Cyberscope

Audit Report

Hodl

December 2024

SHA256

0a541a90ed95f5d55b613f37b0cb4f7fb7a8e8f0189acd86d81cc73a749a28b4

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	MMRR	Missing Maximum Reward Restriction	Acknowledged
	IMRA	Inconsistent Maximum Reward Amount	Unresolved
	DRA	Dynamic Reward Allocation	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L13	Divide before Multiply Operation	Unresolved

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

Audit Updates

Initial Audit	29 Nov 2024 https://github.com/cyberscope-io/audits/blob/main/hodl/v1/audit.pdf
Corrected Phase 2	05 Dec 2024 https://github.com/cyberscope-io/audits/blob/main/hodl/v2audit.pdf
Corrected Phase 3	13 Dec 2024
Test Deploy	https://sepolia.basescan.org/address/0x065d47707b09D82c6dF4ffb535dea59624B35387

Source Files

Filename	SHA256
OwnableUpgradeable.sol	ebf38dc17b401ac3a98de2db8c31e184b52dac2c7e8ac5e53884298a8f815a0c
HODL_upgrade.sol	0a541a90ed95f5d55b613f37b0cb4f7fb7a8e8f0189acd86d81cc73a749a28b4
HODLTypes.sol	eef155255804841da50c9ab9f30e0335b987a8ae58cd99e02c60e3b0893f59e7

Overview

The Hodl contract implements a token mechanism with staking and reward distribution functionalities. Users can claim rewards collected by the contract in the form of fees. Stakers can also claim from the reward pool by compounding their share. Additionally, users may choose to reinvest their rewards by swapping them for Hodl tokens. Main functionalities include:

startStacking Function:

This function enables staking for a user and stakes all of their token balance except for 1 token. The contract checks that the user does not have an active staking session and that the balance exceeds a minimum threshold. If that is true, the user's balance is transferred to the `STACKING_ADDRESS` and the staking information is stored in a structure including the staked amount, the timestamp at the start of the staking and the claim period.

redeemRewards Function:

The contract allows a holder of the token to claim a percentage of the reward pool based on their token balance . Specifically, if the current balance of the contract exceeds a threshold, the user's rewards are calculated as a percentage of that threshold, proportional to their balance as a percentage of the circulating supply. Otherwise, if the contract's balance is less than the threshold, the former is used. It is important to notice that the user's balance is used for the calculation of rewards, rather than the deposited balance in the `STACKING_ADDRESS`. The `redeemRewards` function can be therefore called by external addresses that do not necessarily stake the token.

stopStackingAndClaim Function:

This function allows users to terminate their staking and claim their rewards. Rewards are calculated through a call to the `getStacked` function. Here the contract considers several cases.

If the contract holds more tokens than the cap set for the claimable rewards, the latter is used in the calculations. In this case, rewards are estimated as a portion of the cap proportionally to the staked balance of the user as a percentage of the circulating supply, multiplied by the integer of periods the user has been staking.

$$rewards = rewardCap * \frac{userStaked}{circulatingSupply} * stakedPeriods$$

In this calculation, it is possible that the rewards exceed the balance of the contract.

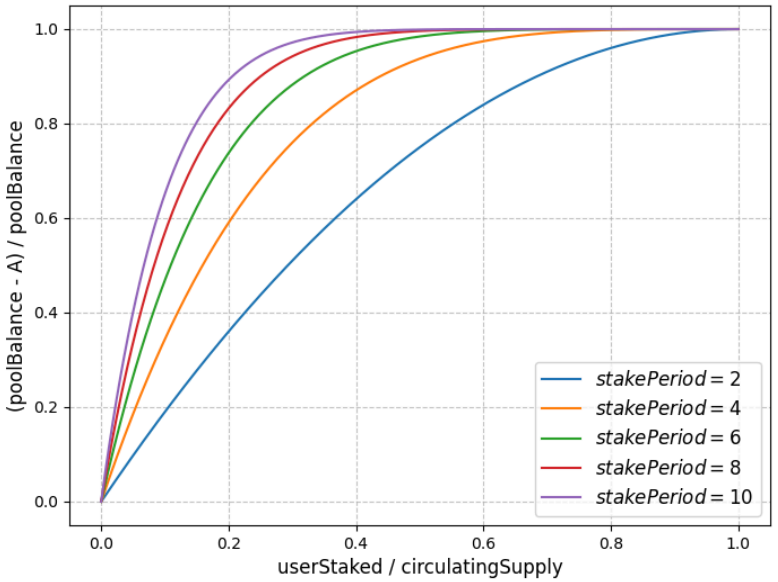
In this case, the rewards are re-evaluated by an interesting but rather complicated distribution mechanism which is inspired by the compound interest methodology. Specifically, the contract utilizes a series approximation to calculate the equation:

$$A = poolBalance * (1 - userStaked/circulatingSupply)^{stakedPeriods}$$

This calculation returns the portion of the pool balance that is inaccessible to the user. To estimate the portion that the user may claim, A is subtracted from the pool balance:

$$reward = poolBalance - A$$

The latter estimates the claimable rewards as a function of the staked amount, the staking period and the pool size. In the following graph, the available rewards as a portion of the pool size are plotted for different magnitudes of staked amounts and staked durations:

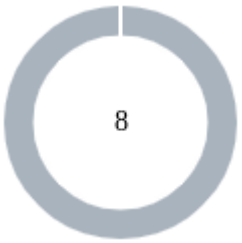


In this figure, a larger portion of the pool becomes available, on the y-axis, as the staked amount or the staking period increases. Specifically, for a constant staked amount, more rewards are unlocked as time passes, while for the same staking period, larger staked amounts yield larger rewards.

The graph reveals that multiple configurations allow the estimated rewards to converge to the total pool becoming available. Users therefore, need to elect the proper strategy to maximize their profits.

Furthermore, these rewards are adjusted by a percentage of the pool amount that remains inaccessible proportionally to the user’s staked balance and the elapsed time from the current period. At the end of these calculations a hard threshold is applied on the estimated rewards ensuring that they cannot exceed a predefined limit, set at the start of the staking.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	7	1	0	0

ST - Stops Transactions

Criticality	Minor / Informative
Location	HODL_upgrade.sol#L239,505
Status	Unresolved

Description

The contract owner can set a maximum daily maximum sell limit of 0.25% of the total supply. Consequently, users will be restricted from selling more than this amount within a single day.

```
function changeMaxSellAmount(
uint256 newValue
) external onlyOwner onlyPermitted {
if (
newValue < (super.totalSupply() * 25) / 10_000 ||
newValue > (super.totalSupply() * 500) / 10_000
) revert ValueOutOfRange();
uint256 oldValue = maxSellAmount;
maxSellAmount = newValue;
emit ChangeValue(oldValue, newValue, "maxSellAmount");
}
```

```
// Ensures daily sell limit is enforced for each user
function ensureMaxSellAmount(address from, uint256 amount) private
{
WalletAllowance storage wallet = userWalletAllowance[from];

// Reset daily sell allowance if 24 hours have passed since last
transaction
if (block.timestamp > wallet.lastTransactionTimestamp + 1 days) {
wallet.lastTransactionTimestamp = 0;
wallet.dailySellVolume = 0;
}

uint256 totalAmount = wallet.dailySellVolume + amount;
if (totalAmount > maxSellAmount) revert ExceededDailySellLimit();

// Update daily allowance tracking
if (wallet.lastTransactionTimestamp == 0) {
wallet.lastTransactionTimestamp = block.timestamp;
}
wallet.dailySellVolume = totalAmount;
}
```

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 governance model where users will vote about the actions.

Permanent Solution:

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

DRA - Dynamic Reward Allocation

Criticality	Minor / Informative
Location	HODL_upgrade.sol#L375
Status	Unresolved

Description

The contract distributes rewards from a pool of accumulated fees. These rewards are finite and can be claimed at any time by any eligible address. At the time of the claim, rewards are calculated based on the available pool balance or a cap value. In the first case, race conditions and optimization points are induced where users are incentivized to claim rewards before the pool is depleted. This may disincentivize users from compounding their rewards through longer deposits to the `STACKING_ADDRESS`. While it could also result in unexpected redeemable rewards as the pool balance changes over time.

```
if (initialBalance >= tmpStack.rewardPoolCapAtStart) {
    reward =(((uint256(tmpStack.rewardPoolCapAtStart) *
    tmpStack.stackedAmount) / currentRewardPoolShare)
*stackedTotal) / 1E6;
if (
    reward >= initialBalance ||
    initialBalance - reward < tmpStack.rewardPoolCapAtStart
) {
    reward = _calculateStackedReward(
    initialBalance,
    tmpStack,
    stacked,
    rest,
    currentRewardPoolShare);
    }
}
else {
    reward = _calculateStackedReward(
    initialBalance,
    tmpStack,
    stacked,
    rest,
    currentRewardPoolShare
);
}
```

Recommendation

Implementing an economic design that ensures proportional distribution according to the staked balance and period, while guaranteeing that the expected amount accumulates independently of changes in the reward balance, will enhance consistency and user trust in the system. The team is advised to implement a mechanism that updates the user's rewards at the time the balance is updated.

IMRA - Inconsistent Maximum Reward Amount

Criticality	Minor / Informative
Location	HODL.sol#L178
Status	Unresolved

Description

The contract implements the `redeemRewards` function, which allows the caller to redeem rewards from the contract's reward reserves. In this calculation, the user's amount is estimated based on the caller's current balance, not the actual staked amount. Rewards are calculated proportionally to the balance as a function of the circulating supply. This approach disincentivizes users from staking in the contract, as rewards can be claimed simply by holding tokens. Additionally, it allows users to transfer funds between their own accounts to exploit favorable staking claim dates without triggering the calculation of a `newCycleBlock` in the new wallet.

```
function redeemRewards(uint8 perc) external nonReentrant {
    if (perc > 100) revert ValueOutOfRange();
    uint256 userBalance = super.balanceOf(msg.sender);
    if (nextClaimDate[msg.sender] > block.timestamp)
    revert ClaimPeriodNotReached();
    if (userBalance == 0) revert NoHODLInWallet();
    uint256 currentBNBPool = address(this).balance;
    uint256 reward = currentBNBPool > bnbRewardPoolCap
    ? (bnbRewardPoolCap * userBalance) / rewardPoolShare
    : (currentBNBPool * userBalance) / rewardPoolShare;
    executeRedeemRewards(perc, reward);
}
```

Recommendation

It is advisable to ensure that rewards are claimable only by staked accounts to maintain consistency and fairness in the reward distribution mechanism. Additionally, ensuring the proper application of claiming restrictions across all addresses will prevent potential manipulation of the reward pool.

MMRR - Missing Maximum Reward Restriction

Criticality	Minor / Informative
Location	HODL_upgrade.sol#L178
Status	Acknowledged

Description

The contract implements two methods for withdrawing rewards from the pool's reserves: by calling the `redeemRewards` function or by using the `stopStackingAndClaim` function to withdraw staked amounts and claim available rewards. In the latter case, a maximum reward amount is always applied, equal to a `rewardLimit` initialized at the start of staking. However, in the first case, this limit is not imposed, allowing users to claim rewards exceeding the limit.

```
function redeemRewards(uint8 perc) external nonReentrant {
    if (perc > 100) revert ValueOutOfRange();
    uint256 userBalance = super.balanceOf(msg.sender);
    if (nextClaimDate[msg.sender] > block.timestamp)
    revert ClaimPeriodNotReached();
    if (userBalance == 0) revert NoHODLInWallet();
    uint256 currentBNBPool = address(this).balance;
    uint256 reward = currentBNBPool > bnbRewardPoolCap
    ? (bnbRewardPoolCap * userBalance) / rewardPoolShare
    : (currentBNBPool * userBalance) / rewardPoolShare;
    executeRedeemRewards(perc, reward);
}
```

Recommendation

The team is advised to ensure consistency between the different claiming mechanisms. Calculating rewards using the same formula and restrictions will ensure the smooth operation of the rewards pool.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	HODL_upgrade.sol#L653
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapTokensForEth(uint256 tokenAmount) private {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = PANCAKE_ROUTER.WETH();

PANCAKE_ROUTER.swapExactTokensForETHSupportingFeeOnTransferTokens(
    tokenAmount,
    0,
    path,
    address(this),
    block.timestamp
    );
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	HODL_upgrade.sol#L77,88,89,96,97
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 private previousTokenBalance
uint256 public bnbStackingLimit
uint256 public minTokensToStack
bool public rewardSwapEnabled
bool public stackingEnabled
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	HODL_upgrade.sol#L318,327
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.

```
bool _enable
```

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/stable/style-guide.html#naming-conventions.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	HODL_upgrade.sol#L384,385,392,439,671,675,698,718,723,746,751
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 calcExponent = (exponent * (exponent - 1)) / 2
reward -=
(coefficient * calcExponent) /
calcFactorOne /
calcFactorTwo /
calcFactorThree
```

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**
OwnableUpgradeable	Implementation	Initializable, ContextUpgradeable		
	_getOwnableStorage	Private		
	__Ownable_init	Internal	✓	onlyInitializing
	__Ownable_init_unchained	Internal	✓	onlyInitializing
	owner	Public		-
	owner2	Public		-
	owner3	Public		-
	permittedBy	Public		-
	permittedTo	Public		-
	permittedAt	Public		-
	_isOwner	Internal		
	_checkOwner	Internal		
	_checkPermission	Internal		
	_cancelPermission	Internal	✓	
	givePermission	External	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner onlyPermitted
	transferOwner2	Public	✓	onlyOwner onlyPermitted
	transferOwner3	Public	✓	onlyOwner onlyPermitted
	_transferOwnership	Internal	✓	

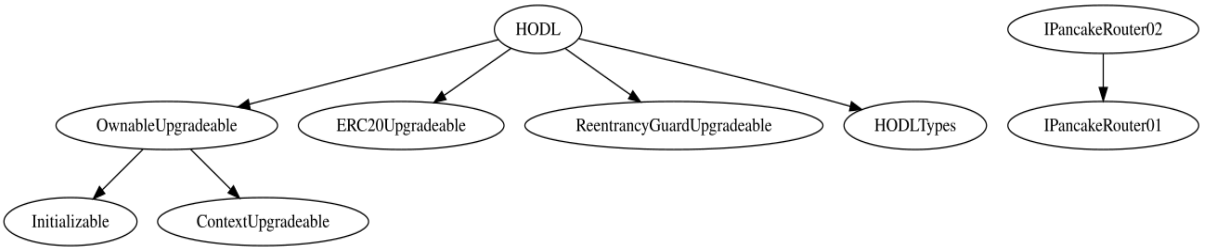
	_transferOwner2	Internal	✓	
	_transferOwner3	Internal	✓	
IPancakeRouter02	Interface	IPancakeRouter01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IPancakeRouter01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-

	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
HODL	Implementation	ERC20Upgradeable, OwnableUpgradeable, ReentrancyGuardUpgradeable, HODLTypes		
		External	Payable	-
	upgrade	External	✓	onlyOwner reinitializer
	stopStackingAndClaim	External	✓	nonReentrant
	startStacking	External	✓	-
	redeemRewards	External	✓	nonReentrant
	updateIsTaxFree	External	✓	onlyOwner onlyPermitted
	excludeFromRewardPoolShare	External	✓	onlyOwner onlyPermitted
	changeBuyTaxes	External	✓	onlyOwner onlyPermitted
	changeSellTaxes	External	✓	onlyOwner onlyPermitted
	changeMaxSellAmount	External	✓	onlyOwner onlyPermitted
	changeMinTokensTriggerRewardSwap	External	✓	onlyOwner onlyPermitted

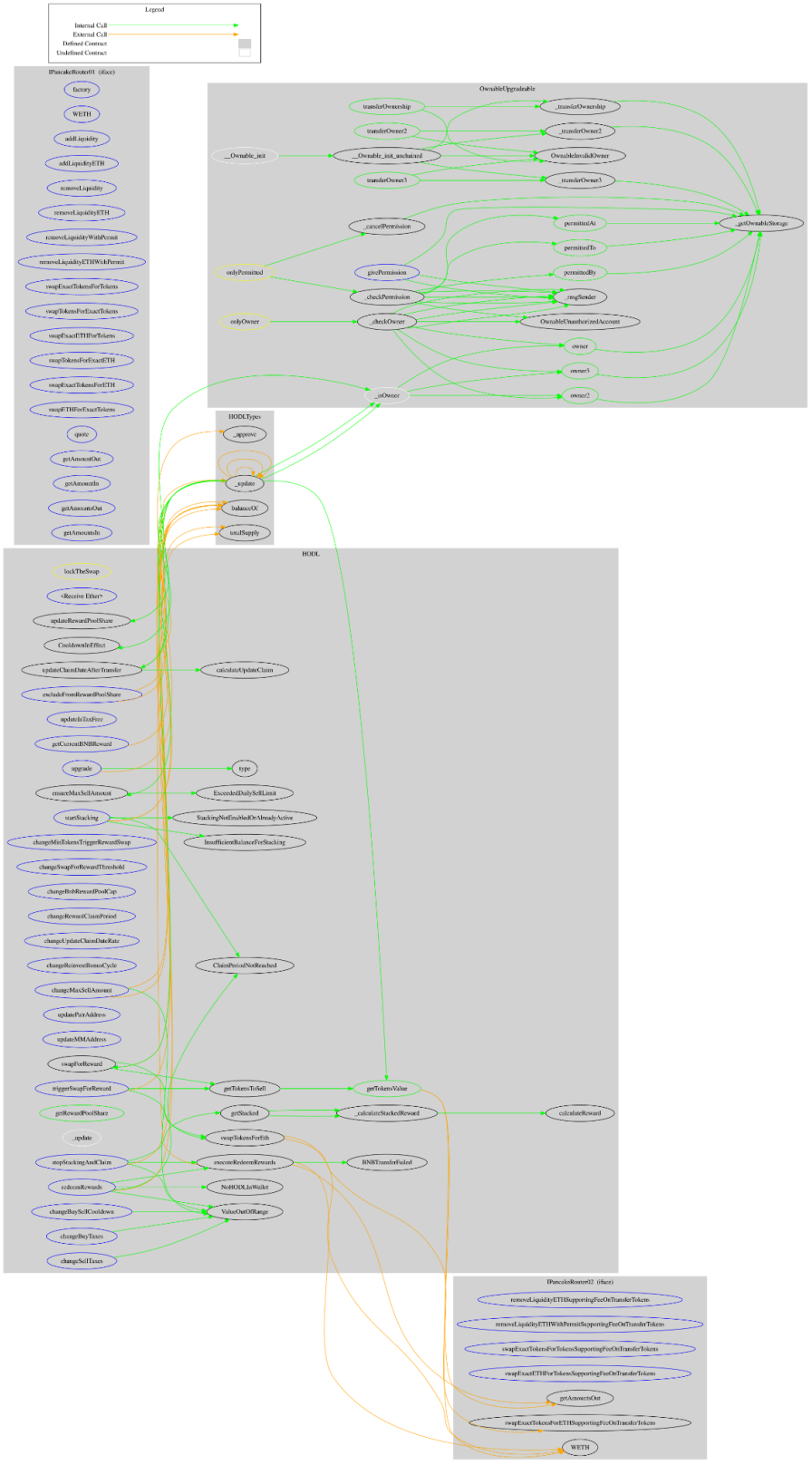
	changeSwapForRewardThreshold	External	✓	onlyOwner onlyPermitted
	changeBnbRewardPoolCap	External	✓	onlyOwner onlyPermitted
	changeRewardClaimPeriod	External	✓	onlyOwner onlyPermitted
	changeUpdateClaimDateRate	External	✓	onlyOwner onlyPermitted
	changeReinvestBonusCycle	External	✓	onlyOwner onlyPermitted
	changeBuySellCooldown	External	✓	onlyOwner onlyPermitted
	updatePairAddress	External	✓	onlyOwner onlyPermitted
	updateMMAddress	External	✓	onlyOwner onlyPermitted
	triggerSwapForReward	External	✓	lockTheSwap onlyPermitted
	getCurrentBNBReward	External		-
	getRewardPoolShare	Public		-
	getTokensValue	Public		-
	getStacked	Public		-
	_calculateStackedReward	Internal		
	_update	Internal	✓	
	updateRewardPoolShare	Private	✓	
	ensureMaxSellAmount	Private	✓	
	executeRedeemRewards	Private	✓	
	updateClaimDateAfterTransfer	Private	✓	
	swapForReward	Private	✓	lockTheSwap
	getTokensToSell	Private	✓	
	swapTokensForEth	Private	✓	

	calculateUpdateClaim	Private		
	calculateReward	Private		
HODLTypes	Implementation			

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IPancakeRouter01 (iface)
factory
WETH
addLiquidity
addLiquidityETH
removeLiquidity
removeLiquidityETH
removeLiquidityWithPermit
removeLiquidityETHWithPermit
swapExactTokensForTokens
swapTokensForExactTokens
swapExactETHForTokens
swapTokensForExactETH
swapExactTokensForETH
swapETHForExactTokens
quote
getAmountOut
getAmountIn
getAmountsOut
getAmountsIn
OwnableUpgradeable
transferOwnership
transferOwner2
__Ownable_init
__Ownable_init_unchained
transferOwner3
_transferOwnership
_transferOwner2
OwnableInvalidOwner
_transferOwner3
cancelPermission
onlyPermitted
givePermission
_checkPermission
onlyOwner
_checkOwner
permittedAt
permittedTo
permittedBy
_msgSender
OwnableUnauthorizedAccount
owner
owner3
owner2
_getOwnableStorage
_isOwner
HODLTypes
_approve
_update
balanceOf
totalSupply
HODL
lockTheSwap
<Receive Ether>
updateRewardPoolShare
CooldownInEffect
updateClaimDateAfterTransfer
excludeFromRewardPoolShare
updateIsTaxFree
getCurrentBNBReward
upgrade
type
ensureMaxSellAmount
ExceededDailySellLimit
startStacking
StackingNotEnabledOrAlreadyActive
changeMinTokensTriggerRewardSwap
InsufficientBalanceForStacking
changeSwapForRewardThreshold
changeBnbRewardPoolCap
changeRewardClaimPeriod
changeUpdateClaimDateRate
changeReinvestBonusCycle
ClaimPeriodNotReached
changeMaxSellAmount
updatePairAddress
updateMMAddress
swapForReward
triggerSwapForReward
getTokensToSell
getTokensValue
getRewardPoolShare
getStacked
_calculateStackedReward
calculateReward
_update
swapTokensForEth
stopStackingAndClaim
executeRedeemRewards
BNBTransferFailed
redeemRewards
NotHODLInWallet
changeBuySellCooldown
ValueOutOfRange
changeBuyTaxes
changeSellTaxes
IPancakeRouter02 (iface)
removeLiquidityETHSupportingFeeOnTransferTokens
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
getAmountsOut
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH

Summary

HODL contract implements a token and staking mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. Renouncing ownership will eliminate all the contract threats.

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