#HashEx
BLOCKCHAIN SECURITY

Defy Swap

smart contracts
final audit report

February 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

2. Overview

HashEx was commissioned by the Defy Swap team to perform an audit of their smart contract. The audit was conducted between 28.12.2021 and 13.01.2022.

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 in GitHub repository @defyswap/defyswap (commit 2dd877a) and in the Fantom Network (verified with ftmscan).

Update: Defy Swap team responded to the audit. 1 new contract (SafeDefyOwner) was included in the scope of the audit during the recheck process.

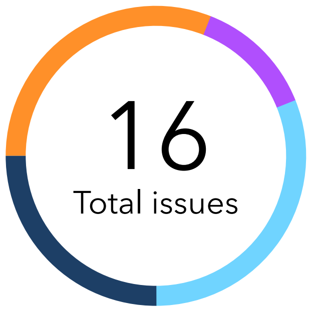
2.1 Summary

Project name	Defy Swap
URL	https://defyswap.finance/
Platform	Fantom Network
Language	Solidity

2.2 Contracts

Name	Address
DefyMaster	0x53e986884c55c9AEDB7f003583f350EE789505D0
DfyToken	0x84b0b7718f8480A9eDa3133Fd385D7EDf2b1d1c4
DefySwapRouter	0x35e11136FA9a734AF6B81658D66519854fB6793c
DefySwapPair	0xAffdbEAE1ec595cba4C262Bdb52A6083aEc2e2a6
DefySwapFactory	0xAffdbEAE1ec595cba4C262Bdb52A6083aEc2e2a6
ImpermanentLossProtection	0x159ff61903ED0336B206c4F2be2C9e220211d910
DefySwapERC20	0xAffdbEAE1ec595cba4C262Bdb52A6083aEc2e2a6
BurnVault	0x9C1c2bCa17A797B579CCb398bC3c46d0D1106Ba7
SafeDefyOwner	0x6d3eb83fa1a6f165cd663325ff8a31840ea091bc

3. Found issues



●	High	5 (31%)
●	Medium	2 (13%)
●	Low	5 (31%)
●	Info	4 (25%)

C3d. DefyMaster

ID	Severity	Title	Status
C3dI23	● High	STUB token logic	✓ Acknowledged
C3dI19	● High	Owner setters abuse	✓ Resolved
C3dI1c	● Medium	BONUS_MULTIPLIER	✓ Resolved
C3dIcf	● Low	Possible wrong ILP address	✓ Acknowledged
C3dI1f	● Low	Gas consumption	✓ Acknowledged
C3dI1e	● Low	Balance check for second reward token	✓ Acknowledged
C3dI22	● Low	Constructor arguments	✓ Acknowledged
C3dI1a	● Info	Reentrancy	✓ Acknowledged
C3dI20	● Info	Token support	✓ Acknowledged
C3dI1b	● Info	Flash-loan vulnerability	✓ Acknowledged

C3c. DfyToken

ID	Severity	Title	Status
C3cI11	● High	setPair problem	✓ Acknowledged
C3cI12	● Medium	DAO functionality	✓ Acknowledged

C41. DefySwapRouter

ID	Severity	Title	Status
C41I18	● Low	Redundant call	✓ Acknowledged

C3f. DefySwapPair

ID	Severity	Title	Status
C3fI13	● High	Tax free tokens	✓ Acknowledged

C3e. DefySwapFactory

ID	Severity	Title	Status
C3eI26	● Info	Floating liquidity fee	✓ Acknowledged

C43. ImpermanentLossProtection

ID	Severity	Title	Status
C43I14	● High	Flash-loan vulnerability	✓ Acknowledged

4. Contracts

C3d. DefyMaster

Overview

This is a farm contract. In this contract, users can stake their tokens (that have a pool in this farm). For their deposits, users get two reward tokens. The first reward token is DfyToken and the second token is determined by the admin.

This farm has the functionality of refunding users if their deposit is decreased in value. This feature works only for stakes that have been deposited for 30 days or more and for pools that have been marked by the admin as refundable. Refunds are coming from ImpermanentLossProtection pool.

Issues

C3dI23 STUB token logic ● High ✓ Acknowledged

Minting of the STUB token in a pool can be abused by a malefactor.

A user can call the `deposit()` function, deposit LP tokens, and get minted STUB tokens on their account. Then they call the `emergencyWithdraw()` function. After all, they get all the tokens back to their account plus STUB tokens for free.

Moreover, the users must be informed that they shouldn't transfer STUB tokens if they want to withdraw their deposit. If they transfer their STUB tokens, they won't be able to call the `withdraw()` function.

Recommendation

STUB tokens shouldn't be used as LP tokens in pools of DefyMaster or be traded on DEXes. Because users can mint an unlimited amount of STUB tokens without a corresponding amount of LP tokens of a pool.

The admin should check for all existing stub tokens in the contract if they are trading anywhere. If they are traded somewhere, any user can attack this pool by unlimited mint and withdraw all funds from it.

STUB tokens should revert all users' transfers or in the documentation for the users, there should be added information about transfers of STUB tokens.

C3dI19 Owner setters abuse ● High ✓ Resolved

In the contract there are several setters that the admin can abuse:

1. Through the `setDFY()` function, the admin can break the `deposit()` and `withdraw()` functions passing an invalid address to this function. Also, the admin can withdraw the funds of the users by changing the address of the DfyToken to a token, that's been deposited into a pool. The same with the `setSecondaryReward()` function.
2. `add()` and `set()` functions contain an external call to the ILP contact with 3 addresses: `_lpToken`, `_token0`, and `_token1`. Wrong token addresses or their order may cause a mistake return value of the ILP contract.

Recommendation

In the `setDFY()` and `setSecondaryReward()` functions there should be a validation for token address. A new token address mustn't be a token, that is in a pool of the contract.

In the `add()` function, there should be a check that a new token doesn't equal a `secondR` address.

We recommend refactoring the ILP contract ignoring input `_token0` and `_token1` parameters, which could be taken from `_lpToken` pair.

Update

The issue is fixed. DefySwap team transferred ownership of the contract (`owner` and `devaddr` variables) to the SafeDefyOwner contract.

C3dI1c BONUS_MULTIPLIER ● Medium ✓ Resolved

If the admin sets a new value of the `BONUS_MULTIPLIER` variable that is higher than 1, the math of the contract will crash. The variable `defyPerSec` is calculated taking into account that `BONUS_MULTIPLIER` equals 1 and if this variable exceeds 1, at the end of the cycle, not all users will get their rewards because a much bigger amount of tokens will have been minted, and mint will fail because of the max total supply in the DfyToken token.

Update

This issue is resolved. DefySwap team will transfer ownership of the contract (`owner` and `devaddr` variables) to the SafeDefyOwner contract.

C3dIcf Possible wrong ILP address ● Low ✓ Acknowledged

Through the

Through the `setImpermanentLossProtection()` the admin can change the address of the ImpermanentLossProtection contract. If the admin passes a wrong new address, `deposit()` and `withdraw()` functions won't work.

C3dI1f Gas consumption ● Low ✓ Acknowledged

If dealing with a big number of pools, the contract will consume a lot of gas and in extreme cases, users won't be able to enter and leave the contract because the gas for the transactions will be bigger than a gas limit in a block.

Affected

functions: massUpdatePools(), updateEmissionRate(), updateSecondReward(), add(), set(), and updatePoolPb().

The owner must pay attention to the number of added pools.

C3dI1e Balance check for second reward token ● Low ✓ Acknowledged

There is no check that the contract holds enough tokens for rewards. Users may lose their rewards by calling any function that updates the user's `rewardDebt` value.

C3dI22 Constructor arguments ● Low ✓ Acknowledged

In the constructor `_initMint` and `MAX_SUPPLY` variables should come from the values of the `defy` contract, not from the arguments.

C3dI1a Reentrancy ● Info ✓ Acknowledged

The `emergencyWithdraw()` and `deposit()` functions can be attacked if a token in a pool gives a user an opportunity to perform a call inside the transfer function.

C3dI20 Token support ● Info ✓ Acknowledged

The contract doesn't support tokens with a rebase. Also, if a token is an RFI token, all the rewards from distribution will be lost on the contract.

C3dI1b Flash-loan vulnerability ● Info ✓ Acknowledged

Because of the ImpermanentLossProtection contract and a bug inside it, a user can withdraw funds from it by a flash-loan.

Recommendation

A solution is described in the section for the ImpermanentLossProtection contract.

Team response

We'll update the ILP contract with the suggestions.

C3c. DfyToken

Overview

The main token of Defy Swap project. Transfers in this token are commissioned. A fee is split into four parts: to a burn vault, to a ILP vault, and to a dev. The admin of the token can exclude some addresses from paying these fees.

Also, the contract has the functionality of DAO, but it is not working because of the logic errors in the code.

Issues

C3cI11 setPair problem ● High ✓ Acknowledged

The admin can call the `setPair()` function with an argument `false` for an lp pair, that has the DFYtoken as one of its tokens. Because of that, the pair may have two problems:

1. If that pair has a `RemoveTaxFree` option on the DFYtoken, the users won't be able to withdraw their liquidity
2. If that pair has a `AddTaxFree` option on the DFYtoken, the users won't be able to add new liquidity

Recommendation

Function `transferFromTaxFree()` should be wrapped into try/catch statement. In case of fail should be called ordinary `transfer()` function.

C3cI12 DAO functionality ● Medium ✓ Acknowledged

The DAO functionality isn't working because of the two reasons:

1. There is no call of the `_moveDelegates()` function in transfers.
2. In the `burn()` function arguments for the `_moveDelegates()` function are in the wrong order. They should be in the following order: `_delegates[msg.sender], address(0), amount`.

C41. DefySwapRouter

Overview

This is a fork of Uniswap's UniswapV2Router02 contract with some functionality for DfyToken for transfers without fees.

Issues

C41I18 Redundant call ● Low ✓ Acknowledged

In the `removeLiquidityETHSupportingFeeOnTransferTokens()` function a call of the `TransferHelper.safeTransfer()` function after if/else block is redundant. Also, the `transfer()` and `transferFrom()` functions of the tokens that don't truly implement the ERC20 standard may fail because the contract will try to transfer zero tokens in this redundant call.

C3f. DefySwapPair

Overview

This is a fork of Uniswap's UniswapV2Pair contract with several changes:

1. The fee to the admin is changed
2. Added some functionality for the DfyToken token. The admin can remove fees on transfer for the DfyToken token.

Issues

C3fI13 Tax free tokens ● High ✓ Acknowledged

The admin can mark a token that does not have the `transferTaxFree()` function as `AddTaxFree` or `RemoveTaxFree` in the pair. Because of that action pair and router will fail to transfer this token.

1. If marked as an `AddTaxFree` token, users won't be able to add new liquidity
2. If marked as an `RemoveTaxFree` token, users won't be able to withdraw their liquidity

Recommendation

The address of the DfyToken token can be stored in the DefySwapFactory contract and in the `setLiqTax()` and the `setRLiqTax()` functions there should be a validation of tokens. Only the DfyToken token can be marked as `true`.

C3e. DefySwapFactory

Overview

This is a fork of Uniswap's UniswapV2Factory contract.

Issues

C3eI26 Floating liquidity fee ● Info ✓ Acknowledged

The liquidity fee obtained by `feeTo` address is equal to the multiplication of `sFee`/30 and liquidity growth. The `feeTo` variable has a restricted setter, meaning `feeToSetter` can change liquidity mint proportion between him and users in the range of [0;1] any time he wants. Though, `sFee` has public visibility and can be checked right before liquidity operations.

C43. ImpermanentLossProtection

Overview

This contract is used to collect a part of fees from the DfyToken token (fee to an ILP vault). This fund is used by the DefyMaster contract to refund users if needed.

Issues

C43I14 Flash-loan vulnerability ● High ✓ Acknowledged

The contract can be attacked by a flash-loan. Users can withdraw all funds through the DefyMaster contract by attacking reserves of the lp pair, that they are staking, manipulating `getDefyPrice()` and `getDepositValue()` return values.

Recommendation

A better practice is to use price oracles. The Uniswap has an example of that [1].

In the ImpermanentLossProtection contract, a pair of the DfyToken token should be selected with another token. This pair will only be used for calculating the price of the DfyToken token through an oracle.

Before getting the price of the DfyToken token the `update()` function in the oracle should be called. Also, in the oracle, the `granularity` variable should be big enough for a `windowSize` variable.

As well the `update()` function in this oracle should be called when a swap in the router is made (if the swap is made in the oracle's pair).

Team response

We'll update the ILP contract with the suggestions.

C40. DefySwapERC20

Overview

This is a fork of Uniswap's UniswapV2ERC20 contract. No issues were found.

C42. BurnVault

Overview

This contract is used to collect part of fees from the DfyToken token (fee to a burn vault). Also is used to burn the DfyToken tokens, that were sent to it. Burn action can perform the admin or the DFYMaster contract in some cases. No issues were found.

Caf. SafeDefyOwner

Overview

The owner of the DefyMaster contract (`owner` and `devaddr` variable). No issues were found.

5. Conclusion

5 high severity issues were found, 1 of them was fixed by implementing the SafeDefyOwner contract. The 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.

This audit includes recommendations on the code improving and preventing potential attacks. We recommend adding tests with coverage of at least 90% with any updates in the future.

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

Appendix C. References

[1] Uniswap's ExampleSlidingWindowOracle contract - https://github.com/Uniswap/v2-periphery/blob/master/contracts/examples/ExampleSlidingWindowOracle.sol

contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx
BLOCKCHAIN SECURITY