HodlessBotToken Security Analysis

by Pessimistic

This report is public

December 29, 2023

Abstract

In this report, we consider the security of smart contracts of HodlessBotToken project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of HodlessBotToken smart contracts. We described the audit process in the section below.

The initial audit showed three issues of medium severity: No documentation, Minimum swap return and Admin role. Also, several low-severity issues were found.

The initial code quality was mediocre. It did not contain any critical issues, however there were a lot of low severity issues for the project size. We recommended reviewing development process and writing a general technical documentation with the requirements.

After the audit, the developers provided us with an updated version of the code. In this version, they removed some functionality that was necessary for the initial code deployment and fixed several low-severity issues. However, the update also introduced two new low-severity issues.

General recommendations

We recommend fixing the remaining issues and writing documentation for the project.

Project overview

Project description

For the audit, we were provided with HodlessBotToken project on a private GitHub repository, commit c5c7a689502486b3abc2964f9d3091be81a7674e.

The scope of the audit included **HodlessBotToken.sol** file.

The project has no documentation.

10 tests out of 12 pass successfully. The code coverage is 82.35%.

The total LOC of audited sources is 160.

Codebase update

After the initial audit, the developers provided us with a new version of the code (commit 1b71723ab490289250b6cb87aa2f74aa7adf4e8e). In that update, the developers changed the code keeping in mind that the latest version will be an update to an already existing one and is not intended to be deployed on a new network. The update includes fixes for several low-severity issues, introduces two new issues, and adds one new test.

13 tests out of 13 pass successfully. The code coverage is 70%.

Token details

Name:	Hodless BOT
Symbol:	HBOT
Decimals:	18
Total Supply:	10000000 * 10**18

Audit process

We started the audit on November 15, 2023 and finished on November 21, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- The correctness of the Uniswap V2 integration;
- Compliance with the ERC20 token standard;
- The correctness of the Uniswap V2 integration;
- Correctness of role responsibilities.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts. We also sent the results to the developers in the text file.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, we discussed the results with the developers. On November 26, 2023, the developers provided us with an updated version of the code. In this update, they fixed some of the issues from our report and removed the functionality necessary for the token initialization. This resulted in a more optimized code. However, the update also introduced several low-severity issues, which can be found in the L11 and L12 sections.

We reviewed the updated codebase and ran Slither tool as well as our plugin Slitherin.

Finally, we updated the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. No documentation

The project has no documentation. We recommend adding it since it helps to find discrepancies between the business logic and the implementation.

M02. Minimum swap return

At line 205 of **HodlessBotToken.sol**, the token swap function call uses zero value for the `minReturn` argument, opening the possibility of sandwich attacks. We recommend using TWAP to determine the `minReturn` argument value.

M03. Admin role

The project heavily relies on the `DEFAULT_ADMIN_ROLE` role. It has the following powers:

- withdraw ether and any tokens from the contract address. This includes Uniswap LP tokens;
- set router address;
- set maximum swap amount;
- set maximum swap amount;
- whitelist any account;
- deploy Uniswap V2 pair.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Initialization of the implementation contract

The common pattern for upgradeable contracts is to move all the contract initialization into `initialize` function. This allows to initialize proxy storage. However, there is an attack vector that exploits `initialize` function by making a call directly to the logic contract. The current implementation is safe from this type of attack. Nevertheless, we recommend preventing logic contract initialization in order to make future versions of the code more secure.

L02. Precision issue (fixed)

Line 199 of the **HodlessBotToken.sol** performs division before multiplication. We recommend changing the order of the operations in order to increase precision of the result.

The issue has been fixed and is not present in the latest version of the code.

L03. Redundant code (fixed)

At line 120 of **HodlessBotToken.sol**, the contract approves Uniswap LP tokens to the router. However, as the contract currently does not support any interactions with the router besides token swaps, this approval cannot be used.

The issue has been fixed and is not present in the latest version of the code.

L04. Redundant code (fixed)

In **HodlessBotToken.sol** file there is a `_swapPathTo` variable that is used in the swap as an output token parameter. This variable is not marked as `immutable` and it can be changed by calling `setSwapPath` function. However, according to the implementation of the `swapExactTokensForETHSupportingFeeOnTransferTokens` function, the last parameter of the swap path should be equal to `WETH` token address. Hence, we recommend marking `_swapPathTo` variable as `immutable` and removing setter function from the code.

The issue has been fixed and is not present in the latest version of the code.

L05. Remove unused approval (fixed)

In `setRouterAddress` function, we recommend setting approval to zero for the previous router.

The issue has been fixed and is not present in the latest version of the code.

L06. Receive function

In **HodlessBotToken.sol** there is a `receive` function that accepts ether, but has no logic implemented. Since this function is only used for the contract initialization (i.e. for the `beDominator` function call), we recommend restricting access in order to prevent users from accidentally sending ether to the contract.

L07. Typo (fixed)

There is a typo in the **HodlessBotToken.sol** at lines 156-157. The correct variable name is `taxFree` instead of `taxFee`.

The issue has been fixed and is not present in the latest version of the code.

L08. Unchecked token transfer return

In **HodlessBotToken.sol**, `withdrawalToken` function does not validate token `transfer` return value. According to ERC20 token standard: "callers MUST handle `false` from `returns (bool success)`. Callers MUST NOT assume that `false` is never returned!". However, since some of the tokens do not return any return value (for example, USDT token), we recommend using SafeERC20 library when working with unknown tokens.

L09. Zero token transfers (fixed)

According to ERC20 token standard it should be possible to transfer zero tokens. We recommend saving gas on such transfers and not performing swaps for them.

The issue has been fixed in commit 75603bd78e0e28075700aa641b2f9cace5b2f3e0.

L10. Production parameters (fixed)

HodlessBotToken contract charges a 4% fee from the transfers. On such transfers, it sells the accumulated fees, but no more than `maxSwapByAmount`, which equals to 15% of the transfer amount. It means, that the contract will try to sell `currentBalance - 1` tokens and lines 199-200 are redundant.

After this, the value of `currentBalance - 1` is checked against the `_maxSwapAmount` value. If the balance is greater, then `_maxSwapAmount` of tokens are swapped. However, the current value of `_maxSwapAmount` is `25000` tokens. It is not reachable since `25000 / 0.04 = 625000`, which is greater than `_maxAmountInOnWallet = 250000`.

The issue has been fixed and is not present in the latest version of the code.

L11. ERC20 standard violation (fixed)

According to the ERC20 token standard, transfers of zero token values must trigger the `Transfer` event. However, in the current code version, the event is not emitted in the `transfer` function for zero token amounts.

The issue has been fixed in commit 75603bd78e0e28075700aa641b2f9cace5b2f3e0.

L12. Message sender value (fixed)

In the code, the message sender's address is obtained in two different ways: by using the `_msgSender` function and by directly using `msg.sender` value. We recommend using only one approach for consistency.

The issue has been fixed in commit 8ac6c162cff3a3ba6cff4211860ed87822e3a3e2.

Notes

N01. The code from the past uniswap initialization (fixed)

Token code is already deployed on the mainnet and its `beDominator()` function had already been called. As a result, the code at lines 98-101 might be deleted since `_launchTime` cannot be equal to `block.timestamp`. This will reduce transaction costs.

The issue has been fixed and is not present in the latest version of the code.

N02. Token withdrawal logic (fixed)

The code includes logic for withdrawing accidentally sent tokens. Currently, this code both sends all tokens to the user and gives approval to the contract owner. We recommend separating these actions, either sending the tokens or providing approval.

The issue has been fixed and is not present in the latest version of the code.

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Nikita Kuznetsov, Security Engineer
Irina Vikhareva, Project Manager
Konstantin Zherebtsov, Business Development Lead
Alexander Seleznev, Founder

December 29, 2023