#HashEx
BLOCKCHAIN SECURITY

BlueBit

smart contracts final audit report

April 2022


hashex.org


contact@hashex.org

Contents

1. 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.

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, then 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 (hashex.org).

2. Overview

HashEx was commissioned by the BlueBit team to perform an audit of their smart contracts. The audit was conducted between 22/03/2022 and 25/03/2022.

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 for understanding 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 code is available at the @BluebitFinance/bluebit-contracts GitHub repository and was audited after the commit 6863dae.

Update: the BlueBit team has responded to this report. The updated code is located in the GitHub repository after the commit 539c531.

The audited contracts are deployed to the Aurora Chain mainnet:

BlueBit: 0x947dD92990343aE1D6Cbe2102ea84eF73Bc5790E

BluebitToken: 0x4148d2Ce7816F0AE378d98b40eB3A7211E1fcF0D

FeeDistributor: 0x47E4e267836E6C1057d780130Ed519175f29c2BC

RebatePool: 0x91698AE2851003f384666C3D996FC030349ED915

SwapPath: 0x315A6156f18Ff230a9c427Ded6Ae65a5580f4653

TimeLock: 0x8dF81C799e7281927c3Ef823dD39d601450C90ac

TrisolarisVault:

0x16B8DD191D13C7dbbF00610f1afAeE1405962F56

0xf306f75a6Fc3a52A32FDFe71b0F5b1541AbecDcF

0x5809122175Ee82d6b32FC9D87089ED373c2585F0

0x31714afaA1c61E113BA58A275827564B3440C8F5

0xFE27c7b84ec48D3E87ac300Ddb2704aD4f7eC4c4

0x4699CBd5EA28ac00e2CCe3bb088BA708Bd60842a

TrisolarisVaultV2: 0xE02DbE931c563bf448B5D50a91FdcB899F6589eb

BluebitOwner: 0x1e2192B42a1E6617D36a2E5e4f38d2415BD07135

2.1 Summary

Project name	BlueBit
URL	https://www.bluebit.fi/
Platform	Aurora
Language	Solidity

2.2 Contracts

Name	Address
Bluebit	`0x947dD92990343aE1D6Cbe2102ea84eF73Bc5790E`
BluebitToken	`0x4148d2Ce7816F0AE378d98b40eB3A7211E1fcF0D`

FeeDistributor	0x47E4e267836E6C1057d780130Ed519175f29c2BC
RebatePool	0x91698AE2851003f384666C3D996FC030349ED915
SwapPath	0x315A6156f18Ff230a9c427Ded6Ae65a5580f4653
TrisolarisVault	0x16B8DD191D13C7dbbF00610f1afAeE1405962F56
Lockable	
Ownable	
Pausable	
Token	
Vault	
Timelock	0x8dF81C799e7281927c3Ef823dD39d601450C90ac
TrisolarisVaultV2	0xE02DbE931c563bf448B5D50a91FdcB899F6589eb
BluebitOwner	0x1e2192B42a1E6617D36a2E5e4f38d2415BD07135

3. Found issues

High	6 (23%)
Medium	1 (4%)
Low	14 (54%)
Info	5 (19%)

C1. Bluebit

ID	Severity	Title	Status
C1-01	High	Locked withdraw due to reward token	Resolved
C1-02	Low	Unused functionality	Acknowledged
C1-03	Low	Gas optimization	Partially fixed
C1-04	Low	Unable to withdraw funds	Acknowledged

C2. BluebitToken

ID	Severity	Title	Status
C2-01	High	Unprotected mint by owner	Partially fixed
C2-02	Low	Gas optimization	Resolved

C3. FeeDistributor

ID	Severity	Title	Status
C3-01	High	100% fees	Resolved
C3-02	Low	Gas optimization	Resolved

C4. RebatePool

ID	Severity	Title	Status
C4-01	High	Unprotected withdrawal by owner	Resolved
C4-02	Low	Gas optimization	Partially fixed
C4-03	Info	Unused functionality	Acknowledged
C4-04	Info	Swaps with 100% slippage	Acknowledged
C4-05	Info	Path for swap	Acknowledged

C5. SwapPath

ID	Severity	Title	Status
C5-01	Low	Gas optimization	Resolved
C5-02	Low	Unchecked path	Resolved

C8. Ownable

ID	Severity	Title	Status
C8-01	Low	Gas optimization	Resolved

C10. Token

ID	Severity	Title	Status
C10-01	Low	Gas optimization	Resolved
C10-02	Low	Unused function	Acknowledged

C11. Vault

ID	Severity	Title	Status
C11-01	High	100% fees	Resolved
C11-02	High	Unprotected withdrawal by owner	Resolved
C11-03	Medium	Unlimited withdrawInterval	Resolved
C11-04	Low	Balance discrepancy	Resolved
C11-05	Low	Gas optimization	Partially fixed
C11-06	Info	Swaps with 100% slippage	Acknowledged
C11-07	Info	Path for swap	Acknowledged

C12. Timelock

ID	Severity	Title	Status
C12-01	Low	Gas optimization	Acknowledged

4. Contracts

C1. Bluebit

Overview

The contract allows users to stake their LP tokens and get rewards in BluebitTokens. It does not support tokens with fees (reflect tokens, etc.).

The owner of the contract can disable the accrual of rewards at any time.

Issues

C1-01 Locked withdraw due to reward token

High Resolved

Withdrawals may be blocked for users due to limitations in the reward token (`bluebitToken`).

The BluebitToken contract has the `maxSupply` limit. When the limit is reached, the `withdraw()` function will throw an error due to an attempt to exceed the limit during minting rewards.

Recommendation

This issue can be fixed, for example, by adding try/catch error handling or conditionals (branching) in the code when the `maxSupply` is reached.

Update

There is still one case where a withdrawal can be blocked due to a reward token.

The contract owner can change reward token using the `setBluebitToken()` function. But there is no guarantee that the new reward token will fully comply with the `BluebitToken` interface. So if the new reward token does not have the `maxSupply()`, `totalSupply()`, `mint()` functions, this may lead to a revert when the `withdraw()` function is executed.

The Bluebit ownership was transferred to BluebitOwner contract that can't use

`setBluebitToken()` function.

C1-02 Unused functionality

Low Acknowledged

The function `compound()` updates the pool's `interestRatePerBlock` variable. But this variable is never used in the contracts.

Consider the need for such a variable.

C1-03 Gas optimization

Low Partially fixed

a. The functions `setPool()`, `migratePool()`, `setFactorWeight()`, `setRewardPerBlock()`, `setFeeDistributor()`, `poolLength()`, `depositsOf()`, `sharesOf()`, `balanceOf()`, `getBoostFactor()`, `pendingRewards()`, `deposit()`, `withdraw()`, `harvest()`, `compound()`, `inCaseTokensGetStuck()` can be declared as `external` to save gas.

b. State variables `veToken`, `bluebitToken` can be declared as `immutable` to save gas.

C1-04 Unable to withdraw funds

Low Acknowledged

User funds can be locked in the contract if the owner pauses (locks) the contract because the function `withdraw()` would be inactive.

The same issue can occur as well if the Vault contract is paused.

C2. BluebitToken

Overview

The contract is inherited from the contract Token.

An ERC20 token with open mint for the owner (manager) and max supply value. The token serves as a reward token for the Bluebit contract.

Issues

C2-01 Unprotected mint by owner

High Partially fixed

The owner, in addition to the manager (the Bluebit contract), can mint the token until it reaches the `maxSupply` value. This may result in the Bluebit contract being unable to mint rewards for users.

Recommendation

Restrict minting for the owner.

Update

The contract owner still has ability for unprotected mint. He can change manager address to a controlled one using `setManager()` function. It will allow to mint tokens from new account. The token ownership was transferred to the Timelock contract with 24h minimum delay. Users may check the BluebitToken's ownership themselves.

Developer response

This particular "risk" appears because we allow for the scenario that the BlueBit contract might be subject to an upgrade in the future, including adopting a new product strategy, fixing any possible found bugs from the ImmuneFi Bug Bounty program or other sources.

C2-02 Gas optimization

Low Resolved

The function `mint()` can be declared as `external` to save gas.

C3. FeeDistributor

Overview

The contract allows distributing commissions to specified addresses.

Issues

C3-01 100% fees

High Resolved

The contract owner has the ability to set a fee of 100% for each type of fee. This will cause the users to get nothing when they try to make a call to the `harvests()` functions of the Bluebit contract.

It is also possible that the sum of all commissions will exceed 100%. This will block calls to the `harvest()`, `harvests()`, `deposit()`, `withdraw()` functions of the Bluebit contract.

Recommendation

It is necessary to limit the amount of fees that the owner can set.

C3-02 Gas optimization

Low Resolved

a. All `public` functions of the contract, except the `totalFee()` function, can be declared as `external` to save gas.

b. The calculation in the `totalFee()` function can be simplified to the form: `amount * (fee1+fee2+fee3) / 10000` .

C4. RebatePool

Overview

The contract allows to deposit collected fees on certain dates. A share of such fees can be claimed by the user based on their locked deposits as of a certain date in the veToken contract.

Without documentation, it is impossible to knowingly assess the correctness of the contract code, especially regarding calculation in the `claim()` function.

Issues

C4-01 Unprotected withdrawal by owner

High Resolved

The owner has the ability to withdraw any deposited token that may be intended for contributors using the `inCaseTokensGetStuck()` function at any time.

Recommendation

Remove the ability to use the `inCaseTokensGetStuck()` function for the owner until all tokens have been distributed.

C4-02 Gas optimization

Low Partially fixed

a. The functions `setSwapPath()`, `deposit()`, `lastdays()`, `depositsAt()`, `claimable()`, `claim()`, `inCaseTokensGetStuck()` can be declared as `external` to save gas.

b. The state variables `swapRouter`, `veToken`, `token0`, `token1`, `startPeriod` can be declared as `immutable` to save gas.

C4-03 Unused functionality

Info Acknowledged

The state variable `daily` is never read in the contract.

Consider the need for such a variable.

C4-04 Swaps with 100% slippage

Info Acknowledged

The function `_swap()` can call router with 100% slippage. Transactions sent from this contract may be front-runned resulting in swaps with an undesired rate (sandwich attacks).

C4-05 Path for swap

Info Acknowledged

The `deposit()` function performs the swap of the input token.

It has to be borne in mind that the path for the swap of the tokens must exist in the contract SwapPath. Otherwise, the functions `deposit()`, `FeeDistributor.distribute()`, `Bluebit._withdrawFee()`, `Bluebit.harvests()`, `Bluebit._harvest()` would be blocked.

C5. SwapPath

Overview

The contract stores the trading path for token swaps.

Issues

C5-01 Gas optimization

Low Resolved

The functions `set()`, `get()` can be declared as `external` to save gas.

C5-02 Unchecked path

Low Resolved

The value of the `path` parameter of the `set()` function is not checked.

It is necessary to make validation for the first and last values of the `path` array. For example:

```
require(path[0] == address(tokenIn), "reason message");
require(path[path.length-1] == address(tokenOut), "reason message");
```

C6. TrisolarisVault

Overview

The contract is fully inherited from the contract Vault. Additional issues are indicated in the contract Vault. No issues were found.

The contract has also been deployed with the following addresses:

0x16B8DD191D13C7dbbF00610f1afAeE1405962F56

0xf306f75a6Fc3a52A32FDFe71b0F5b1541AbecDcF

0x5809122175Ee82d6b32FC9D87089ED373c2585F0

0x31714afaA1c61E113BA58A275827564B3440C8F5

0xFE27c7b84ec48D3E87ac300Ddb2704aD4f7eC4c4

0x4699CBd5EA28ac00e2CCe3bb088BA708Bd60842a

C7. Lockable

Overview

The contract provides functionality to restrict access to contract functions. No issues were found.

C8. Ownable

Overview

The contract provides the functionality to pause the functions.

Issues

C8-01 Gas optimization

Low Resolved

The functions `setOwner()`, `setManager()` can be declared as `external` to save gas.

C9. Pausable

Overview

A simplified version of the Pausable token, unlike the common version of OpenZeppelin library. No issues were found.

C10. Token

Overview

A simplified ERC20 contract in contrast to the widespread OpenZeppelin library.

The functions `increaseAllowance()`, `decreaseAllowance()` and transfer hooks are not implemented in the contract.

Issues

C10-01 Gas optimization

Low Resolved

The functions `transfer()`, `allowance()`, `approve()`, `transferFrom()` can be declared as `external` to save gas.

C10-02 Unused function

Low Acknowledged

The internal function `_burn()` is never used in the contract and can be deleted to reduce deployment costs.

C11. Vault

Overview

The contract provides the functionality to block the execution context of the function.

Note: since there is no Farm contract in the audited repository we can't fully audit functions `deposit()`, `withdraw()`, `_farmWithdraw()`.

Issues

C11-01 100% fees

High Resolved

The contract owner has the ability to set the `withdrawFee` of 100% using `setWithdrawFee()` function. This will cause the users to get nothing when they try to make a call to the `withdraw()` functions of the Bluebit contract.

Recommendation

It is necessary to limit the amount of fees that the owner can set.

C11-02 Unprotected withdrawal by owner

High Resolved

The contract owner has the ability to withdraw all deposited funds in the following flow.

The withdraw function can be called only by `manager` (usually Bluebit contract). The owner can change the `manager` address and withdraw all funds to the new address by calling the `withdraw()` function.

Recommendation

It is necessary to restrict the owner's ability to modify the `manager` contract address, to prevent withdrawal by owner.

C11-03 Unlimited withdrawInterval

Medium Resolved

The contract owner has the ability to set any `withdrawInterval` using `setWithdrawInterval()`. This will result in users being required to pay the `withdrawFee` for an unlimited amount of time.

C11-04 Balance discrepancy

Low Resolved

Anyone can temporarily block the work of the `withdraw()` function if they send `swapPair` tokens to the contract. This will cause the `balance` (L127) to exceed `totalSupply`. So at L130, the `withdraw()` function will always fail.

(At the same time, the work of the contract can be restored if the owner withdraws the surplus using the function `inCaseTokensGetStuck()`)

C11-05 Gas optimization

Low Partially fixed

a. All `public` functions of the contract can be declared as `external` to save gas.

b. The state variables `swapRouter`, `swapPair`, `farm`, `farmId` can be declared as `immutable` to save gas.

c. Since the `swapPair` is never changed, the `token0` and `token1` of the `swapPair` are constants. In this case, the use of immutable variables will be cheaper than calls to read variables of another contract (L163-164, L177-178).

C11-06 Swaps with 100% slippage

Info Acknowledged

The function `_swap()` can call router with 100% slippage. Transactions sent from this contract may be front-runned resulting in swaps with an undesired rate (sandwich attacks).

C11-07 Path for swap

Info Acknowledged

The `_harvest()` function performs the swap of the input token.

It should be borne in mind that the path for the swap of the tokens must exist in the contract SwapPath. Otherwise, the functions `_harvest()`, `Bluebit.compound()` will be blocked.

C12. Timelock

Overview

A timelock contract aimed to provide a delay before a transaction can be executed. At first, the transaction is queued, after passing the delay the contract admin can execute it.

Issues

C12-01 Gas optimization

Low Acknowledged

The functions `setDelay()`, `setPendingAdmin()`, `queueTransaction()`, `cancelTransaction()`, `executeTransaction()` can be declared as external to save gas.

C13. TrisolarisVaultV2

Overview

The contract is inherited from contract Vault with overridden functions for the MasterChefV2 contract. Additional issues are indicated in the contract Vault.

C14. BluebitOwner

Overview

The contract allows to perform owner functions of the Bluebit contract.

5. Conclusion

6 high, 1 medium, 14 low, and 5 informational severity issues were found.

5 high, 1 medium, and 7 low severity issues have been resolved in the update. 1 high severity issues was partially resolved.

The reviewed contracts are highly dependent on the owner’s account. Users of the project have to trust the owner and that the owner's account is properly secured. We recommend putting the contract behind a Timelock and multisig wallet. Users should check the contract governance scheme themselves.

At the time of writing the report, the ownership of the contract was not transferred to the TimeLock contract

We strongly suggest adding unit and functional tests for all contracts.

We also recommend using pragma fixed to the version the contracts have been tested and are intended to be deployed with. This helps to avoid deploying using an outdated compiler version and shields from possible bugs in future solidity releases.

This audit includes recommendations on improving the code and preventing potential attacks.

Appendix A. Issues severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Info.** Issues that do not impact the contract operation. Usually, info severity issues are related to code best practices, e.g. style guide.

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

 contact@hashex.org

 @hashexbot

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY