

BondAppétit

smart contracts audit report

Prepared for:
BondAppétit.io

Authors: HashEx audit team
September 2021

Contents

Disclaimer	3
Introduction	4
Contracts overview	4
Found issues	5
Conclusion	12
References	12
Appendix A. Issues' severity classification	13
Appendix B. List of examined issue types	13

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (<https://hashex.org>).

Introduction

HashEx was commissioned by the BondAppétit team to perform an audit of BondAppétit's smart contracts. The audit was conducted between May 19 and May 24, 2021 for the first part and between August 20 and September 07, 2021 for the second.

The code located in the github repository @bondappetit/bondappetit-protocol was audited after the commit [21f9f5a](#) for the first part and [be4fef0](#) for the second one. Only 6 contracts from the repository were in the scope of the audit. The repository contains tests for audited contracts.

Documentation and white paper can be found on bondappetit.io.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts.
- Formally check the logic behind given smart contracts.

Information in this report should be used to understand the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The public audit by MixBytes of several BondAppétit contracts can be found in their GitHub [\[1\]](#).

Update: the BondAppétit team has responded to this report. Individual responses were added after each item in the [section](#). The updated code is located in the same repository after the [c22106c](#) commit. Updated contracts are deployed to the Ethereum mainnet:

0x377f59Eb8aBd0eFc1F33f45696A0A4Bb51CdA72F	VoteDelegator,
0xdF00204A61Df6F1d241b6751BD34B9a3e4930257	YieldEscrow,
0xD4E861Ead6E81Fe0EB20d0197c72590498E0e944	ProfitDistributor (lockPeriod = 2343110),
0x3C9fEea39ef1D4847813E380bb424677142cf2b1	ProfitDistributor (lockPeriod = 1145375),
0xda47F10c3c8128C9C85312af693c34bcbeFA2057	ProfitDistributor (lockPeriod = 543235),
0x17c665c20693EAD6D5d233ecAf48d37eD87a6c28	UniV2BuybackDepositaryBalanceView,
0x38fDc8C3E8409527F960Df83Cf0Be5274040d9fC	BuybackDepositaryBalanceView.

Contracts overview

YieldEscrow.sol

BondAppetit Governance yield token of ERC20 standard [\[2\]](#) with the functionality of exchanging yBAG tokens to BAG and vice versa.

VoteDelegator.sol

Exchange contract with vote delegation deployed by the YieldEscrow.

ProfitDistributor.sol

Staking contract with linear rewards distribution.

UniV2BuybackDepositoryBalanceView.sol

Buyback contract with restricted callable functions. Further referred to as Buyback.

Market.sol

Exchange contract with ChainLink price feed.

BuybackDepositoryBalanceView.sol

Buyback contract for stable tokens.

Found issues

ID	Title	Severity	Response
01	VoteDelegator: ownership transfers	High	Fixed
02	ProfitDistributor: direct rewards transfers	High	Fixed
03	ProfitDistributor: rewardsDuration is monotonically non-increasing	High	Fixed
04	Market: changeCumulativeToken() + withdraw() allows owner to take all the balance	High	Fixed
05	Market: setting priceFeed problem	High	Fixed
06	BuybackDepositoryBalanceView: productAmount exceeds balance	High	Fixed
06	Market: buy and buyFromETH front-run problem	High	Fixed
07	YieldEscrow: createVoteDelegator() transfers governance tokens from its balance	Medium	Fixed
08	YieldEscrow: clone VoteDelegators to save gas	Medium	Fixed
09	Buyback: misuse of deadline and slippage	Medium	Acknowledged
10	Buyback: price check without commission	Medium	Acknowledged

11	Buyback: token decimals not checked	Medium	Fixed
12	Market: transferProductToken() and transferRewardToken() sweep contract balance	Medium	Acknowledged
13	Market: division before multiplication	Medium	Fixed
14	Market: set uniswapRouter address	Medium	Acknowledged
15	Market: different versions of routers	Medium	Fixed
16	Market: checking token addresses	Medium	Fixed
18	BuybackDepositaryBalanceView: negative issuerInbalance throw	Medium	Fixed
17	ProfitDistributor: error of division	Low	Fixed
18	Buyback: input data not checked	Low	Fixed
19	Market: unindexed events	Low	Fixed
22	BuybackDepositaryBalanceView: checks for zero input data	Low	Fixed
20	Multiple: excessive imports	Low	Fixed
21	Gas optimizations	Low	P/Fixed
22	General recommendations	Low	P/Fixed

#01 VoteDelegator: ownership transfers

High

Each VoteDelegator contract is meant to be deployed by the YieldEscrow and to be used only by the msg.sender. However, VoteDelegator is Ownable [\[3\]](#), and the creator can transfer or renounce the ownership. But the YieldEscrow stores the original creator as the VoteDelegator's owner making it completely broken if the ownership was transferred. Moreover, the user loses the ability to exchange tokens at all because YieldEscrow denies the exchanges when the VoteDelegator contract is active ([L80](#), [L107](#)).

Recommendation: remove the Ownable inheritance and store the owner permanently. Or override Ownable functions to change the `_voteDelegators[]` in YieldEscrow.

Update: the issue was fixed.

#02 ProfitDistributor: direct rewards transfers High

The `transfer()` function of ProfitDistributor contract [L269](#) allows the owner to directly transfer any amount of reward tokens to an arbitrary address. This may break the `notifyRewardAmount()` function [L296](#).

Recommendation: remove `transfer()` and restrict access to this function by putting the contract behind Timelock.

Update: the issue was fixed, the function was removed.

#03 ProfitDistributor: rewardsDuration is monotonically non-increasing High

The `notifyRewardAmount()` function of ProfitDistributor contract [L289](#) updates the `rewardsDuration` variable. With every call from the `exit()` function this variable decreases to 1 in extreme cases. Then a call from the rewardsDistribution contract creates a new rewards period with `rewardsDuration` duration and greatly increased `rewardRate`. An extreme case of `rewardsDuration = 1` will effectively break the contract.

Recommendation: do not update `rewardsDuration` value in `block.number < periodFinish` case.

Update: the issue was fixed.

#04 Market: `changeCumulativeToken() + withdraw()` allows owner to take all the balance High

`changeCumulativeToken()` function [L93](#) allows the owner to set an arbitrary cumulative address. Combining with the `withdraw()` function [L304](#), the owner can transfer the contract's balance of `rewardToken` or `productToken`.

Recommendation: filter the input parameters of the `changeCumulativeToken()` function or transfer the ownership to the Timelock contract.

Update: the issue was fixed.

#05 Market: setting priceFeed problem High

In the function `allowToken()` ([L109](#)) owner can set Price Feed chain. If they set an invalid chain that contains not the Chainlink Price Feed addresses, the contract breaks. Moreover, if the owner sets an invalid chain that consists of Chainlink Price Feed addresses but with an incorrect layout or with incorrect pairs in these contracts, users who enter the Market contract might receive either more or less `productToken` and `rewardToken` tokens than they should.

Recommendation: the severity of this issue could be mitigated by transferring the ownership to the Timelock contract.

Update: the issue was fixed by removing the Oracles completely which creates the new issue [#02](#).

#06 BuybackDepositaryBalanceView: productAmount exceeds balance High

BuybackDepositaryBalanceView contract [L93](#) contains check of amount \geq balance. Error message and transfer in [L97](#) state that it should be amount \leq balance.

Update: the issue was fixed.

#07 Market: buy and buyFromETH front-run problem High

Sandwich attacks can be reproduced on functions `buy()` ([L247](#)) and `buyFromETH()` ([L276](#)) as they don't implement a slippage and deadline mechanism [\[5\]](#).

Recommendation: pass slippage and deadline parameters for swap in function arguments.

Update: `buyFromETH()` function was removed, updated `buy()` uses input parameter for slippage but still calls router with current timestamp as deadline.

#08 YieldEscrow: createVoteDelegator() transfers governance tokens from its balance Medium

The `createVoteDelegator()` function of YieldEscrow contract [L53](#) checks the balance of `msg.sender` and transfers that amount from itself to the newly created instance of VoteDelegator. We doubt this is the desired behavior as the YieldEscrow contract theoretically may have an empty balance at the moment and the calling user may have yBAG tokens from another account that has exchanged them with VoteDelegator contract.

Update: the issue was fixed. Whitelist for transfers of yBAG tokens was added.

#09 YieldEscrow: clone VoteDelegators to save gas Medium

The `createVoteDelegator()` function of YieldEscrow contract [L50](#) creates a new instance of VoteDelegator contract via `new`. Gas could be saved with the `creationCode` or using the Clones library [\[4\]](#).

Update: the issue was fixed.

#10 Buyback: misuse of deadline and slippage Medium

The `buy()` function in [L123](#) of the `UniV2BuybackDepositoryBalanceView` contract calls for Uniswap router's `swapExactTokensForTokens()` method with a wrong deadline and `amountOutMin` parameters [\[2\]](#). Setting these parameters greater than the current timestamp and `amountOut` value calculated by the Library `getAmountOut()` function makes little sense since it will never trigger the corresponding `require()` statements.

#12 Buyback: price check without commission Medium

Checking the price in [L113-114](#) is performed without taking Uniswap commission of 0.003 into account.

#12 Buyback: token decimals not checked Medium

The `buy()` function doesn't work if the value of token decimals is greater than the one of the `stableToken`.

Update: the issue was fixed, the decimals value is checked upon updating.

#13 Market: `transferProductToken()` and `transferRewardToken()` sweep contract balance Medium

`transferProductToken()` and `transferRewardToken()` functions in [L155](#), [L164](#) allow the owner to transfer the contract's balance of `rewardToken` and `productToken` to an arbitrary address.

Recommendation: we are not sure if ownership would be transferred to another contract or not. In the case of EOA as owner, these functions should be removed or have restricted access. In the case of contract-controlled ownership, it should not be able to transfer the ownership further.

Update: functions were merged into the `transfer()` function with the same functionality.

#14 Market: division before multiplication Medium

Errors of division could be reduced ([L198](#), [L205](#)); division should be calculated over the result of multiplication.

Update: the issue was fixed.

#15 Market: set uniswapRouter address Medium

In the function `changeUniswapRouter()` ([L82](#)) owner can change the address of the uniswap router. They can set an invalid address and this may break the contract.

#16 Market: different versions of routers

Medium

In several projects in Binance Smart Chain forks of UniswapV2 minor changes in the router contract were made. They replaced `WETH()` and `swapExactETHForTokens()` functions to `WBNB()` and `swapExactBNBForTokens()` accordingly. If the owner sets the address of the uniswap router to a changed version, functions like `_path()` and `buyFromETH()` will be broken.

Update: the issue was fixed.

#17 Market: checking token addresses

Medium

In the constructor ([L64](#)) and `changeCumulativeToken()` ([L93](#)) function there is no check that `productToken`, `rewardToken` and `cumulative` are the different addresses.

Update: the issue was fixed.

#18 BuybackDepositaryBalanceView: negative issuerInbalance throw

Medium

`issuerInbalance` variable in `BuybackDepositaryBalanceView` contract [L99](#) uses safe subtraction of the balance of the Issuer contract from total supply. However, `rebalance()` function of Issuer contract [L65](#) suggests that total supply could be greater than balance. In that case, `buy()` function will throw without a specific error.

Update: the issue was fixed.

#19 ProfitDistributor: error of division

Low

The `getReward()` function of `ProfitDistributor` contract [L226-227](#) may leave the remainder of the division on the contract's balance.

Update: the issue was fixed.

#20 Buyback: input data not checked

Low

Constructor parameters are not checked.

Update: the issue was fixed.

#21 Market: unindexed events

Low

Events [L40-55](#) contain no indexed parameters.

Update: the issue was fixed.

#22 BuybackDepositaryBalanceView: checks for zero input data Low

BuybackDepositaryBalanceView [L47](#) and Issuer [L38](#), [47](#) lack the zero check of input data. We recommend adding this check to avoid calling functions with unset parameters.

Update: the issue was fixed.

#23 Multiple: excessive imports Low

No need to import SafeEC20 if only certain ones are transferred. Requiring the transfer result is enough and gas-wise.

VoteDelegator: no need in ERC20, SafeERC20 could be used for IERC20 or GovernanceToken. Importing only interfaces for YieldEscrow and GovernanceToken could save gas on deployments.

ProfitDistributor: custom OwnablePausable contract is imported but the Pausable functionality is never used. Importing the original Ownable from OZ should be enough.

Update: the issue was fixed.

#24 Gas optimizations Low

YieldEscrow:

The createVoteDelegator() function in [L53-57](#) performs 3 excessive reads of _voteDelegators[account] variable.

The withdrawFromDelegator() function in [L121](#) creates unused local variable voteDelegator.

governanceToken variable should be declared immutable.

VoteDelegator:

The constructor in [L20](#) is gas inefficient — excessive reads of yieldEscrow and owner() state variables (_owner is shadowed) if the condition is always true.

governanceToken variable should be stored as immutable.

ProfitDistributor:

The exit() function in [L251](#) calls this.notifyRewardAmount() which should be public instead of external or be splitted into external + private.

The notifyRewardAmount() function in [L133](#) uses multiple reads of state variables lockPeriod, unlockPeriod, _stakeAt[account].

The rewardPerToken() function in [L163](#) uses multiple reads of state variable _totalSupply.

The updateReward() modifier in [L108](#) uses multiple reads of state variables.

Buyback (UniV2BuybackDepositaryBalanceView):

token address and tokenDecimals should be declared immutable ([L22](#)).

No need to calculate amountOut as it is checked in the swapExactTokensForTokens(). The required amount in [L114](#) should be used as a parameter in [L123](#).
No need to use safeApprove() for Uniswap router calls ([L116-122](#)).
stableToken and stableTokenDecimals could be stored as variables and updated with the issuer.

Market:

Excessive computations in [L198](#).
Global variables productToken and rewardToken can be set immutable.
In the function changeCumulativeToken() ([L93](#)) in the event, a global variable is passed instead of a local variable.
In the function price() ([L187](#)) there are multiple reads of the global variable productToken.
In the functions buy() ([L247](#)) and buyFromETH() ([L276](#)) global variables productToken and uniswapRouter are read multiple times.
In the function withdraw() ([L304](#)) a global variable cumulative is read multiple times.

BuybackDepositaryBalanceView:

BuybackDepositaryBalanceView imports Treasury.sol contract that is never used.
product variable in BuybackDepositaryBalanceView contract should be declared immutable.
decimals variable in BuybackDepositaryBalanceView contract should be declared constant.

Update: the issues were partially fixed.

#25 General recommendations

Low

We recommend using fixed pragma versions and unmodified OpenZeppelin contracts.
Function balance() in [L138](#) of the UniV2BuybackDepositaryBalanceView contract lacks the NatSpec description.
AgregateDepositaryBalanceView is misspelled.
Typos in comments in BuybackDepositaryBalanceView contract [L27](#), [30](#), [54](#), [56](#), [71](#), [80](#).
Event Buyback in BuybackDepositaryBalanceView contract should index at least address to allow filtering by its value.

Conclusion

6 high severity issues were found. We recommend fixing them before deployment.

Audit includes recommendations on the code improving and preventing potential attacks.

Update: the BondAppétit team has responded to this report. Most of the issues were fixed including all the high ones. Individual responses were added after each item in the [section](#). The updated code is located in the same repository after the [c22106c](#) commit. Updated contracts are deployed to the Ethereum mainnet:

0x377f59Eb8aBd0eFc1F33f45696A0A4Bb51CdA72F	VoteDelegator,
0xdF00204A61Df6F1d241b6751BD34B9a3e4930257	YieldEscrow,
0xD4E861Ead6E81Fe0EB20d0197c72590498E0e944	ProfitDistributor (lockPeriod = 2343110),
0x3C9fEea39ef1D4847813E380bb424677142cf2b1	ProfitDistributor (lockPeriod = 1145375),
0xda47F10c3c8128C9C85312af693c34bcbeFA2057	ProfitDistributor (lockPeriod = 543235),
0x17c665c20693EAD6D5d233ecAf48d37eD87a6c28	UniV2BuybackDepositaryBalanceView,
0x38fDc8C3E8409527F960Df83Cf0Be5274040d9fC	BuybackDepositaryBalanceView.

References

1. [Audit by MixBytes](#)
2. [ERC-20 standard](#)
3. [Ownable.sol by OpenZeppelin](#)
4. [Clones.sol by OpenZeppelin](#)
5. [Uniswap router docs](#)

Appendix A. Issues' severity classification

We consider an issue critical, if it may cause unlimited losses or break the workflow of the contract and could be easily triggered.

High severity issues may lead to limited losses or break interaction with users or other contracts under very specific conditions.

Medium severity issues do not cause the full loss of functionality but break the contract logic.

Low severity issues are typically nonoptimal code, unused variables, errors in messages. Usually, these issues do not need immediate reactions.

Appendix B. List of examined issue types

Business logic overview

Functionality checks

Following best practices

Access control and authorization

Reentrancy attacks

Front-run attacks

DoS with (unexpected) revert

DoS with block gas limit

Transaction-ordering dependence

ERC/BEP and other standards violation

Unchecked math

Implicit visibility levels

Excessive gas usage

Timestamp dependence

Forcibly sending ether to a contract

Weak sources of randomness

Shadowing state variables

Usage of deprecated code