ETHOS
AUDIT

Bean Chart

Audit Report

Mar. 22, 2022

Contents

Executive Summary

Audit Details

Project Name	Bean Chart
Codebase	https://bscscan.com/address/0x2b970c9d1e87b0b18a4911068a84213568696a18#code
Source Code	BeanChart.sol
Initial Audit Date	Mar. 22, 2022
Revision Dates	NA
Methodology	Manual

Methodology

This audit's objectives are to evaluate:

- Security-related issues
- Code quality
- Relevant documentation
- Adherence to specifications
- Adherence to best practices

This audit examines the possibility of issues existing along the following vectors (but not limited to):

- Single & Cross-Function Reentrancy
- Front Running (Transaction Order Dependence)
- Timestamp dependence
- Integer Overflow and Underflow
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Number rounding errors
- DoS with (Unexpected) Revert
- DoS with Block Gas Limit
- Insufficient gas griefing
- Forcibly sending native currency
- Logical oversights
- Access control
- Centralization of power
- Logic-Specification Contradiction
- Functionality duplication
- Malicious token minting

The code review conducted for this audit follows the following structure:

1. Review of specifications, documentation to assess smart contract functionality
2. Manual, line-by-line review of code
3. Code's adherence to functionality as presented by documentation
4. Automated tool-driven review of smart contract functionality
5. Assess adherence to best practices
6. Provide actionable recommendations

Contract Details

Contract IDs	**NA**
Network	BSC
Language	Solidity
Compiler	v0.8.12+commit.f00d7308
Verification Date	Mar. 15, 2022
Contract Type	BEP-20 Token
Libraries	Open Zeppelin

Token Details

Contract Name	BeanChart
Symbol	BCT
Decimals	9
Total Supply	10,000,000,000
Max Tx Amount	10,000,000,000
Max Wallet Amount	10,000,000,000
Total Tx Tax %	12
Miner Reflection %	5
Liquidity Provision %	3
Marketing Reflection %	4
Liquidity Provision Trigger	10,000

Risk Levels

LOW

The issue is informational and does not pose an immediate risk, but is relevant to security best practices.

MEDIUM

The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.

HIGH

The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.

EXTREME

The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.

Result Summary

Ethos' audit of the BeanChart token smart contract has concluded with a **POSITIVE** result.

- Bean Chart is a token project extending from the existing Baked Beans miner ecosystem.

- Bean Chart token is designed to be an ecosystem token which generates auto-liquidity provision, marketing wallet replenishment and Baked Beans miner rewards allocation from transaction fees.

- Buy/Sell tax values can be updated by the contract owner, however this is by design according to the team. As long as changes are implemented with proper community transpareny, this is not a huge issue.

- All other issues were informational only, and acknowledged by the team. No revision is needed since they do not impact the overall security of the token mechanics.

- While the Buy/Sell taxes can be set to any specified value, the distribution of accumulated reflections is determined by the *Share* values.

- Accumulated reflections are distributed as BNB to a Marketing wallet, Miner wallet, and also added LP povision over time.

- While taxes are taken on all buy/sell transactions (unless excluded from fees) reflections to respective wallets are only triggered on Sell transactions. This causes buy transactions to cost significantly less gas than sell transactions. This is not an issue on the Binance Smart Chain.

- The token smart contract is well formed and complete in it's design and therefore, it recieves a **POSITIVE** audit result.

Issues Reported

Severity	Unresolved	Acknowledged	Resolved
Extreme	0	0	0
High	0	0	0
Medium	0	1	0
Low	0	4	0

Issues Summary

ID	Title	Severity	Status
BCT-0	Owner Privilege	Medium	Acknowledged
BCT-1	State variable visibility is not set	Low	Acknowledged
BCT-2	Unhandled return value	Low	Acknowledged
BCT-3	Environment & function/variable naming mismatch	Low	Acknowledged
BCT-4	Functions that could be declared external	Low	Acknowledged

Detailed Findings

BCT-0 – Owner Privilege

Severity: Medium **Status**: Acknowledged

Description: The "*addLiquidity*" function calls the uniswapV2Router.addLiquidityETH function with the "to" address specified as owner() for acquiring the generated LP tokens from the WWI-BNB pool. The contract also allows the owner to update the buy/sell taxes, max wallet size and transaction sizes.

Risk: Over time the _owner address will accumulate a significant portion of LP tokens. If the _owner is an Externally Owned Account, mishandling of its private key can have devastating consequences to the project as a whole.

Recommendation: We advise the to address of the uniswapV2Router.addLiquidityETH function call to be replaced by the contract itself, i.e. address(this), and to restrict the management of the LP tokens within the scope of the contract's business logic. This will also protect the LP tokens from being stolen if the _owner account is compromised. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, i.e. Multisig wallets.

Feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency
- Assignment of privileged roles to multi-sig wallets to prevent single point of failure
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Team Comment: The team acknowledges that there is a centralization risk in the smart contract not just from the location of the auto-liquidity but also some privileged functions. However this was determined to be a feature of the project in order to allow the maximum flexibility to adapt with the full backing and transparency given to the community.

BCT-1 – Variables can be declared as 'constant'

Severity: Low **Status**: Acknowledged

Description: Variables *name*, *symbol* and *decimals* could be declared as constant since these state variables are never to be changed.

Risk: This is a minor gas optimization issue.

Recommendation: We recommend declaring these variables as 'constant' if they aren't going to be changed.

BCT-2 – Unhandled return value

Severity: Low **Status**: Acknowledged

Description: The return value of the function call "*addLiquidityETH*" is not checked or handled.

Risk: Execution will resume even if the "*addLiquidityETH*" throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behaviour in the subsequent program logic.

Recommendation: We recommend using variable to receive the return value of the "*addLiquidityETH*" function call and handle both success and failure scenarios.

SWC Registry: SWC-104

BCT-3 – Environment & function/variable naming mismatch

Severity: Low **Status**: Acknowledged

Description: The contract uses Pancakeswap for swapping and liquidity adds using BNB, however, functions and variable are named with Uniswap and Ethereum.

Risk: Mismatched function and variables names from the environment in which a smart contract operates can cause confusion.

Recommendation: We recommend changing Uniswap and ETH to Pancakeswap and BNB.

BCT-4 – Functions that could be declared external

Severity: Low **Status**: Acknowledged

Description: Several functions are declared as public visibility, however, since they are never called by the contract they should be declared external.

Risk: This is a gas optimization issue.

Recommendation: We recommend that functions that are never called by the contract to be declared as external to save gas.

Code Documentation

The code has a moderate amount of comments. This could be improved in order to help others understand the contract.

Adherence to Specifications

The smart contract adheres to the smart contract functionality described by the Bean Chart team and is in line with its intended usage.

Adherence to Best Practices

The smart contract adheres to the majority of best practices associated with a standard BEP-20 token.