DefyFarm

smart contracts audit report

Prepared for:

defy.farm

Authors: HashEx audit team

May 2021

Contents

Disclaimer

Introduction

HashEx was commissioned by the DefyFarm team to perform an audit of DefyFarm's smart contracts. The audit was conducted between May 02 and May 06, 2021.

The audited code is deployed to Binance Smart Chain (BSC):
`0x0acbb2C3D3826B82B17C09e2Dfa605b5279E0C63` DefyCoinV2,
`0x23185fd9b304f03f7BAd458e6DacECA20Ce5Caa1` ImpermenantLossProtection (DefyILP),
`0xB3534770A194226925F90e20e669c8AFbb3B2bff` DefyMaster.
No documentation was provided.

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 DefyCoinV2 token is based on Reflect.finance [1] custom token with an audit report available [2]. DefyMaster is a fork of the MasterChef [3] contract by Sushiswap (audit [4,5]).

Update: DefyFarm team has responded to this report. Individual responses to the high severity issues were added after each item in section.

Contracts overview

`DefyCoinV2`

Implementation of BEP20 token standard with the custom functionality of auto-yield by burning tokens and distributing the fees on transfers.

`DefyMaster`

Farming contract similar to the MasterChef from Sushiswap.

`ImpermenantLossProtection (DefyILP)`

A contract that intends to compensate for the impermanent loss. Interacts with the ApeSwap.

Found issues

ID	Title	Severity	Response
01	BEP20 token standard violation	High	Informed
02	excludeAccount() abuse	High	Informed
03	for() loop in getCurrentSupply()	High	Informed
04	No safeguards on reward parameters	High	Informed
05	onlyFarm modifier includes owner	High	Informed
06	Inconsistent reward model in DefyMaster	High	Informed
07	DefyILP works only for DEFY pools	Medium	Informed
08	getDefyPrice() order in pair	Medium	
09	Hardcoded address of Uniswap router	Medium	
10	token0 & token1 in PoolInfo structure	Medium	
11	Hardcoded addresses WDev & WFarm	Low	
12	Check if sender is zero address	Low	
13	Wrong pragma version	Low	
14	setImpermenantLossProtection() no return	Low	
15	setFarmsender() check for address(0)	Low	
16	BONUS_MULTIPLIER is not constant	Low	
17	Gas savings in includeAccount()	Low	
18	General recommendations	Low	

#01 BEP20 token standard violation High

Implementation of `transfer()` function in DefyCoinV2 L674 does not allow to input zero amount as it's demanded in ERC-20 [6] and BEP-20 [7] standards. This issue may break the interaction with smart contracts that rely on full ERC20 support.

Response from Team DEFY:
It's not an issue in our ecosystem because no one need to transfer 0 amount in any case. Non-zero amount requirement is also implemented in original RFI contract. (we have closely followed the original RFI contract)

#02 excludeAccount() abuse High

The owner of the DefyCoinV2 contract can redistribute part of the tokens from users to a specific account. For this owner can exclude an account from the reward and include it back later. This will redistribute part of the tokens from holders in profit of the included account. The abuse mechanism can be seen in Appendix C. We suggest lock exclusion/inclusion methods by renouncing ownership.

Response from Team DEFY:
We need `exclude()` function to prevent contracts from getting holders reward. After completely stabilizing the system, we'll renounce the ownership as you suggest.

#03 for() loop in getCurrentSupply() High

The mechanism of removing addresses from auto-yielding in DefyCoinV2 implies a loop over excluded addresses for every transfer operation or balance inquiry. This may lead to extreme gas costs up to the block gas limit and may be avoided only by the owner restricting the number of excluded addresses. In an extreme situation with a large number of excluded addresses transaction gas may exceed maximum block gas size and all transfers will be effectively blocked. Moreover, `includeAccount()` function relies on the same `for()` loop which may lead to irreversible contract malfunction.

Response from Team DEFY:
The amount of exclusions required to increase gas cost to a point of unusability is far, far more than we would ever have on exclusions. We have only 4-6 contracts to exclude like liquidity pools, farm and ILP contracts.

`#04 No safeguards on reward parameters` High

There are no restrictions on `defyPerBlock` or `BONUS_MULTIPLIER` in DefyMaster L1121 and L1130 contracts. The owner of the contract can set the very big reward per block and take the full balance of the contract with the front-run transaction.

Response from Team DEFY:
Reward per block depends on the daily volume of the previous day so we can't limit the reward per block. But we will renounce the ownership of this contract as well when we have stabilized that. (reward sending and reward updating will be done by the reward sender contract so owner can not manually update the reward)

`#05 onlyFarm modifier includes owner` High

DefyILP L615 contract contains onlyFarm modifier that includes both DefyMaster's and owner's addresses. This leads to the overpowered owner, especially in the case of `defyTransfer()` function. The owner is allowed to transfer any amount of DefyILP balance.

Response from Team DEFY:
This includes owner, purposefully, in case the owner needs to do any modifications. Specifically, this will be used for potential upgrades to the impermanent loss contract to add more pools (i.e. defy-bnb), and to add pools back in as necessary upon launch of a new ILP. Without this functionality, the LP pools code get wonkey if, say, PCS does another migration to a new version of LP. Owner is also given access to grabbing the Defy out so he may put into the new ILP contract.

`#06 Inconsistent reward model in DefyMaster` High

The DefyMaster contract uses the same reward model as the original SushiSwap's MasterChef (`reward = rewardPerBlock * (delta of block.number)`). At the same time the contract's balance is refilled externally and there are no guarantees that it's balance will be equal to the generated reward. This may lead to a dangerous situation of inconsistency between the expected value from `pendingdefy()` and the real transfer amount.

`#07 DefyILP works only for DEFY pools` Medium

`_getDepositValue()` function in DefyILP L731 contract returns the result of calculations based on `price DEFY/BUSD`. The lack of documentation makes it hard to comprehend this behavior, but we believe that the current realization works only for the DEFY pools from the DefyMaster contract.

Update: Team DEFY is informed of the issue and says not defy-busd pools are not supposed to be added for DefyILP.

Response from Team DEFY:
By design, this impermanent loss contract is only equipped for defy-busd pools. Possible upgrades may include other pools.

`#08 getDefyPrice() order in pair` Medium

Function `getDefyPrice()` in DefyILP L744 depends on the DefyCoinV2 address. If the token was deployed to address which value is bigger than the address of BUSD in BSC, the function would return a wrong result. Developers must take care of that in case of future redeployments.

`#09 Hardcoded address of Uniswap router` Medium

Hardcoded address of Uniswap router in DefyCoinV2 L643. This require statement is useless as the DefyCoinV2 contract is deployed to BSC and the router address is from Ethereum mainnet. We believe it should be the address of ApeSwap router.

`#10 token0 & token1 in PoolInfo structure` Medium

`PoolInfo` structure in DefyMaster L1055 contains token0 and token1. Those are unnecessary as they can be fetched from the `lpToken` address. Besides, setting them separately can lead to mistakes and inconsistencies in the functioning of the contract in case of wrong sorting order.

`#11 Hardcoded addresses WDev & WFarm` Low

Hardcoded addresses of WDev and Wfarm in DefyCoinV2 L502-503. Address hardcoding should be avoided whenever it’s possible.

`#12 Check if sender is zero address` Low

No need to check if sender is zero address in `burn()` and `transfer()` functions of DefyCoinV2 contract.

`#13 Wrong pragma version` Low

DefyCoinV2 contract uses pragma ^0.6.0, but can't be compiled with the 0.6.0 version due to the outdated Address library. Address library is not in use anywhere and should be removed.

`#14 setImpermenantLossProtection() no return` Low

`setImpermenantLossProtection()` function of DefyMaster L1103 should return boolean value, but does not return anything.

`#15 setFarmsender() check for address(0)` Low

`setFarmsender()` function of DefyMaster L1107 should check it for zero address.

`#16 BONUS_MULTIPLIER is not constant` Low

`BONUS_MULTIPLIER` variable in DefyMaster L1076 is named capitalized, but is not declared constant and could be changed with `updateMultiplier()` function.

`#17 Gas savings in includeAccount()` Low

`includeAccount()` function of DefyCoinV2 L652 performs gas savings by removing the element of `_excluded[]` array. It should also delete the corresponding elements of `_isExcluded` and `_tOwned`.

`#18 General recommendations` Low

Code is very inefficient, gas could be saved almost in every major function, especially on token transfers. Excluding 4 addresses in DefiCoinV2 would cost ~500k of gas for `transferStardard()` function. DefyILP has unused code such as `devAddr` and `UserInfo`. DefyCoinV2 contains `_getTaxFee()` and `_getMaxTxAmount()` which are not in use. Impermanent loss is misspelled through all the contracts.

Conclusion

Reviewed contracts are deployed at
`0x0acbb2C3D3826B82B17C09e2Dfa605b5279E0C63`,
`0x23185fd9b304f03f7BAd458e6DacECA20Ce5Caa1`,
`0xB3534770A194226925F90e20e669c8AFbb3B2bff` in BSC. The audited token contract is based on the Reflect.finance model with different fees. Farming contract is based on MasterChef from SushiSwap with additional impermanent loss protection via DefyILP contract.

6 high severity issues were found. DefyFarm team is informed of them and has responded to the issues. You can find the response in updates after corresponding issues.

Issues number 2, 3, 4, 5, 7 can be mitigated if the owner of the contracts is trusted and uses the contracts with extra care.

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

References

1. Reflect.finace github repo
2. Audit report for Reflect.finance
3. SushuSwap’s MasterChef contract
4. SushiSwap audit by PeckShield
5. SushiSwap audit by Quantstamp
6. ERC-20 standard
7. BEP-20 standard
8. Uniswap Router02 on ethscan

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

Appendix C. Hardhat framework test for possible abuse of excludeAccount()

```
const { expect } = require("chai");
const { formatUnits } = ethers.utils;
const formatAmount = (amount) => formatUnits(amount, 8)
describe("DefyV2", function() {
  it("should exclude and include accounts", async function() {
    const [owner, alice] = await ethers.getSigners()
    const Defy = await ethers.getContractFactory("DefyCoinV2");
    const token = await Defy.deploy();
    let totalSupply = await token.totalSupply();
    console.log(`total supply: ${formatAmount(totalSupply)}`)
    let balance = await token.balanceOf(owner.address)
    console.log(`owner balance is: ${formatAmount(balance)}`)

     console.log('excluding...')
    await token.excludeAccount(owner.address)
    await token.transfer(alice.address, totalSupply.div(2))
    totalSupply = await token.totalSupply();
    console.log(`total supply: ${formatAmount(totalSupply)}`)
    balance = await token.balanceOf(owner.address)
    console.log(`owner balance is: ${formatAmount(balance)}`)
    let aliceBalance = await token.balanceOf(alice.address)
    console.log(`alice balance is: ${formatAmount(aliceBalance)}`)

    console.log('including...')
    await token.includeAccount(owner.address)
    balance = await token.balanceOf(owner.address)
    console.log(`owner balance is: ${formatAmount(balance)}`)
    aliceBalance = await token.balanceOf(alice.address)
    console.log(`alice balance is: ${formatAmount(aliceBalance)}`)
    totalSupply = await token.totalSupply();
    console.log(`total supply: ${formatAmount(totalSupply)}`)
  })
});
```

Hardhat framework test output

```
  DefyV2
total supply: 3000000000000000.0
owner balance is: 3000000000000000.0
excluding...
total supply: 2970000000000000.0
owner balance is: 1500000000000000.0
alice balance is: 1378125000000000.0
including...
owner balance is: 1515306122448979.59183673
alice balance is: 1363775510204081.63265306
total supply: 2970000000000000.0
```