BB21 SMART CONTRACTS FINAL AUDIT REPORT

BY CRYPTICOCEAN
MARCH 1ST, 2021

On February 24, 2021, the CrypticOcean Audit Security Team received the BB21 team's application for smart contract security Audit of the BB21 token and locking contract.

The following are the details and results of this smart contract security Audit :

Project Name:

- B21

Smart Contract Files:

- BB21.sol
- swapBsc.sol
- swapEth.sol
- liquidity Provider.sol

Audit Item	Audit SubClass	Audit result
Overflow	—	Passed
Race Condition	—	Passed
Permissions	Permission vulnerability Audit Excessive Auditing Authority	Passed
Safety Design	Zeppelin safemath	Passed
DDOS Attack	Call function security	Passed
Gas Optimization	—	Passed
Design Logic	—	Passed
Know Attacks	—	Passed

Audit Result: Passed

Audit Team: CrypticOcean Security Team

(Statement: CrypticOcean Technologies only issues this report based on the facts that have occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts that occur or exist later after the report, CrypticOcean cannot judge the security status of its smart contract. CrypticOcean is not responsible for it. The security Audit analysis and other contents of this report are based on the documents and materials provided by the information provider to CrypticOcean as of the date of this report (referred to as “the provided information”). CrypticOcean assumes that there has been no information missing, tampered with, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, CrypticOcean will not bear any responsibility for the resulting loss and adverse effects.)

INTRODUCTION

This Audit Report will only highlight the overall security and business logic of B21 Smart Contract. With this report, we have tried to ensure the reliability, security of their smart contract by complete assessment of their system's architecture, and the smart contract codebase.

AUDITING APPROACH AND METHODOLOGIES APPLIED

The CrypticOcean team has performed thorough testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries. Our team then performed a formal line by line inspection of the code of the Smart Contract to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks as mentioned in the above table. In the Unit testing Phase, we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration with our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the process.
- Analysing the complexity of the code by thorough, manual review of the code, line-by-line.

- Deploying the code on testnet using multiple clients to run live tests
- Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

AUDIT DETAILS

- Project Name: B21
- Languages: Solidity (Smart contract), Javascript (Unit Testing)
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Slither, Surya

SUMMARY OF B21 SMART CONTRACT

CrypticOcean conducted a Security Audit of a smart contract of B21. B21 contract is used to create the BEP20 token, which is a BB21, Smart contract contains basic functionalities of a BEP20 token.

Name: BB21

Symbol: BB21

OTHER CONTRACTS

Locking contract for Binance Smart Chain

Locking Contract for Ethereum

Liquidity Reward contract for Ethereum/BSC

And some advanced features other than essential functions.

- Lock tokens on ETH contract to mint over Binance smart chain (Interoperability)
- Liquidity provider token staking and rewards

AUDIT GOALS

The focus of the Audit was to verify that the smart contract system is secure, resilient and working according to its specifications that have been provided to the Auditing team. The Audit activities can be grouped into the following three categories:

Security: Identifying security related issues within each contract and the system of contracts.

Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

https://crypticocean.com hello@crypticocean.com

SECURITY LEVEL REFERENCES

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed as recommended.

Medium severity issues could potentially bring problems and should eventually be fixed according to suggestions and recommendations.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

NUMBER OF ISSUES PER SEVERITY

	Low	Medium	High
Open	0	0	0
Closed	2	0	0

High severity issues:-

No High Severity Issue.

Medium Severity Issues:-

No Medium Severity Issue.

Low Severity Issues:-

1. SafeMath is re-used:

SwapEth.sol#79-155

- BB21.sol#132-266
- B21.sol#12-36
- BB21.sol#132-266
- SwapBsc.sol#79-155
- BB21.sol#132-266
- LiquidityProvider.sol#86-162
- BB21.sol#132-266

Status: Fixed

2. liquidityProviderToken (LiquidityProvider.sol#165-275) inherits from a contract for which the name is reused.

-Pausable (SwapEth.sol#38-63)

Status: Fixed

UNIT TESTING

TEST SUITE

CONTRACT: B21 TOKEN CONTRACTS

- ✓ Should check a name of a token BB21
- ✓ Should check a symbol of a token
- ✓ Should check a decimal of a token
- ✓ Should check an owner of a token (61ms)
- ✓ Should check a balance of a token contract
- ✓ Should check a balance of an owner (73ms)
- ✓ Should correctly initialize constructor values of BB21 Token Contract (73ms)
- ✓ Should correctly initialize constructor values of BB21 Token Contract (63ms)
- ✓ Should check the balance of an Owner
- ✓ Should correctly initialize constructor values of liquidity Contract (54ms)
- ✓ Should correctly initialize constructor values of Swap Contract (52ms)
- ✓ Should check the address of liquidity token contract at locking
- ✓ Should check the address of liquidity token contract at locking
- ✓ Should check three-month locking per cent
- ✓ Should check Six-month locking per cent
- ✓ Should check Nine-month locking per cent
- ✓ Should check Twelve-month locking per cent
- ✓ Should add Address of an IP token
- ✓ Should check the address of liquidity token contract at locking
- ✓ Should check the address of liquidity token contract at locking using function

- ✓ Should check address of liquidity token contract at locking using function
- ✓ Should check address of liquidity token contract at locking using function
- ✓ Should check address of liquidity token contract at locking using function
- ✓ Should check claimable tokens
- ✓ Should check address of liquidity token contract at locking using function
- ✓ Should check address of b21 token contract at locking
- ✓ Should check address of b21 fees collection at locking
- ✓ Should check fees of b21 locking in ETH
- ✓ Should check if address is subadmin or not locking in token
- ✓ Should add subadmin (47ms)
- ✓ Should add subadmin accounts 2 (40ms)
- ✓ Should check if address is subadmin or not locking in token
- ✓ Should check if address is subadmin or not locking in token
- ✓ Should Not add subadmin by non owner account (64ms)
- ✓ Should Not add subadmin by non owner account
- ✓ Should remove subadmin accounts 2
- ✓ Should set fees in ETH
- ✓ Should check if the address is subadmin or not locking in token
- ✓ Should check the Total Supply of BTCCToken Tokens
- ✓ Should check the Name of a token of BTCC Token contract
- ✓ Should check the symbol of a token of BTCCToken contract
- ✓ Should check the decimal of a token of BTCCToken contract
- ✓ Should check the balance of a Owner
- ✓ Should check the owner of a contract

- ✓ Should check the balance of a contract
- ✓ Should Not be able to transfer tokens to accounts[1] without having token (56ms)
- ✓ Should be able to transfer tokens to accounts[1] (99ms)
- ✓ Should Approve address[3] to spend specific token on the behalf of owner (39ms) ✓ should increase the allowance (75ms)
- ✓ Should decrease the allowance (66ms)
- ✓ Should Not be able to transfer tokens to accounts[3] it self after approval from accounts[0] more then allowed (89ms)
- ✓ Should be able to burn tokens (58ms)
- ✓ Should check the balance of a Owner after burn
- ✓ Should check the Total Supply of BSCB21 Tokens
- ✓ Should Approve swap contract by account 0 (43ms)
- ✓ Should lock token with fees in token
- ✓ Should check the balance of a Owner (53ms)
- ✓ Should check the balance of a locking contract
- ✓ Should check fees of b21 locking in eth
- ✓ Should lock token with fees in token
- ✓ Should check the balance of a Owner (52ms)
- ✓ Should check the balance of a locking contract
- ✓ Should Approve swap contract by account 0 to spend LP tokens
- ✓ Should lock tokens (75ms)

Final Result of Test:

✓ **70 Passing (3s) PASSED**

✗ **0 Failed**

AUTOMATION TOOL TESTING: SLITHER, MYTHRIL, ECHIDNA, MANTICORE

```
INFO:Detectors:
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (SwapEth.sol#6)
setbaseFees(uint256,uint256) should be declared external:
        - LockingEB21.setbaseFees(uint256,uint256) (SwapEth.sol#181-186)
addSubAdmin(address,uint256) should be declared external:
        - LockingEB21.addSubAdmin(address,uint256) (SwapEth.sol#188-193)
removeSubAdmin(address) should be declared external:
        - LockingEB21.removeSubAdmin(address) (SwapEth.sol#195-199)
balanceOf(address) should be declared external:
        - BasicToken.balanceOf(address) (B21.sol#80-82)
        - ERC20Basic.balanceOf(address) (B21.sol#45)
transfer(address,uint256) should be declared external:
        - ERC20Basic.transfer(address,uint256) (B21.sol#46)
        - BasicToken.transfer(address,uint256) (B21.sol#64-73)
transferFrom(address,address,uint256) should be declared external:
        - StandardToken.transferFrom(address,address,uint256) (B21.sol#115-125)
approve(address,uint256) should be declared external:
        - StandardToken.approve(address,uint256) (B21.sol#137-141)
allowance(address,address) should be declared external:
        - StandardToken.allowance(address,address) (B21.sol#149-151)
increaseApproval(address,uint256) should be declared external:
        - StandardToken.increaseApproval(address,uint256) (B21.sol#159-163)
decreaseApproval(address,uint256) should be declared external:
        - StandardToken.decreaseApproval(address,uint256) (B21.sol#165-174)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-as-external
```

```
INFO:Detectors:
BB21.constructor(address) (BB21.sol#351-359) uses literals with too many digits:
        - _totalSupply = 500000000000000000000000000 (BB21.sol#355)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
ERC20Basic.totalSupply (B21.sol#44) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

```
INFO:Detectors:
Variable Migrations.last_completed_migration (Migrations.sol#6) is not in mixedCase
Parameter Owned.transferOwnership(address)._newOwner (SwapEth.sol#27) is not in mixedCase
Event LockingEB21lockTokens(address,address,uint256) (SwapEth.sol#170) is not in CapWords
Parameter BasicToken.transfer(address,uint256)._to (B21.sol#64) is not in mixedCase
Parameter BasicToken.transfer(address,uint256)._value (B21.sol#64) is not in mixedCase
Parameter BasicToken.balanceOf(address)._owner (B21.sol#80) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._from (B21.sol#115) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._to (B21.sol#115) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._value (B21.sol#115) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._spender (B21.sol#137) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._value (B21.sol#137) is not in mixedCase
Contract liquidityProviderToken (LiquidityProvider.sol#165-275) is not in CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
```

```
INFO:Detectors:
Different versions of Solidity is used in :
        - Version used: ['0.5.16', '>=0.4.22<0.8.0']
        - 0.5.16 (BB21.sol#1)
        - >=0.4.22<0.8.0 (Migrations.sol#2)
        - 0.5.16 (SwapEth.sol#1)
        - 0.5.16 (B21.sol#5)
        - 0.5.16 (SwapBsc.sol#1)
        - 0.5.16 (LiquidityProvider.sol#5)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version0.5.16 (BB21.sol#1) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version>=0.4.22<0.8.0 (Migrations.sol#2) is too complex
Pragma version0.5.16 (SwapEth.sol#1) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version0.5.16 (B21.sol#5) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version0.5.16 (SwapBsc.sol#1) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version0.5.16 (LiquidityProvider.sol#5) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
```

```
INFO:Detectors:
BB21.allowance(address,address).owner (BB21.sol#419) shadows:
        - Ownable.owner() (BB21.sol#297-299) (function)
BB21._approve(address,address,uint256).owner (BB21.sol#545) shadows:
        - Ownable.owner() (BB21.sol#297-299) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

IMPLEMENTATION RECOMMENDATIONS

setCompleted(uint256) should be declared external:

- Migrations.setCompleted(uint256) (Migrations.sol#16-18)

balanceOf(address) should be declared external:

- ERC20.balanceOf(address) (SwapEth.sol#6)

setbaseFees(uint256,uint256) should be declared external:

- LockingEB21.setbaseFees(uint256,uint256) (SwapEth.sol#181-186)

addSubAdmin(address,uint256) should be declared external:

- LockingEB21.addSubAdmin(address,uint256) (SwapEth.sol#188-193)

removeSubAdmin(address) should be declared external:

- LockingEB21.removeSubAdmin(address) (SwapEth.sol#195-199)

balanceOf(address) should be declared external:

- BasicToken.balanceOf(address) (B21.sol#80-82)
- ERC20Basic.balanceOf(address) (B21.sol#45)

transfer(address,uint256) should be declared external:

- ERC20Basic.transfer(address,uint256) (B21.sol#46)
- BasicToken.transfer(address,uint256) (B21.sol#64-73)

transferFrom(address,address,uint256) should be declared external:

- StandardToken.transferFrom(address,address,uint256) (B21.sol#115-125)

approve(address,uint256) should be declared external:

- StandardToken.approve(address,uint256) (B21.sol#137-141)

allowance(address,address) should be declared external:

- StandardToken.allowance(address,address) (B21.sol#149-151)

increaseApproval(address,uint256) should be declared external:

- StandardToken.increaseApproval(address,uint256) (B21.sol#159-163)

decreaseApproval(address,uint256) should be declared external:

- StandardToken.decreaseApproval(address,uint256) (B21.sol#165-174)

COMMENTS

The use case of the smart contract is very well designed and Implemented. Overall, the code is well written and demonstrates effective use of abstraction, separation of concerns, and modularity. B21 development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.

All the bugs, suggestions and recommends has been considered by the B21 team and some of the issues they will handle on their own as those issues or calls have been handle by the only owner.