KODA token

smart contract audit report

Prepared for:

koda.finance

Authors: HashEx audit team

October 2021

Contents

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, 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.

Introduction

HashEx was commissioned by the Koda Finance team to perform an audit of their smart contracts. The audit was conducted between September 20 and September 21, 2021.

The audited code was provided directly in .sol files without any documentation or tests.

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.

We found out that the KODA token is based on Reflect.finance [1] with an audit report available [2].

Update: the updated code was deployed to the Binance Smart Chain (BSC): `0x8094e772fA4A60bdEb1DfEC56AB040e17DD608D5`.

Contracts overview

`KODA`

Implementation of ERC20 token standard [5] with the custom functionality of auto-yield by burning tokens and distributing the fees on transfers. Fees are distributed between users (default value 2%) and sent to EOA (default value 8%).

`Ownable`

A modified version of OpenZeppelin's contract with temporary renounce functionality.

Found issues

ID	Title	Severity	Response
01	No limits in reduceBuyFee()	High	Fixed
02	_takeLiquidity() recipient	Medium	Responded
03	setMaxTxnTokens() wrong limit	Medium	Fixed
04	Prohibited transfers from/to dEaD address	Medium	Fixed
05	Fee pause not working with pair transfers	Medium	Fixed
06	Unused parameters & imports	Low	Fixed
07	Missing events	Low	Responded
08	Unreachable code	Low	Fixed
09	Incorrect error message	Low	Fixed
10	Inconsistent comments & typos	Low	Fixed
11	General recommendations	Low	Acknowledged

#01 No limits in reduceBuyFee() High

`reduceBuyFee()` function in L945 doesn't perform any checks on the input value of `buyFeeReductionPercentage` variable and could be set way above 100% blocking all the transfers from the swap pair. Once called, the `buyFeeReduced` boolean variable can't be reverted to false, `buyFeeReducedTime` is not used.

Recommendation: refactor the `buyFeeReduction` functionality from scratch.

Update: the issue was fixed.

#02 _takeLiquidity() recipient Medium

Unlike the SafeMoon token, KODA contract doesn't implement the auto liquify feature but sends the liquidity fee of 8% (default value, up to 10%) to `_kodaLiquidityProviderAddress` account with unclear ownership. `setKodaLiquidityProviderAddress()` function allows the owner to change the recipient of liquidity fee to any address.

Team response: The setKodaLiquidityProviderAddress function is to allow us to change the contract that manages the fee. The fee that is sent to this is fixed at a maximum of 10% regardless, (Reflections plus fee must be equal to or less than 10%). We need the ability to change this provider address in case of upgrading the fee manager contract in the future. This would then alleviate the need to migrate to another version of Koda.

#03 setMaxTxnTokens() wrong limit — Medium

`setMaxTxnTokens()` function in L930 checks the maximum valid input value but updates the `_maxTxnAmount` variable with input value multiplied by 10^decimals.

Update: the issue was fixed. Transaction limit was removed completely.

#04 Prohibited transfers from/to dEaD address — Medium

`_approve()` and `_transfer()` functions check the input addresses to not be equal to `0xdEaD` burn address instead of zero to prevent accidentally locked tokens.
Also, transfer functions of the reviewed contract don't throw error messages for the amounts bigger than the sender's balance (like "`ERC20: transfer amount exceeds allowance`" in OpenZeppelin's ERC20 implementation) which may confuse users.

Update: the issue was fixed.

#05 Fee pause not working with pair transfers — Medium

`transferFeePaused` boolean variable doesn't work for transfers from or to swap pair, see L1106-1107.

Update: the issue was fixed.

#06 Unused parameters & imports — Low

`buyFeeReducedTime`, `summitSwapRouter`, `_burnAddress` variables aren't in use.
`ISummitSwapRouter02`, `ISummitSwapRouter01`, `ISummitSwapPair`, `ISummitSwapFactory` interfaces aren't in use.
SafeMath library should be removed or updated to the 0.4 release version.

Update: the issue was fixed.

#07 Missing events — Low

There're zero custom events implemented. We recommend emitting a specific event every time the system parameter is updated.

Team response: we've decided against implementing this.

#08 Unreachable code Low

`_tokenTransfer()` function contains unreachable condition in L1135.

Update: the issue was fixed.

#09 Incorrect error message Low

Incorrect error message in L875: must be "Account is already included".

Update: the issue was fixed.

#10 Inconsistent comments & typos Low

TODO comments L1, L700. Inconsistent comments in L704, L1091. Typos in L727, 729, 732, 797,918, 927, 962, 1015, 1038, 1143, 1160, 1177.

Update: the issue was fixed.

#11 General recommendations Low

The code is ineffective in terms of computational costs. For example, `removeAllFee()` and `restoreAllFee()` functions write 6 variables to free a transfer from fees. While this is not a big deal in BSC, the code should be refactored before possible deployment to the Ethereum mainnet.

We recommend fixing the pragma version to avoid discrepancies in contract behavior compiled with different Solidity versions.

We discourage changing well-tested libraries contracts such as the Ownable contract from OpenZeppelin. Additional functionality should be implemented via inheritance. SafeMath should be updated to the corresponding release to save gas on unnecessary checks.

Conclusion

The audited contract is a fork of Reflect.finance smart contract with some changes.

1 high severity and 4 medium severity issues were found.

The contract, which implies high risks for token holders as if the owner account is compromised an attacker can break the token functionality completely (for example, by blocking any transfer).

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

Update: most of the issues were fixed including the High severity one. The updated code was deployed to the Binance Smart Chain (BSC):
`0x8094e772fA4A60bdEb1DfEC56AB040e17DD608D5`.

References

1. Reflect.finace github repo
2. Audit report for Reflect.finance
3. ERC-20 standard

Appendix A. Issues' severity classification

We consider an issue critical if it may cause unlimited losses or breaks 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