STAYSAFU
AUDIT

August 3rd, 2022

HODL

TABLE OF CONTENTS

AUDIT SUMMARY

This report was written for HODL ($HODL) in order to find flaws and vulnerabilities in the HODL project's source code, as well as any contract dependencies that weren't part of an officially recognized library.

A comprehensive examination has been performed, utilizing Static Analysis, Manual Review, and HODL Deployment techniques. The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors
- Assessing the codebase to ensure compliance with current best practices and industry standards
- Ensuring contract logic meets the specifications and intentions of the client
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Through line-by-line manual review of the entire codebase by industry expert

AUDIT OVERVIEW

PROJECT SUMMARY

Project name	HODL
Description	HODL is an BEP20 token which operates on the Binance Smart Chain (BSC) Network as a dual-reward token offering both BNB rewards and reflections to its investors (HODLers), along with staking, farming, NFTs, and its own DEX.
Platform	Binance Smart Chain
Language	Solidity
Codebase	https://bscscan.com/address/0x1c8c15ad376fe44ae487268abe780ca9d00ca132#code *Note: This is the implementation contract address of the HODL Token that was initially audited https://bscscan.com/address/0xFf5B297140522394b1578b04Da13A9E797cCeF2E#code *Note: Line numbers within the report are based on this contract. MD5 Hash | File Name 54b53976c23a4b043b496bc33b98b823 HODLv3.sol https://bscscan.com/address/0xb6875f2e79d8e8d6cf768c0a045788e5b3070ca4#code *Note, This is the current implementation contract address of the HODL Token. The 12 new functions introduced in this version were also audited.

FINDINGS SUMMARY

Vulnerability	Total	Resolved
Critical	0	0
Major	0	0
Medium	3	1
Minor	5	3
Informational	8	4

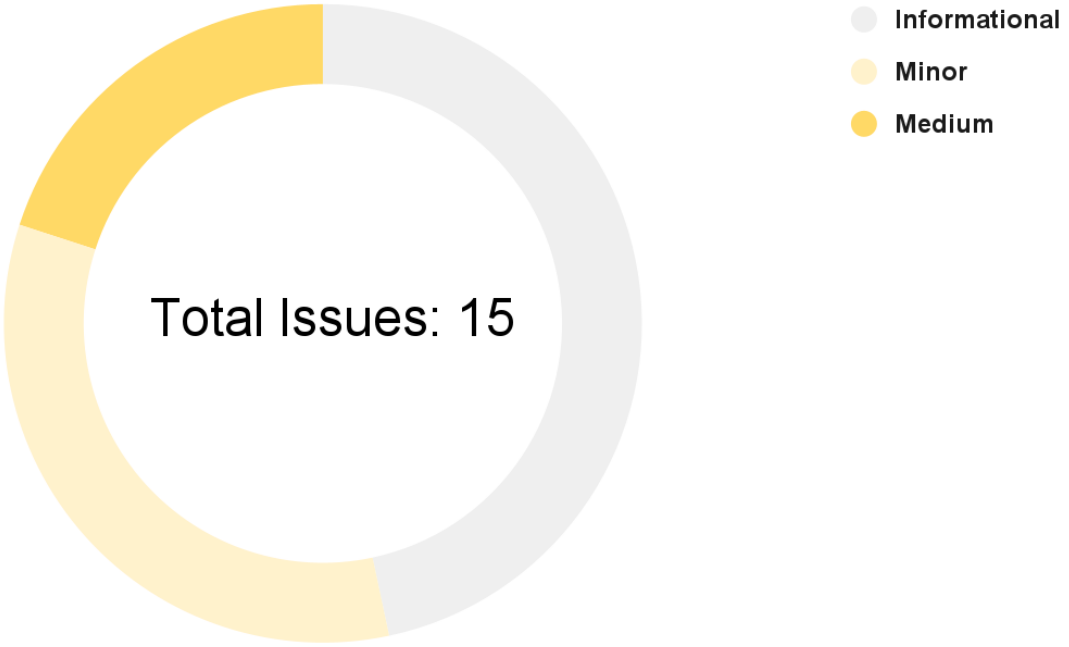
EXECUTIVE SUMMARY

HODL is a DeFi project built on the Binance Smart Chain (BSC). HODL is revolutionary and constantly innovating to drive more rewards and value to all holders. It was the first project to reward its holders with BNB and reflections just for holding and has set the record for the biggest payouts of all time. (HODL, n.d.)

At the heart of HODL is a highly-innovative smart contract which captures tax revenues from buys, sells and transfers of the token. Our sell bot liquifies these tokens converting them into BNB and then places the funds into the reward pool. By holding HODL you can collect your share of the reward pool every 7-days and will be sent reflections throughout. (HODL, n.d.)

There have been no major or critical issues related to the codebase and all findings listed here are minor or informational. The medium security issues are the dependence on a decentralized exchange platform, centralization of privileges, and missing threshold checks. The missing thresholds check was resolved.

AUDIT FINDINGS



Code	Title	Severity
CENT-1	Centralization of major privileges	Medium
EXT-1	External protocol dependencies	Medium
THRE-1	Missing threshold checks	Medium
THRE-2	Missing zero address checks	Minor
MSG-1	Missing event emits	Minor
BLOC-1	Use of block.timestamp	Minor
BLOC-2	Use of block.number	Minor
COMP-1	Unfixed version of compiler	Minor

GAS-1	Unoptimized function type ●	Informational
BP-1	Function naming convention ●	Informational
BP-2	Use of enum values in place of IDs	Informational
BP-3	Undescriptive variable names ●	Informational
MSG-2	Limited NatSpec comments	Informational
FUNC-1	Unused functions	Informational
FUNC-2	Redundant function ●	Informational

Legend:

● -> Resolved

CENT-1 | Centralization of major privileges

Description

The onlyOwner modifier in the smart contract(s) give major privileges over them (including/excluding addresses from rewards, updating fees)*. This can be a problem, in the case of a hack, an attacker who has taken possession of this privileged account could damage the project and the investors.

*This list is not exhaustive but presents the most sensitive points

Recommendation

We recommend at least to use a multi-sig wallet as the owner address, and at best to establish a community governance protocol to avoid such centralization.

See: https://solidity-by-example.org /app/multi-sig-wallet/

EXT-1 | Dependence to an external protocol

Description

The contract serves as an underlying entity to interact with third party Uniswap/PancakeSwap protocols. The scope of the audit would treat this third party entity as black box and assume it is fully functional. However in the real world, third parties may be compromised and may have led to assets lost or stolen.

Recommendation

We encourage the team to constantly monitor the security level of the entire Uniswap/PancakeSwap project, as the security of the token is highly dependent on the security of the decentralized exchange platform.

THRE-1 | Missing threshold checks | Resolved

Description

Functions which can change sensitive variables within HODL's contract do not contain threshold checks to ensure these variables are not changed to unreasonable values. This includes fees and max tx amount. As such it is important to add a threshold to prevent an attacker from setting max transaction amount as 0 or fees as 100% easily. Key examples of Identified functions with this issue have been listed below:

- setTaxFeePercent -> Line 1282
- setLiquidityFeePercent -> Line 1286
- setMaxTxPercent* -> Line 1681
- changeSelltax* -> Line 1977
- changeBuytax* -> Line 1982
- changeTransfertax* -> Line 1986
- Changebnbclaimtax > Line 1996
- changerewardHardcap -> Line 2016
- changeclaimBNBLimit -> Line 2037
- changereinvestLimit > Line 2041
- changeBNBstackingLimit -> Line 2061

Recommendation

We recommend adding threshold checks using require statements for each of the identified functions above and other functions with this issue.

THRE-2 | Missing zero address checks | Resolved

Description

Some functions which change sensitive addresses are missing checks to prevent them from being changed to the zero address. Functions found with this issue are listed below:

- initOwner -> Line: 298
- changereservewallet -> Line: 1941
- changemarketingwallet -> Line: 1945
- changetriggerwallet -> Line: 1949
- migrateBnb -> Line: 1968
- changeHODLMasterChef -> Line: 2045
- changeStackingWallet -> Line: 2049

OpenZeppelin has standardized the use of zero address checks in functions which change sensitive address variables. This is to prevent accidental loss of privileged access and/or functionality within a project.

Recommendation

We recommend changing these functions to include a zero address check. An example can be seen in the transferOwnership function in Ownable.sol.

Source: openzeppelin-contracts/Ownable.sol at master

MSG-1 | Missing event emits | Resolved

Description

Some functions within HODL's contracts modify sensitive variables without emitting an event. This issue includes functions which modify fees such as setTaxFeePercent and setLiquidityFeePercent. Examples of functions with this issue are listed below:

- setTaxFeePercent -> Line 1282
- setLiquidityFeePercent -> Line 1286
- setMaxTxPercent -> Line 1681
- changerewardCycleBlock -> Line 1937
- changeThreshHoldTopUpRate -> Line 1973
- changeSelltax -> Line 1977
- changeBuytax -> Line 1982
- changeTransfertax -> Line 1986
- changeminTokenNumberToSell -> Line 2006
- changeminTokenNumberUpperlimit -> Line 2011
- changerewardHardcap -> Line 2016

Recommendation

We recommend amending these functions to include event emits to ensure transparency with users.

BLOC-1 | Use of block.timestamp

Description

The use of block.timestamp can be problematic. The timestamp can be partially manipulated by the miner (see https://cryptomarketpool.com/block-timestamp-manipulation-attack/).

Recommendation

We fully understand that the use of block.timestamp within the HODL Protocol is required for certain functionality such as the redemption of rewards. Nevertheless, it is still useful to point out this kind of potential security problem.

BLOC-2 | Use of block.number | Resolved

Description

The use of block.number can be problematic. The timestamp can be partially manipulated by the miner (see https://cryptomarketpool.com/block-timestamp-manipulation-attack/). Since the timestamp of a block cannot be fully trusted, the exact block counting at an exact timestamp cannot be fully trusted.

Recommendation

We fully understand that the use of block.number within the HODL Protocol is required for the functionality of the random function (line 680). Nevertheless, it is still useful to point out this kind of potential security problem.

COMP-1 | Unfixed version of compiler

Description

HODL token's contract does not have locked compiler versions, meaning a range of compiler versions can be used. This can lead to differing bytecodes being produced depending on the compiler version, which can create confusion when debugging as bugs may be specific to a specific compiler version(s).

Recommendation

To rectify this, we recommend setting the compiler to a single version, the lowest version tested to be compatible with the code, an example of this change can be seen below.

```
pragma solidity 0.6.8;
```

GAS-1 | Unoptimized function type | Resolved

Description

Throughout HODL's contracts some functions are of type public although they are never called within the contract. External functions require significantly less gas to call. Examples of such found functions are listed below:

- excludeFromReward -> Line 1229
- excludeFromFee -> Line 1274
- includeInFee -> Line 1278
- setSwapAndLiquifyEnabled -> Line 1290
- isExcludedFromFee -> Line 1429
- setMaxTxPercent -> Line 1681
- setExcludeFromMaxTx -> Line 1686
- redeemRewards -> Line 1711
- triggerSwapAndLiquify -> Line 1897

Recommendation

We recommend reviewing each of the functions listed above and others like them within the contract, and where possible switch their type from public to external.

BP-1 | Function naming convention | Resolved

Description

Some names of functions within HODL's contract don't follow best practices. This hinders the contracts readability, especially for non english speakers. Examples of such functions have been listed below.

- changereservewallet -> Line: 1941
- changemarketingwallet -> Line: 1945
- changetriggerwallet -> Line: 1949
- reflectionfeestartstop -> Line: 1954
- changereinvestLimit -> Line 2041

Recommendation

We recommend changing the name of these functions to use camel case as is standard.
Source: https://docs.soliditylang.org/en/v0.8.14/style-guide.html

BP-2 | Use of enum values in place of IDs

Description

Throughout HODL'S contracts, fixed arrays are used to store distinct variables. An example of such an array is path (swapTokensForEth, line 739), which is an address array of size 2. This array is only used to store address(this), and router address respectively such that path[0] would return the address(this) address. It is industry standard to use enum values for such arrays of fixed size and use. This improvement helps in maintainability. Other functions with this issue are listed below:

- swapETHForTokens -> Line 759
- swapTokensForTokens -> Line 782
- getAmountsout -> Line 804

Recommendation

We recommend In place of numerical array index IDs, (0, 1 in the example above), to declare an enum struct and use these values instead. As can be seen in the example below, this change can make development and code review clearer.

```
//Example enum declaration
enum PATH_ID { L_ADDRS, P_ADDRS }

//Example enum use
require(path[PATH_ID.L_ADDRS] != address(0), "Zero address")
```

BP-3 | Undescriptive variable names | Resolved

Description

The names of variables with the function calcReward (line 891) are non descriptive primarily consisting of single character names. This issue goes against naming conventions and makes the code difficult to understand thus reducing maintainability.

Recommendation

We recommend modifying variable names to be more descriptive to their purpose. Single character variables should only be used as an index within loops.
See: https://docs.soliditylang.org/en/v0.8.14/style-guide.html

MSG-2 | Limited NatSpec comments

Description

Throughout HODL's contracts many functions remain uncommented. This can make understanding the code's functionality difficult for developers and users (if the code is open source) thus reducing maintainability.

Recommendation

We recommend using NetSpec standard comments throughout all of HODL's contracts.

See:

https://docs.soliditylang.org/en/v0.5.17/style-guide.html?highlight=natspec%23natspec

FUNC-1 | Unused functions

Description

Multiple functions within HODL's contract are defined as private or internal but are never called within the contract. This wastes contract space as there is a maximum size a contract can have. Functions found with this issue have been listed below:

- _msgData -> Line 140
- sendValue -> Line 162
- functionCall -> Line 176
- functionCallWithValue -> Line 191
- random -> Line 680*
- quote -> Line 974

Recommendation

We recommend removing these functions from the contract.

*The "random" function has been removed in the latest version of the contract.

FUNC-2 | Redundant function | Resolved

Description

The function initOwner (line 298) isn't part of the official OpenZeppelin Ownable.sol contract. This function doesn't provide any functionality to the contract as transferOwnership (line 317) can be used in its place.

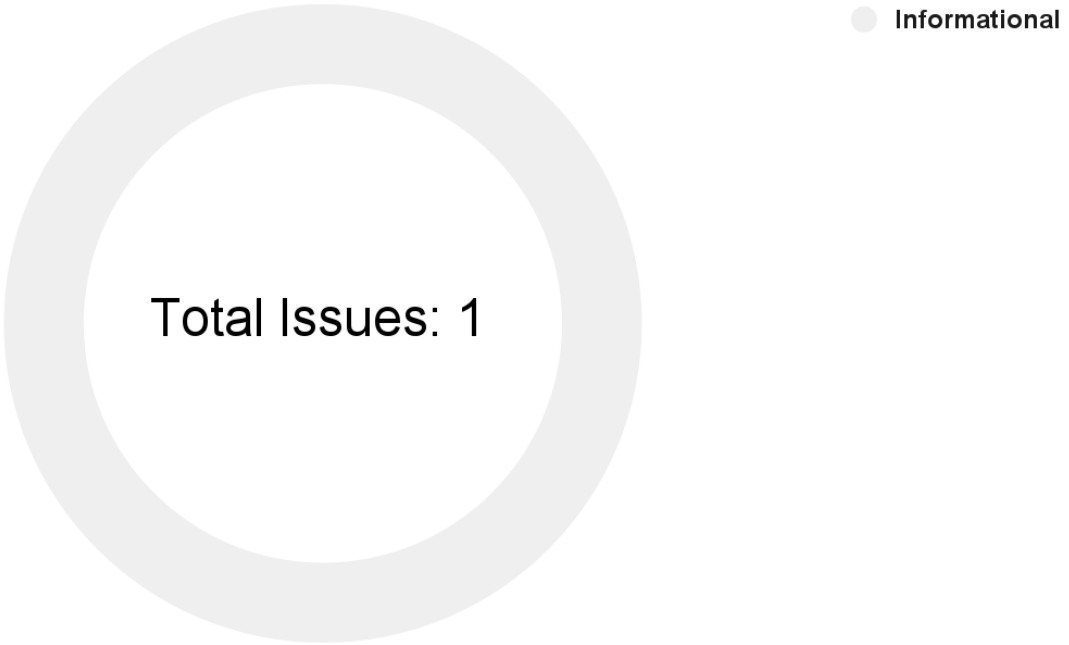
Recommendation

We recommend removing this redundant function from the contract and use transferOwnership in its place.

AUDIT FINDINGS - NEW FUNCTIONS

The scope of these issues are limited to the 12 new functions introduced in the latest version of the contract. The list of these functions can be seen below.

- rawFulfillRandomWords
- verifyCallResultFromTarget
- includeExcludeFromFee
- doSwapAndLiquify
- changeAnyValue
- changeAnyAddress
- enableDisableAnyFunction
- fulfillRandomWords
- burnTokens
- getAllTickets
- requestRandomWords
- evaluatePendingLottery



Code	Title	Severity
BP-1	Grouped function	Informational

BP-1 | Grouped function

Description

Functions within HODL's contract perform multiple distinct functionalities and what operation is performed is dependent on an uint256 parameter and a series of if statements. This can be an issue as having such a single function to control fee changes or address changes can lead to the wrong fee, or address variable being changed if a developer inputs the incorrect uint parameter. Functions which have this issue are listed below.

- changeAnyValue | line -> 2518
- changeAnyAddress | line -> 2589
- enableDisableAnyFunction | line -> 2612

Recommendation

We understand that having a single function to control groups of state variable changes can reduce contract size. However, we recommend having a system that is less prone to error such as comparing strings (through keccak256) instead of using uint to decide what operation is performed.

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement.

This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without StaySAFU'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 considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts StaySAFU to perform a security assessment.

This report 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 in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

StaySAFU's position is that each company and individual are responsible for their own due diligence and continuous security. StaySAFU'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 fun.