QuillAudits

Audit Report
September, 2022

For

B4REAL

Table of Content

Table of Content

Executive Summary

Project Name	B4Real
Overview	Basic token sale contract. Accepts one token as input. Transfer project token according to fixed rate. Crowdsale should be able to run until the owner calls the endCrowdsale function. If endCrowdsale timestamp is specified, then crowdsale ends at a fixed timestamp.
Method	Manual Review, Functional Testing, Automated Testing
Scope of Audit	The scope of this audit was to analyze B4Real codebase for quality, security, and correctness. **https://github.com/B4Biz/B4real-smart-contracts-Audit**

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	3	1	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	2	1	10

Types of Severities

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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

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.
- Implementation of ERC-20 token standards.
- 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

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - Crowdsale contract

High Severity Issues

A1. Wrong tokens purchased in case when decimals for input token > 18

Line	Function -
168	`uint256 tokenPurchased = inputTokenDecimals >= 18 ? _inputTokenAmount.mul(inputTokenRate[address(_inputToken)]).mul(// should be divide 10**(inputTokenDecimals - 18) ) : _inputTokenAmount.mul(inputTokenRate[address(_inputToken)]).mul( 10**(18 - inputTokenDecimals) );`

Description

Assume a case where input decimal value is 24 and token decimals is 18. The exchange rate is 10**18 that means 1 output token for 1 input token. In this case if a user sends 10**24 input tokens he will receive 10**30 output tokens, instead he should get back 10**18

Remediation

On line 168 it should be div instead of mul

```
uint256 tokenPurchased = inputTokenDecimals >= 18
            ?
_inputTokenAmount.mul(inputTokenRate[address(_inputToken)]).div(
                10**(inputTokenDecimals - 18)
            )
            :
_inputTokenAmount.mul(inputTokenRate[address(_inputToken)]).mul(
                10**(18 - inputTokenDecimals)
            );
```

Status

Resolved

Medium Severity Issues

A.2 Crowdsale tokens not transferred to contract

Line	Function -
287	`function updateMaxCrowdsaleAllocation(uint256 _crowdsaleTokenAllocated) external onlyOwner { crowdsaleTokenAllocated = _crowdsaleTokenAllocated; emit CrowdsaleTokensAllocationUpdated(crowdsaleTokenAllocated); }`

Description

When updating the tokens allocated for the crowdsale, the actual number of tokens on the contract should also be updated. This should be done atomically in a single transaction. For Eg: if allocated tokens increase from 100 to 200 the extra 100 tokens need to be transferred to the crowdsale contract for distribution.

Status

Acknowledged

Low Severity Issues

A.3 Update to latest solidity version

Line	Function -
2	`pragma solidity 0.7.6;`

Description

Pragma version is not latest. As mentioned in the solidity docs:

“When deploying contracts, you should use the latest released version of Solidity. Apart from exceptional cases, only the latest version receives security fixes. Furthermore, breaking changes as well as new features are introduced regularly.”

Status

Acknowledged

A.4 URLUpdated event update

Line	Function -

138	`function updateTokenURL(address tokenAddress, string memory _url)`
	`external`
	`onlyOwner`
	`{`
	`updateMetaURL(tokenAddress, _url);`
	`emit URLUpdated(_url);`
	`}`

Description

URL can be updated for multiple tokens so the event should also contain the token address for which the url is being updated

Status

Resolved

Informational Isuues

A.5 Solidity style guide

Description

https://docs.soliditylang.org/en/v0.8.16/style-guide.html#order-of-functions.

Order of functions is not as per the solidity style guide.

Status

Resolved

B. Contract - Metadata contract

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Isuues

B.1 Struct lose packing

Line	Function -

5	`struct TokenMetadata {`
	`address routerAddress;`
	`string imageUrl;`
	`bool isAdded;`
	`}`

Description

Simply reordering the order of variables defined in this struct can save gas. It occupies three storage slots at the moment and we can reorder it to occupy two slots which can save a significant amount of gas cost.
https://fravoll.github.io/solidity-patterns/tight_variable_packing.html
https://ethereum.stackexchange.com/questions/76168/does-packing-bools-alongside-addresses-reduce-storage-costs

Remediation

pack bool and address together

Status

Resolved

C. Contract - MerkleWhitelisting

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

D. Contract - StakingPoolUpdatableFixedAPR

High Severity Issues

D.1 Withdraw fee bypass

Description

An attacker can bypass the withdrawal fee by combining the D2 and D3 vulnerabilities. He can emergency withdraw just before the harvest period ends. His lp tokens will be withdrawn without any fee deduction. He can then deposit the same amount again immediately after harvest period ends getting all his rewards back.

Status

Resolved

Medium Severity Issues

D.2 rewardLockedUp not set to zero in emergencyWithdraw

Line	Function -
561	

```
    function emergencyWithdraw() external nonReentrant {
        UserInfo storage user = userInfo[msg.sender];

        if (user.amount > 0) {
            farmInfo.numFarmers--;
        }
        totalInputTokensStaked =
totalInputTokensStaked.sub(user.amount);
        user.amount = 0;

        uint256 totalRewardPools = rewardPool.length;
        for (uint256 i = 0; i < totalRewardPools; i++) {
            user.rewardDebt[rewardPool[i].rewardToken] = 0;
        }
        _updateRewardPerSecond();
        farmInfo.inputToken.safeTransfer(address(msg.sender),
user.amount);
        emit EmergencyWithdraw(msg.sender, user.amount);
    }
```

Description

rewardLockedUp for a user is not being set to zero, so a user can withdraw his locked up rewards even after he has emergency withdrawn his lp tokens. He can deposit some new small amount again and he will get back his locked up rewards.

Remediation

locked up rewards should be set to zero along with the reward debt.

Status

Resolved

D.3 No withdraw fee in emergencyWithdraw method

Line	Function -

561	`function emergencyWithdraw() external nonReentrant {`
	`UserInfo storage user = userInfo[msg.sender];`
	
	`if (user.amount > 0) {`
	`farmInfo.numFarmers--;`
	`}`
	`totalInputTokensStaked = totalInputTokensStaked.sub(user.amount);`
	`user.amount = 0;`
	
	`uint256 totalRewardPools = rewardPool.length;`
	`for (uint256 i = 0; i < totalRewardPools; i++) {`
	`user.rewardDebt[rewardPool[i].rewardToken] = 0;`
	`}`
	`_updateRewardPerSecond();`
	`farmInfo.inputToken.safeTransfer(address(msg.sender), user.amount);`
	`emit EmergencyWithdraw(msg.sender, user.amount);`
	`}`

Description

We want a user to call emrgencyWithdraw method only when there is an emergency but a user can use this method for his benefits. In some cases it might be possible that the reward that the user has collected is lesser than the fees he is paying, in that case he will prefer to call this method

Status

Acknowledged

Low Severity Issues

No issues found

Informational Issues

D.4 DoS attack possibility on for loops

Description

For loops are used. So care needs to be taken that the number of pools is not so much that it is more than the block gas limit. Just FYI in case this information is missing.

Remediation

proper gas usage checking everytime when adding a new reward pool.

Status

Acknowledged

D.5 blockRewardPersec redundant values

Line	Function -
381	

```
function _updateRewardPerSecond() internal {
        /*
            APR = ( SECONDS_IN_YEAR * RewardPerSecond * 100 )
/ Total deposited
            RewardPerSecond = ( APR * Total deposited ) / (
SECONDS_IN_YEAR )
        */
        uint256 totalRewardPools = rewardPool.length;
        for (uint256 i = 0; i < totalRewardPools; i++) {
            RewardInfo storage rewardInfo = rewardPool[i];
            uint256 effectiveRewardPerSecond = (
                expedtedAPR.mul(totalInputTokensStaked)
            ).div(SECONDS_IN_YEAR * 1e18);
            rewardInfo.blockRewardPerSec =
effectiveRewardPerSecond;
        }
    }
```

Description

blockRewardPerSec does not need to be updated for each pool object as it will always be the same for all the pools.

Remediation

blockRewardPerSec field can be removed from RewardInfo struct and a single variable can replace it.

Status

Acknowledged

D.6 Unnecessary code

Line	Function

218	`function _deposit(uint256 _amount, address _user) internal {`
	`UserInfo storage user = userInfo[_user];`
	`user.whiteListedHandlers[_user] = true;`

Description

user is being added as a whitelisted handler for himself. A user can use the withdrawal method directly. Why are we doing this? If this is the required behavior from the code please mark this issue as acknowledged.

Status

Resolved

D.7 Extra if check

Line	Function

530	`if (user.amount == _amount && _amount > 0) {`
	`farmInfo.numFarmers--;`
	`}`
	`if (_amount > 0) {`

Description

user is being added as a whitelisted handler for himself. A user can use the withdrawal method directly. Why are we doing this? If this is the required behavior from the code please mark this issue as acknowledged.

Status

Resolved

D.8 Unused event

Line	Function
90	`event MaxAllowedDepositUpdated(uint256 _maxAllowedDeposit);`

Description

event MaxAllowedDepositUpdated is not used

Status

Resolved

D.9 Unused function parameter

Line	Function

677	`function updateExpectedAPR(uint256 _expectedAPR, uint256 _rewardTokenIndex)`
	`external`
	`onlyOwner`
	`{`
	`massUpdatePools();`
	`expedtedAPR = _expectedAPR;`
	`_updateRewardPerSecond();// no use of _rewardTokenIndex as all the rewards are being updated`
	`emit ExpectedAprUpdated(_expectedAPR, _rewardTokenIndex);`
	`}`

Description

no need to pass _rewardTokenIndex as the reward per second for all the pools is updated and not for a single pool.

Status

Resolved

D.8 Unused event

Line	Function
6	`import "./library/IPolydexPair.sol";`

Description

not used anywhere

Status

Resolved

D.11 Solidity style guide

Description

https://docs.soliditylang.org/en/v0.8.16/style-guide.html#order-of-functions. Order of functions is not as per the solidity style guide.

Status

Resolved

E. Contract - StakingPoolUpdatableFixedAPRWhitelisting

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

E.1 Unused storage variable and function

Line	Function

80, 284	`uint256 public maxAllowedDeposit;`
	
	`function updateMaxAllowedDeposit(uint256 _maxAllowedDeposit)`
	`external`
	`onlyOwner`
	`{`
	`maxAllowedDeposit = _maxAllowedDeposit;`
	`emit MaxAllowedDepositUpdated(_maxAllowedDeposit);`
	`}`

Description

maxAllowedDeposit variable is not used in any of the methods.

Status

Resolved

Functional Testing

Contract: StakingPoolUpdatableFixedAPR

- ✓ should deploy
- ✓ should add a RewardToken
- ✓ shouldn't add an existing RewardToken
- ✓ should allow the user to deposit inputToken
- ✓ should allow the user to deposit inputToken for another user
- ✓ should allow the user to withdraw their inputToken
- ✓ should deduct the withdrawalFee on withdrawal of inputToken
- ✓ should update the reward on withdrawal of inputToken
- ✓ should update pending Reward correctly
- ✓ should update RewardDebt on deposit
- ✓ should allow the user to Emergency withdraw their inputToken
- ✓ should allow the user to whitelist another user
- ✓ should allow whitelisted user to withdraw inputToken from user deposit
- ✓ should successfully update the withdrawal Fee
- ✓ should successfully update the harvest interval
- ✓ should successfully update the expected APR
- ✓ should successfully set withdrawal Fee address
- ✓ should sucessfully withdraw reward when 2 users deposit
- ✓ should sucessfully withdraw reward when 2 users deposit after increasing APR

Contract: StakingPoolUpdatableFixedAPRWhitelisting

- ✓ should deploy
- ✓ should add a RewardToken
- ✓ shouldn't add an existing RewardToken
- ✓ should allow the user to deposit inputToken
- ✓ should allow the user to deposit inputToken for another user
- ✓ should allow the user to withdraw their inputToken
- ✓ should deduct the withdrawalFee on withdrawal of inputToken
- ✓ should update the reward on withdrawal of inputToken
- ✓ should update pending Reward correctly
- ✓ should update RewardDebt on deposit
- ✓ should allow the user to Emergency withdraw their inputToken
- ✓ should allow the user to whitelist another user
- ✓ should allow whitelisted user to withdraw inputToken from user deposit
- ✓ should successfully update the withdrawal Fee
- ✓ should successfully update the harvest interval
- ✓ should successfully update the expected APR
- ✓ should successfully set withdrawal Fee address
- ✓ should sucessfully withdraw reward when 2 users deposit
- ✓ should sucessfully withdraw reward when 2 users deposit after increasing APR

Test Coverage

Poor test coverage for contracts. No test cases for crowdsale contracts and only 30 percent for the staking contracts. Contract branch coverage of more than 95 percent is recommended before production deployment.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	48.31	28.31	37.18	48.69	
IERC20.sol	100	100	100	100	
MerkleWhitelisting.sol	100	100	100	100	
Metadata.sol	40	25	50	40	30,31,33
MinimumCrowdsale.sol	0	0	0	0	... 291,292,296
StakingPoolUpdatableFixedAPR.sol	57.95	32.86	48.28	58.43	... 692,694,713
StakingPoolUpdatableFixedAPRWhitelisting.sol	52.22	31.94	38.71	52.75	... 733,735,754
contracts/library/	47.06	25	50	50	
IPolydexPair.sol	100	100	100	100	
Ownable.sol	44.44	25	50	50	48,49,57,58,59
TransferHelper.sol	50	25	50	50	8,9,23,24
contracts/mock/	57.14	21.43	40.91	57.14	
MockERC20.sol	59.57	21.43	45	59.57	... 233,291,293
MockFeeManager.sol	0	100	0	0	6,10
All files	49.17	27.6	38.89	49.59	

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the B4Real contracts. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, 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 B4Real. 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 B4Real Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

600+
Audits Completed

$15B
Secured

600K
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2022

For

B4REAL

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com