QuillAudits

Audit Report
March, 2022

For

Centaurus

Contents

Overview

Centaurus Token by The Centaurus

The Centaurus is the indigenous utility token of the platform, which is a BEP-20 standard token built on the Binance Smart Chain network.

Scope of Audit

The scope of this audit was to analyze Centaurus smart contract's codebase for quality, security, and correctness.

Centaurus Contract deployed at:

https://bscscan.com/address/0xD9619614F8D939DF2002a0C5cBCc56496Fc3ad7A

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- BEP20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Mythril, Slither, C4udit , Solhint

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	3	7
Closed	0	0	0	0

Issues Found – Code Review / Manual Testing

High severity issues

No issues found

Medium severity issues

No issues found

Low severity issues

1. **Not complying with BEP20 standard completely.**

   BEP20 standard makes it mandatory for the tokens to define a function as getOwner, which should return the owner of the token contract.

5.1.1.6 getOwner

```
function getOwner() external view returns (address);
```

- Returns the bep20 token owner which is necessary for binding with bep2 token.
- **NOTE** - This is an extended method of EIP20. Tokens which don't implement this method will never flow across the Binance Chain and Binance Smart Chain.

   However, the token contract doesn't implement/define any such function, as a result the token may not flow across the Binance Chain and Binance Smart Chain, as stated by BEP20 interface documentation.

   Recommendation
   Consider adding the getOwner function.

   Status: Acknowledged

2. **Using Block values as a proxy for time**

   Here in function lock() and unlock() A control flow decision is made based on The 'block.timestamp' environment variable. Note that the values of variables like coinbase, gaslimit, block number,and timestamp are predictable and can be manipulated by malicious miners. Also, keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that the use of these variables introduces a certain level of trust into miners.

```
function lock(address[] memory accounts, uint256 setTime) public onlyOwner returns (bool) {
    for(uint256 i = 0; i < accounts.length; i++) {
        _lock[accounts[i]] = block.timestamp.add(setTime.mul(86400));
    }
    return true;
}

function unlock(address[] memory accounts) public onlyOwner returns (bool) {
    for(uint256 i = 0; i < accounts.length; i++) {
        _lock[accounts[i]] = block.timestamp;
    }
    return true;
}
```

Recommendation
Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use of oracles.

Status: Acknowledged

3. **Centralization risk - Owner Can permanently lock any user account by setting lock time far ahead in the future.**

   Here function lock() and unlock() are controlled by the owner and an owner can lock an account for a very long time and prevent users from utilizing other functionalities of the contract.

   Recommendation
   A proper guideline must be provided about when a user account can be locked and when it can be unlocked. Additionally, if possible, there must be a fixed time and after that account must be automatically unlocked.

   Status: Acknowledged

Informational issues

4. Unlocked pragma (pragma solidity ^0.8.9)

Contracts should be deployed with the same compiler version and flags that the have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation
Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Status: Acknowledged

5. BEP20 approve race condition

The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast ay transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender is able to spend their entire approval amount twice. As the BEP20 standard is proposed by deriving the ERC20 protocol of Ethereum, the race condition exists here as well.

Reference
1. https://eips.ethereum.org/EIPS/eip-20
2. https://github.com/binance-chain/BEPs/blob/master/BEP20.md#5119-approve

Status: Acknowledged

6. Centralization Risk

Centaurus is a fixed supply token and mints the 50% of the total tokens liquidity to the deployer of the contract. The implementation is prone to centralization risk that may arise, if the deployer loses its private key.

Recommendation
Ensure the private keys of the owner account are diligently handled.

Status: Acknowledged

7. Ownership Transfer must be a two step process.

The ownerTransfership() function in contract allows the current admin to transfer his privileges to another address. However, inside ownerTransfership() , the newOwner is directly stored into the storage, owner, after validating the newOwner is a non-zero address, which may not be enough.

```
function ownerTransfership(address newOwner) public onlyOwner returns(bool){
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _transfer(msg.sender,newOwner,_balances[msg.sender]);
    Owner = newOwner;
    return true;
}
```

As shown in the above code snippets, newOwner is only validated against the zero address. However, if current admin enters a wrong address by mistake, he would never be able to take the management permissions back. Besides, if the newOwner is the same as the current admin address stored in _owner , it’s a waste of gas.

Recommendation
It would be much safer if the transition is managed by implementing a two-step approach: _transferOwnership() and _updateOwnership() . Specifically, the _transferOwnership () function keeps the new address in the storage, _newOwner ,instead of modifying the _owner() directly. The updateOwnership() function checks whether _newOwner is msg.sender , which means _newOwner signs the transaction and verifies himself as the new owner. After that, _newOwner could be set into _owner.

Status: Acknowledged

8. Unused internal function

   In the contract, there is an unused internal function named _setupDecimals() which is not used. It is recommended to remove this function.

   Status: Acknowledged

9. Public functions that could be declared external inorder to save gas.

   Whenever a function is not called internally, it is recommended to define them as external instead of public in order to save gas. Here there are many functions (lock, unlock(), etc) which can be declared external.

   Status: Acknowledged

10. Missing event in ownerTransfership()

    Whenever certain significant privileged actions are performed within the contract, it is recommended to emit an event about it. Transferring ownership of the contract to a new address, is a significant action hence it is recommended to emit an event for this action.

Recommendation
Emit an event when ownership of the contract is changed.

Status: Acknowledged

Automated Testing

Slither:

```
BEP20.ownerTransfership(address) (contracts/Token.sol#106-111) should emit an event for:
        - Owner = newOwner (contracts/Token.sol#109)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

BEP20.constructor(string,string,uint8,address,address,address,address,address).Partners_Advisers (contracts/Token.sol#87) lacks a zero-check on :
                - PartnersandAdvisers = Partners_Advisers (contracts/Token.sol#93)
BEP20.constructor(string,string,uint8,address,address,address,address,address)._Marketing (contracts/Token.sol#87) lacks a zero-check on :
                - Project_Marketing = _Marketing (contracts/Token.sol#94)
BEP20.constructor(string,string,uint8,address,address,address,address,address)._Development (contracts/Token.sol#87) lacks a zero-check on :
                - Project_Development = _Development (contracts/Token.sol#95)
BEP20.constructor(string,string,uint8,address,address,address,address,address)._Team (contracts/Token.sol#87) lacks a zero-check on :
                - Project_Team = _Team (contracts/Token.sol#96)
BEP20.constructor(string,string,uint8,address,address,address,address,address)._Charity (contracts/Token.sol#87) lacks a zero-check on :
                - Project_Charity = _Charity (contracts/Token.sol#97)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

BEP20._transfer(address,address,uint256) (contracts/Token.sol#153-163) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_lock[sender] <= block.timestamp,sender address is locked) (contracts/Token.sol#156)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

BEP20._setupDecimals(uint8) (contracts/Token.sol#189-191) is never used and should be removed
Context._msgData() (contracts/Token.sol#32-35) is never used and should be removed
SafeMath.sub(uint256,uint256) (contracts/Token.sol#45-48) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.9 (contracts/Token.sol#6) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable BEP20.Owner (contracts/Token.sol#79) is not in mixedCase
Variable BEP20.PartnersandAdvisers (contracts/Token.sol#81) is not in mixedCase
Variable BEP20.Project_Marketing (contracts/Token.sol#82) is not in mixedCase
Variable BEP20.Project_Development (contracts/Token.sol#83) is not in mixedCase
Variable BEP20.Project_Team (contracts/Token.sol#84) is not in mixedCase
Variable BEP20.Project_Charity (contracts/Token.sol#85) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (contracts/Token.sol#33)" inContext (contracts/Token.sol#27-36)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Centaurus.constructor(address,address,address,address,address) (contracts/Token.sol#212-214) uses literals with too many digits:
        - _mint(msg.sender,100000000000 * 10 ** 8) (contracts/Token.sol#213)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

ownerTransfership(address) should be declared external:
        - BEP20.ownerTransfership(address) (contracts/Token.sol#106-111)
name() should be declared external:
        - BEP20.name() (contracts/Token.sol#113-115)
symbol() should be declared external:
        - BEP20.symbol() (contracts/Token.sol#117-119)
decimals() should be declared external:
        - BEP20.decimals() (contracts/Token.sol#121-123)
totalSupply() should be declared external:
        - BEP20.totalSupply() (contracts/Token.sol#125-127)
balanceOf(address) should be declared external:
        - BEP20.balanceOf(address) (contracts/Token.sol#129-131)
allowance(address,address) should be declared external:
        - BEP20.allowance(address,address) (contracts/Token.sol#138-140)
approve(address,uint256) should be declared external:
        - BEP20.approve(address,uint256) (contracts/Token.sol#142-145)
transferFrom(address,address,uint256) should be declared external:
        - BEP20.transferFrom(address,address,uint256) (contracts/Token.sol#147-151)
lock(address[],uint256) should be declared external:
        - BEP20.lock(address[],uint256) (contracts/Token.sol#195-200)
unlock(address[]) should be declared external:
        - BEP20.unlock(address[]) (contracts/Token.sol#202-207)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Functional Testing Results

- ☑ Should get the owner of the contract.
- ☑ Should get the name of the contract.
- ☑ Should get a symbol of the contract.
- ☑ Should get decimals of the contract.
- ☑ Should get totalSupply of the contract.
- ☑ Should check the address of (Partners Advisers, Marketing, Development, Team, Charity).
- ☑ Should check the share of the owner must be 50% of the total minted tokens.
- ☑ Should check the share of Partners_Advisers must be 10% of the total minted tokens.
- ☑ Should check the share of _Marketing must be 15% of the total minted tokens.
- ☑ Should check the share of _Development must be 15% of the total minted tokens
- ☑ Should check the share of _Team must be 5% of the total minted tokens.
- ☑ Should check the share of _Charity must be 5% of the total minted tokens.
- ☑ Should check ownership transfer functionality.
- ☑ Should Revert if the provided address of the new owner is a zero address.
- ☑ OwnerTransfership() can only be called by the owner.
- ☑ Should check if tokens are transferred.
- ☑ Should revert on transfer to zero address.
- ☑ Should revert if the account is locked.
- ☑ Should update balances correctly.
- ☑ Should call approve to another account
- ☑ Allowance between two accounts should be returned correctly.
- ☑ Should test transferFrom function
- ☑ Should revert is exceeds the allowance
- ☑ Should update the balances of respective accounts correctly
- ☑ Should correctly update allowance value
- ☑ Should Lock and Unlock accounts.
- ☑ Lock and Unlock should be called only by owner.

Closing Summary

Some issues of Low and informational severity were found,which was Acknowledged by the Centaurus team. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Centaurus platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Centaurus Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
March, 2022

For

Centaurus

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com