#HashEx
BLOCKCHAIN SECURITY

FRUIT Token

smart contracts
final audit report

December 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 CryptoFruits team to perform an audit of their smart contract. The audit was conducted between 2022-12-15 and 2022-12-16.

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 0xaDd14CA8E026c4555933D12b568b1310968503a2 in the Binance Smart Chain (BSC).

2.1 Summary

Project name	FRUIT Token
URL	https://cryptofruits.com/index.php
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
FruitToken	`0xaDd14CA8E026c4555933D12b568b1310968503a2`

Ownable
SafeMath
Context
IBEP20

3. Found issues

8
Total issues

●	High	2 (25%)
●	Medium	2 (25%)
●	Low	4 (50%)

C1. FruitToken

ID	Severity	Title	Status
C1-01	● High	Tax percent is unlimited	✓ Resolved
C1-02	● High	Mint is open for owners	✓ Resolved
C1-03	● Medium	Transfer event may mislead explorers	✓ Acknowledged
C1-04	● Medium	Whitepaper nonconformance	✓ Acknowledged
C1-05	● Low	Missing event	✓ Acknowledged
C1-06	● Low	Gas optimizations	✓ Acknowledged

C2. Ownable

ID	Severity	Title	Status
C2-01	● Low	Lack of view functions	✓ Acknowledged

C3. SafeMath

ID	Severity	Title	Status
C3-01	● Low	Outdated version	✓ Acknowledged

4. Contracts

C1. FruitToken

Overview

An ERC-20 token standard implementation with taxable transfers.

Issues

C1-01 Tax percent is unlimited ● High ✓ Resolved

The owner, admin, or partner can update the tax percent (2% by default) to an arbitrary value, meaning a 100% tax is possible, and even an over 100% transfer may break all the transfers.

```
function setTax(uint256 tax) public{
    require(isPartner(msg.sender), 'setAddress: require isPartner(msg.sender)');
    _tax = tax;
}
```

Recommendation

We recommend limiting the max value of the tax. If the re-deployment is not possible, we recommend renouncing 2 or 3 privileged roles, the remaining one must be transferred to a contract with limited max tax value.

Update

The `setTax()` function can be called by the owner, admin, or partner. The ownership was renounced, the MasterChef as admin can't call `setTax()`, and the FruitMaster as partner has 2% limit for new tax value.

C1-02 Mint is open for owners ● High ✓ Resolved

The `mint()` function allows the owner (or admin or partner, who the owner selects) to mint an unlimited number of tokens.

```
function mint(address _to, uint256 _amount) public onlyPartner {
    _mint(_to, _amount);
}
```

Recommendation

Remove the function or renounce all 3 privileged roles.

Update

The ownership was renounced in 0xa024..0ee2 tx, freezing the FruitMaster contract at 0x75aDb4686a2FC4E31Bd41de0812F9c9A9C900338 as the partner and the MasterChef contract at 0x90837ee7fc2faa9f7b55b55150fe2a40283b8d7f as the admin.

C1-03 Transfer event may mislead explorers ● Medium ✓ Acknowledged

The tax amount in every transfer goes to the `_taxRecipient` address, but the emitted event `TaxTransfer()` may cause explorers, e.g. BscScan, to track the token holders incorrectly, since they use `Transfer()` events to accurately track the balances and update the statistics.

C1-04 Whitepaper nonconformance ● Medium ✓ Acknowledged

According to the documentation:

```
In order to balance the crown tokens price and hold the value,
we will buy and burn portion of the game earnings and trading tax,
as well as transfer transaction fees.

With regard to transfer transaction fees,
all FRUIT tokens transfer transactions will be charged for 2%,
including currency exchange, sending FRUIT tokens and redeeming ICO funds, etc.

Besides, there is a 10% trading tax on every purchase and sale.

Among that, 5% will be used for buying back and burning,
4% for marketing funds, and the last 1% for development funds.
```

Only the plain 2% tax is implemented in the code, a 10% one and its distribution are missing.

Recommendation

Conform with the documentation either by updating the code or documentation.

C1-05 Missing event ● Low ✓ Acknowledged

The function `setTax()` changes the transfer tax and does not emit an event. We recommend emitting events for every important value change to simplify off-chain tracking of changes.

C1-06 Gas optimizations ● Low ✓ Acknowledged

1. The `_totalSupply` is zero at the moment of construction, therefore the L404 is redundant.
2. The `_decimals`, `_name`, and `_symbol` variables have double getters: one from public visibility and a separate one for ERC-20 conformance.
3. The `_tax` and `_taxRecipient` variables are read from storage multiple times in the `_transfer()` function.
4. The functions `setTax()`, `setTaxRecipient()`, `increaseAllowance()`, `decreaseAllowance()`, `mint()` and `burn()` can be declared external.
5. The internal function `_burnFrom()` is not used.
6. Variables `_name`, `_decimals`, `_symbol` are set in the constructor and are never changed. They should be defined as `immutable`.

C2. Ownable

Overview

Modified version of the OpenZeppelin contract with 2 additional privileged accounts: `_admin` and `_partner`.

Issues

C2-01 Lack of view functions ● Low ✓ Acknowledged

The `_admin` and `_partner` addresses have private visibility and no getters. Anyone who wants to check the current values of these variables has to use historical events or read directly from the storage.

Recommendation

We recommend adding the viewers for such important variables.

C3. SafeMath

Overview

A safety mathematical operations library forked from OpenZeppelin.

Issues

C3-01 Outdated version ● Low ✓ Acknowledged

Since the 0.8.0 version of Solidity all math operations are checked against under- and overflow by default, thus OpenZeppelin updated their SafeMath version to reduce the gas costs. The used version is outdated and performs unnecessary checks.

C4. Context

Overview

Similar to the OpenZeppelin version. No issues were found.

C5. IBEP20

Overview

An interface for the BEP-20 standard proposed by Binance for their BSC network. No issues were found.

5. Conclusion

2 high, 2 medium, 4 low severity issues were found during the audit. 2 high issues were resolved in the update.

The reviewed contract is dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on code improvement and the prevention of 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.
- **Informational.** Issues that do not impact the contract operation. Usually, informational 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

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx
BLOCKCHAIN SECURITY