df77da30c530fb9caf9f6a

Audit Updates

Initial Audit	13 Mar 2025 https://github.com/cyberscope-io/audits/blob/main/1-bobe/v1/audit.pdf
Corrected Phase 2	08 Apr 2025 https://github.com/cyberscope-io/audits/blob/main/1-bobe/v2/audit.pdf
Corrected Phase 3	30 Apr 2025

Source Files

Filename	SHA256
TokenContract.sol	537f8e0125ee9bd9dd89ec16e9f01451091bcaaac3eaed85a6dfc25181a33db8
SwapContract.sol	a223222d469cd38c0fae6a468219058bc1b383c92b977a1939c09b8bc59c33eb
StakingContract.sol	3d42d5ce11b40225345dc0ac566ac841f7c8d0eb425901f23e2318e9418f69e0

Overview

TokenContract

The `TokenContract` is an ERC20 token contract designed to manage different token supplies, including liquidity, marketing, and team allocations. The contract ensures controlled distribution by allowing the owner to transfer tokens from the contract to specific addresses while maintaining limits on available amounts for each category.

Contract Details

- **Liquidity Supply**: 80% of the total supply is allocated for liquidity.
- **Marketing Supply**: 12% is reserved for marketing purposes.
- **Team Supply**: The remaining 8% is allocated to the team, with a lock period of 548 days before tokens can be transferred.

Transfer Liquidity

The `transferLiquidity` function allows the owner to transfer liquidity tokens to a specified address. It ensures that the transfer amount does not exceed the remaining liquidity and prevents transfers if the contract's balance is insufficient.

Transfer Marketing

The `transferMarketing` function operates similarly to the liquidity transfer but is specifically for marketing tokens. Only the owner can transfer marketing tokens, and it is protected by the same validation checks. Additionally, only unlocked marketing tokens can be transferred as the contract establishes unlocking periods.

Transfer Team

The `transferTeam` function enables the owner to transfer team tokens, but only after the lock period of 548 days has passed. It ensures that no tokens are transferred before the unlock time and checks that the contract holds enough tokens.Additionally, only unlocked team tokens can be transferred as contract establishes unlocking periods.

Recover Tokens

The `recoverTokens` function allows the owner to recover any leftover tokens in the contract, ensuring that only excess tokens (beyond the locked amounts) are transferred back to the owner.

Team Unlock

The `teamUnlockIn` function provides the remaining time until the team tokens can be unlocked, ensuring transparency on the unlock schedule.

SwapContract

The `SwapContract` facilitates the swapping of tokens (BNB, stablecoins, etc.) for a main token. It integrates with Chainlink for BNB price feeds and provides functionality for purchasing the main token using various stablecoins or native tokens.

Token Initialization

The `initialize` function sets the initial admin, main token price, and supported stablecoins. It also connects to the BNB price feed via Chainlink.

Token Management

The contract allows the admin to set the main token address, update the token price, and manage the allowed stablecoins through `setMainTokenAddress` , `allowStableToken` , and `disallowStableToken` functions.

Token Swaps

- **Swap Native Token**: This function allows users to swap BNB for the main token. It uses the BNB price feed to determine the equivalent amount in USDT, which is then used to calculate the main token amount.
- **Swap Stable Tokens**: This function allows users to swap stablecoins for the main token. It requires that the stablecoin is allowed and works similarly to the native token swap.
- **Swap Any Tokens**: This function allows users to swap any supported tokens for the main token by interacting with a smart router contract.

StakingContract

The `StakingContract` enables users to stake tokens and earn rewards. It supports depositing, staking, unstaking, and claiming rewards. The contract also allows the admin to announce campaigns, set durations, and manage rewards distribution.

Contract Details

- **Staking Token**: The token users stake to earn rewards.
- **Rewards Token**: The token distributed as rewards.
- **Campaign Duration**: The duration of staking campaigns.
- **Unstake Period**: The period after which staked tokens become unlockable for withdrawal.

Token Initialization

The `initialize` function sets up the staking and rewards tokens and ensures they are initialized before staking can occur.

Deposit and Announce

- **Deposit**: Users can deposit rewards tokens into the contract.
- **Announce**: The admin announces the start of a new staking campaign, including the amount of rewards allocated for the campaign and the duration.

Staking and Unstaking

- **Stake**: Users can stake tokens, increasing both their local and global stake.
- **Unstake**: Users can withdraw their staked tokens after the unstake period, provided their tokens are unlockable.

Claiming Rewards

Users can claim their accumulated rewards via the `claimRewards` function. The rewards are calculated based on the global and local stake indices.

Global and User Stats

The contract provides detailed statistics on both individual users and the overall staking campaign. Users can check their stake, pending rewards, and other metrics, while the global stats show total staked amounts and rewards distribution.

Roles

TokenContract

Owner

The owner of the contract has full control over token distribution and can interact with the following functions:

- `function transferLiquidity(address to, uint256 value)`
- `function transferMarketing(address to, uint256 value)`
- `function transferTeam(address to, uint256 value)`
- `function recoverTokens(IERC20 token)`

Retrieval Functions

The following functions provide details about token distribution and availability:

- `function teamUnlockIn()`
- `function getUnlockedMarketingAmount()`
- `function getUnlockedTeamAmount()`
- `function getNextMarketingUnlock()`
- `function getNextTeamUnlock()`
- `function getUnlockStatus()`

SwapContract

Admin

The admin (DEFAULT_ADMIN_ROLE) has full control over configuration and can interact with the following functions:

- `function setUsdtAddress(address newUsdtAddress)`
- `function setBnbPriceFeed(address newPriceFeedAddress)`
- `function setSmartRouterAddress(address newRouterAddress)`
- `function setMainTokenAddress(address newMainTokenAddress)`
- `function setFundingAddress(address newFundingAddress)`
- `function allowStableToken(address token)`
- `function disallowStableToken(address tokenAddress)`

Users

Users can interact with the following functions:

- `function swapNativeToken()`

- `function swapStableTokens(address token, uint256 amountIn)`
- `function swapAnyTokens(address tokenIn, uint256 amountIn, address[] calldata path, uint256 userSlippageBps)`

Retrieval Functions

The following functions provide information on conversions and available swaps:

- `function convertBnbToUsdt(uint256 amount)`
- `function convertDecimals(uint256 value, uint256 sourceDecimals, uint256 targetDecimals)`

Staking Contract

Admin

The admin (DEFAULT_ADMIN_ROLE) can configure key parameters and interact with the following functions:

- `function setCampaignDuration(uint256 newDuration)`
- `function setTokenAddresses(address newStakingToken, address newRewardsToken)`
- `function setUnstakePeriod(uint256 newPeriod)`
- `function announce(uint256 rewardsAmount)`
- `function depositAndAnnounce(uint256 depositAmount)`

Announcer

The announcer (ANNOUNCER_ROLE) is responsible for launching staking campaigns and can interact with:

- `function announce(uint256 rewardsAmount)`
- `function depositAndAnnounce(uint256 depositAmount)`

Users

Users (stakers) can interact with the following functions:

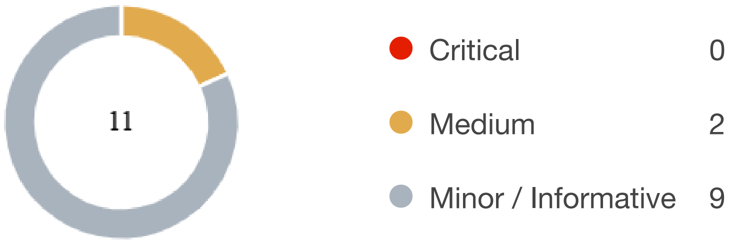
- `function deposit(uint256 amount)`
- `function stake(uint256 amount)`
- `function unstake(uint256 amount)`
- `function claimRewards()`

Retrieval Functions

The following functions can be used to retrieve staking-related information:

- `function getAvailableRewards()`
- `function getUnlockableAmount(address user)`
- `function index()`
- `function rewards(address addr)`
- `function getUserStats(address user)`
- `function getGlobalStats()`
- `function getStakersRewardsBatch(uint256 offset, uint256 batchSize)`
- `function getRewardsByAddresses(address[] memory addresses)`

Findings Breakdown



●	Critical	0
●	Medium	2
●	Minor / Informative	9

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	2	0	0	0
●	Minor / Informative	9	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🟠	SBAA	Staking Before Announcing Allowed	Unresolved
🟠	AME	Address Manipulation Exploit	Unresolved
⚪	CCR	Contract Centralization Risk	Unresolved
⚪	DPI	Decimals Precision Inconsistency	Unresolved
⚪	PIFR	Potential Initialization Front Running	Unresolved
⚪	PMRM	Potential Mocked Router Manipulation	Unresolved
⚪	PTRP	Potential Transfer Revert Propagation	Unresolved
⚪	STPC	Stable Token Price Concern	Unresolved
⚪	TSI	Tokens Sufficiency Insurance	Unresolved
⚪	OCTD	Transfers Contract's Tokens	Unresolved
⚪	L13	Divide before Multiply Operation	Unresolved

SBAA - Staking Before Announcing Allowed

Criticality	Medium
Location	StakingContract.sol#L154
Status	Unresolved

Description

`StakingContract` allows users to stake before the `announce` function is called. If the staking is never announced then the tokens will stay locked until the `unstakePeriod` , providing no value.

```
function stake(uint256 amount) public {
    //...
    IERC20(stakingToken).safeTransferFrom(_msgSender(),
address(this), amount);
    //...
}
```

Recommendation

The team could consider adding some functionality that restricts users from staking before the announcement.

AME - Address Manipulation Exploit

Criticality	Medium
Location	SwapContract.sol#L191,216
Status	Unresolved

Description

The contract's design includes functions that accept external contract addresses as parameters without performing adequate validation or authenticity checks. This lack of verification introduces a significant security risk, as input addresses could be controlled by attackers and point to malicious contracts. Such vulnerabilities could enable attackers to exploit these functions, potentially leading to unauthorized actions or the execution of malicious code under the guise of legitimate operations.

While the contract may be protected from reentrancies, allowing users to insert any address to perform swaps still carries significant risks as it may harm the entire ecosystem of the protocol or create unintended opportunities for buyers.

```
function swapAnyTokens(address tokenIn, uint256 amountIn, address[]
calldata path, uint256 userSlippageBps) external nonReentrant {
    //...

IPancakeSwapV3Router(smartRouterAddress).swapExactTokensForTokens(actualA
mountIn, minAmountOut, path, fundingAddress);
    //...
}
```

Recommendation

To mitigate this risk and enhance the contract's security posture, it is imperative to incorporate comprehensive validation mechanisms for any external contract addresses passed as parameters to functions. A possible solution could be to include checks against a whitelist of approved addresses. Implementing validations helps prevent malicious exploits and ensures that only trusted contracts can interact with sensitive functions.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	TokenContract.sol#L48,69,82,90 SwapContract.sol#L57,67,80,87,93,105,111,126 StakingContract.sol#L70,78,91,113,132
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.

```
function transferLiquidity(address to, uint256 value) external
onlyOwner
function transferMarketing(address to, uint256 value) external
onlyOwner
function transferTeam(address to, uint256 value) external
onlyOwner
function recoverTokens(IERC20 token) external onlyOwner
```

```
function initialize(address adminMultisigAddress, address
fundingMultisigAddress) public initializer
function setUsdtAddress(address newUsdtAddress) external
onlyRole(DEFAULT_ADMIN_ROLE)
function setBnbPriceFeed(address newPriceFeedAddress) external
onlyRole(DEFAULT_ADMIN_ROLE)
function setSmartRouterAddress(address newRouterAddress)
external onlyRole(DEFAULT_ADMIN_ROLE)
function setMainTokenAddress(address newMainTokenAddress)
public onlyRole(DEFAULT_ADMIN_ROLE)
function setFundingAddress(address newFundingAddress) public
onlyRole(DEFAULT_ADMIN_ROLE)
function allowStableToken(address token) public
onlyRole(DEFAULT_ADMIN_ROLE)
function disallowStableToken(address tokenAddress) public

onlyRole(DEFAULT_ADMIN_ROLE)
```

```
function setCampaignDuration(uint256 newDuration) public
onlyRole(DEFAULT_ADMIN_ROLE)
function setTokenAddresses(address newStakingToken, address
newRewardsToken) public onlyRole(DEFAULT_ADMIN_ROLE)
function setUnstakePeriod(uint256 newPeriod) public
onlyRole(DEFAULT_ADMIN_ROLE)
function announce(uint256 rewardsAmount) public {
    require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()) ||
hasRole(ANNOUNCER_ROLE, _msgSender()), "Caller must be admin or
announcer");
    //..
}
function depositAndAnnounce(uint256 depositAmount) public {
    require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()) ||
hasRole(ANNOUNCER_ROLE, _msgSender()), "Caller must be admin or
announcer");
    //...
}
```

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.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	StakingContract.sol#L249
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals are set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

In this case, in the contract's functionality it is assumed that both the staking token and the reward token have 18 decimals. Specifically in the `_rate` function.

```
function _rate() private view returns (uint256) {
    if (globalStake == 0) return 0;
    return 1e18 * scRewardsAmount / globalStake /
(scFinishTimestamp - scStartTimestamp);
}
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals
Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

PIFR - Potential Initialization Front Running

Criticality	Minor / Informative
Location	SwapContract.sol#L54 StakingContract.sol#L60
Status	Unresolved

Description

The contract lacks proper access control during its initialization phase, making it vulnerable to front-running attacks. An unauthorized third party could invoke the initialization process before the intended deployer, gaining control over administrative roles and critical system functions. This could lead to asset mismanagement, unauthorized fund withdrawals, or operational disruptions. Since the initializer modifier only prevents re-execution but does not restrict access, the contract is exposed to potential takeovers.

```
function initialize(address adminMultisigAddress, address
fundingMultisigAddress) public initializer {
    //...
}
```

```
function initialize(address adminMultisigAddress, address
announcerMultisigAddress) public initializer {
    //...
}
```

Recommendation

To mitigate this risk, access to the initialization process should be restricted to authorized team members. This can be enforced by validating the caller against a predefined deployer address or implementing a secure deployment process that prevents external entities from intervening. Additionally, using a deployment script that finalizes initialization in the same transaction as contract creation can eliminate front-running opportunities.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	TokenContract.sol#L87
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the router address. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```
function setSmartRouterAddress(address newRouterAddress)
external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(newRouterAddress != address(0), "Router address
cannot be zero");
    smartRouterAddress = newRouterAddress;
    emit SmartRouterSet(newRouterAddress);
}
```

Recommendation

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	SwapContract.sol#L157,158
Status	Unresolved

Description

The contract sends funds to `adminAddress` as part of the `swapNativeToken` function. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the `SwapContract` and revert the function.

```
(bool success, ) = payable(fundingAddress).call{value:
msg.value}("");
require(success, "Failed to send BNB");
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main contract's flow. This could be achieved by sending the funds in a non-revertable way, while also ensuring that a gas exhaustion does not disrupt the execution flow.

STPC - Stable Token Price Concern

Criticality	Minor / Informative
Location	SwapContract.sol#L62
Status	Unresolved

Description

The price of the token is set once in the `initialize` function and cannot be changed after. Having a stable price may result in unwanted opportunities for buyers. For example if the DEX price of the token is larger than the contract's, buyers may buy large amounts from the contract to then sell it to the DEX and receive more value. Additionally, if the token's DEX price is lower, users will not buy from the contract and only perform swaps in the DEX.

```
mainTokenPriceInUsdt = 1_100_000_000_000_000_000;
```

Recommendation

It is recommended that the team considers the scenarios mentioned above. Additionally, the team can consider adding functionality to change the token price.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	StakingContract.sol#L99
Status	Unresolved

Description

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.

```
function deposit(uint256 amount) public {
    require(amount > 0, "Amount must be > 0");
    require(tokensInitialized, "Token addresses must be set 
first");
    uint256 balanceBefore = 
IERC20(rewardsToken).balanceOf(address(this));
    IERC20(rewardsToken).safeTransferFrom(_msgSender(), 
address(this), amount);
    uint256 balanceAfter = 
IERC20(rewardsToken).balanceOf(address(this));
    uint256 actualAmount = balanceAfter - balanceBefore;
    deposited += actualAmount;
    emit Deposit(actualAmount);
}
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the 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.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	TokenContract.sol#L48,69,82
Status	Unresolved

Description

The contract owner has the authority to claim the majority of the balance of the contract. The owner may take advantage of it by calling the `transferLiquidity` , `transferMarketing` and `transferTeam` function.

```
function transferLiquidity(address to, uint256 value) external
onlyOwner
function transferMarketing(address to, uint256 value) external
onlyOwner
function transferTeam(address to, uint256 value) external
onlyOwner
```

Recommendation

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	TokenContract.sol#L110,115,123,131
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.

```
uint256 elapsedPeriods = (block.timestamp - unlockStart) /
UNLOCK_PERIOD
uint256 currentlyUnlocked = (totalSupply * elapsedPeriods *
UNLOCK_PERCENTAGE) / 100
```

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.

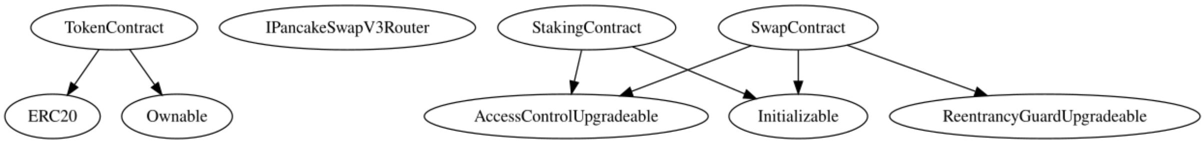
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
TokenContract	Implementation	ERC20, Ownable		
		Public	✓	ERC20 Ownable
	transferLiquidity	External	✓	onlyOwner
	_transferUnlockedTokens	Private	✓	
	transferMarketing	External	✓	onlyOwner
	transferTeam	External	✓	onlyOwner
	recoverTokens	External	✓	onlyOwner
	_getUnlockedAmount	Private		
	_getNextUnlock	Private		
	getUnlockedMarketingAmount	Public		-
	getUnlockedTeamAmount	Public		-
	getNextMarketingUnlock	Public		-
	getNextTeamUnlock	Public		-
	teamUnlockIn	External		-
	_getTokenUnlockInfo	Private		
	getUnlockStatus	External		-
	_transferTokens	Internal	✓	

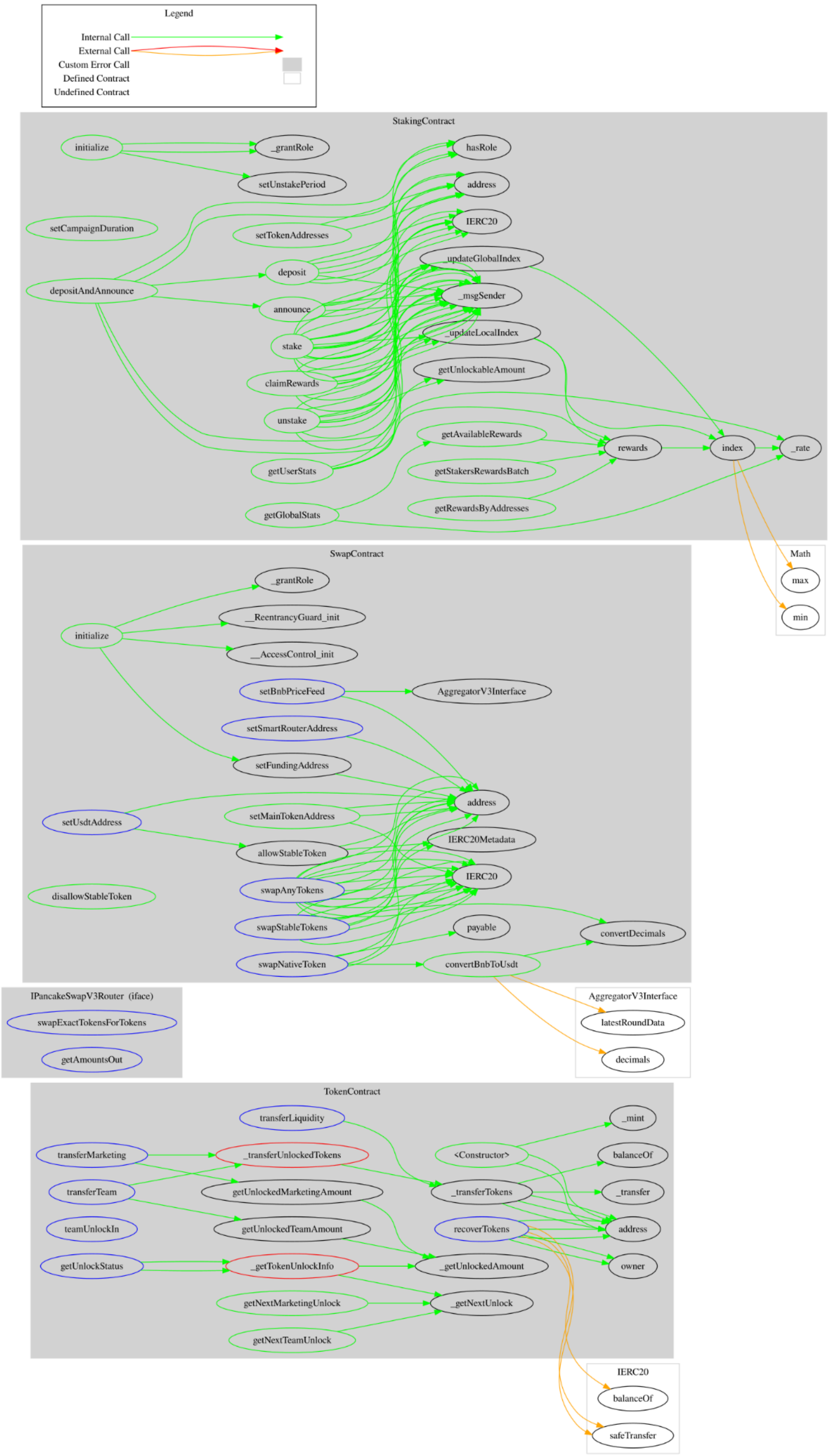
SwapContract	Implementation	Initializable, AccessControlUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	setUsdtAddress	External	✓	onlyRole
	setBnbPriceFeed	External	✓	onlyRole
	setSmartRouterAddress	External	✓	onlyRole
	setMainTokenAddress	Public	✓	onlyRole
	setFundingAddress	Public	✓	onlyRole
	allowStableToken	Public	✓	onlyRole
	disallowStableToken	Public	✓	onlyRole
	convertDecimals	Public		-
	convertBnbToUsdt	Public		-
	swapNativeToken	External	Payable	nonReentrant
	swapStableTokens	External	✓	nonReentrant
	swapAnyTokens	External	✓	nonReentrant
StakingContract	Implementation	Initializable, AccessControlUpgradeable		
	initialize	Public	✓	initializer
	setCampaignDuration	Public	✓	onlyRole
	setTokenAddresses	Public	✓	onlyRole
	setUnstakePeriod	Public	✓	onlyRole
	deposit	Public	✓	-

	announce	Public	✓	-
	depositAndAnnounce	Public	✓	-
	getAvailableRewards	Public		-
	stake	Public	✓	-
	unstake	Public	✓	-
	getUnlockableAmount	Public		-
	claimRewards	Public	✓	-
	_updateGlobalIndex	Private	✓	
	_updateLocalIndex	Private	✓	
	_rate	Private		
	index	Public		-
	rewards	Public		-
	getUserStats	Public		-
	getGlobalStats	Public		-
	getStakersRewardsBatch	Public		-
	getRewardsByAddresses	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
StakingContract
initialize
_grantRole
hasRole
setUnstakePeriod
address
setCampaignDuration
setTokenAddresses
IERC20
_updateGlobalIndex
deposit
depositAndAnnounce
_msgSender
announce
_updateLocalIndex
stake
getUnlockableAmount
claimRewards
unstake
getAvailableRewards
rewards
index
_rate
getUserStats
getStakersRewardsBatch
getGlobalStats
getRewardsByAddresses
Math
max
min
SwapContract
_grantRole
__ReentrancyGuard_init
initialize
__AccessControl_init
setBnbPriceFeed
AggregatorV3Interface
setSmartRouterAddress
setFundingAddress
address
setUsdtAddress
setMainTokenAddress
IERC20Metadata
allowStableToken
disallowStableToken
swapAnyTokens
IERC20
swapStableTokens
payable
convertDecimals
swapNativeToken
convertBnbToUsdt
IPancakeSwapV3Router (iface)
swapExactTokensForTokens
getAmountsOut
AggregatorV3Interface
latestRoundData
decimals
TokenContract
transferLiquidity
_mint
transferMarketing
_transferUnlockedTokens
<Constructor>
balanceOf
transferTeam
getUnlockedMarketingAmount
_transferTokens
_transfer
teamUnlockIn
getUnlockedTeamAmount
recoverTokens
address
getUnlockStatus
_getTokenUnlockInfo
_getUnlockedAmount
owner
getNextMarketingUnlock
_getNextUnlock
getNextTeamUnlock
IERC20
balanceOf
safeTransfer

Summary

BOBE contract implements a token, staking and exchange mechanism. This audit investigates security issues, business logic concerns and potential improvements. The smart contract analysis reported no compiler errors or critical issues.

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

cyberscope.io